



Guida per gli sviluppatori

# AWS Lambda



# AWS Lambda: Guida per gli sviluppatori

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

I marchi e l'immagine commerciale di Amazon non possono essere utilizzati in relazione a prodotti o servizi che non siano di Amazon, in una qualsiasi modalità che possa causare confusione tra i clienti o in una qualsiasi modalità che denigri o discrediti Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà dei rispettivi proprietari, che possono o meno essere affiliati, collegati o sponsorizzati da Amazon.

---

# Table of Contents

|                                                                                                                   |    |
|-------------------------------------------------------------------------------------------------------------------|----|
| Cosa è AWS Lambda? .....                                                                                          | 1  |
| Come si confrontano Lambda Functions e Lambda MicroVMS .....                                                      | 1  |
| Guida alla configurazione dell'agente .....                                                                       | 4  |
| Prerequisiti .....                                                                                                | 4  |
| Codice Claude .....                                                                                               | 4  |
| Installa il plugin aws-core da Agent Toolkit per AWS .....                                                        | 4  |
| Installa il server MCP Serverless .....                                                                           | 5  |
| Codex .....                                                                                                       | 5  |
| Installa il plugin aws-core da Agent Toolkit per AWS .....                                                        | 5  |
| Installa Serverless MCP Server .....                                                                              | 5  |
| Cursore .....                                                                                                     | 6  |
| Installa AWS abilità serverless .....                                                                             | 6  |
| Installa Serverless MCP Server .....                                                                              | 6  |
| Kiro .....                                                                                                        | 6  |
| Installa AWS abilità serverless nella CLI di Kiro .....                                                           | 6  |
| Installa Serverless MCP Server .....                                                                              | 6  |
| Installa Kiro powers (da Kiro IDE) .....                                                                          | 7  |
| GitHub Copilota .....                                                                                             | 7  |
| Installa AWS abilità serverless .....                                                                             | 7  |
| Installa Serverless MCP Server .....                                                                              | 7  |
| Devin Desktop .....                                                                                               | 8  |
| Installa AWS abilità serverless .....                                                                             | 8  |
| Installa Serverless MCP Server .....                                                                              | 8  |
| OpenCode .....                                                                                                    | 8  |
| Installa AWS abilità serverless .....                                                                             | 8  |
| Installa Serverless MCP Server .....                                                                              | 8  |
| Per tutti gli altri agenti compatibili con le competenze degli agenti e la configurazione del server<br>MCP ..... | 9  |
| Installa AWS abilità serverless .....                                                                             | 9  |
| Installa Serverless MCP Server .....                                                                              | 9  |
| Installa competenze Lambda aggiuntive .....                                                                       | 9  |
| Funzioni .....                                                                                                    | 11 |
| Funzioni e funzioni durature .....                                                                                | 12 |
| Come funziona Lambda .....                                                                                        | 12 |

|                                                  |     |
|--------------------------------------------------|-----|
| Funzionalità principali .....                    | 13  |
| Informazioni correlate .....                     | 14  |
| Concetti principali .....                        | 14  |
| Funzioni Lambda e gestori di funzioni .....      | 15  |
| Ambiente di esecuzione Lambda e runtime .....    | 15  |
| Eventi e trigger .....                           | 16  |
| Autorizzazioni e ruoli Lambda .....              | 17  |
| Esecuzione di codice .....                       | 19  |
| Creazione di architetture basate su eventi ..... | 34  |
| Progettazione di un'applicazione .....           | 47  |
| Crea la tua prima funzione .....                 | 55  |
| Registrati per un Account AWS .....              | 55  |
| Creazione della funzione .....                   | 55  |
| Invocazione della funzione .....                 | 62  |
| Eliminazione .....                               | 65  |
| Fasi successive .....                            | 66  |
| App e modelli di esempio .....                   | 67  |
| Elaborazione dei file .....                      | 67  |
| Integrazione del database .....                  | 67  |
| Attività pianificate .....                       | 68  |
| Risorse aggiuntive .....                         | 68  |
| File-processing app .....                        | 69  |
| Scheduled-maintenance app .....                  | 93  |
| Linguaggi di programmazione .....                | 113 |
| Costruire con Node.js .....                      | 114 |
| Costruire con TypeScript .....                   | 192 |
| Compilazione con Python .....                    | 242 |
| Compilazione con Ruby .....                      | 342 |
| Compilazione con Java .....                      | 410 |
| Compilazione con Go .....                        | 524 |
| Compilazione con C# .....                        | 584 |
| Costruire con PowerShell .....                   | 672 |
| Compilazione con Rust .....                      | 687 |
| Runtime .....                                    | 712 |
| Runtime supportati .....                         | 713 |
| Nuovi rilasci di runtime .....                   | 717 |



|                                                           |     |
|-----------------------------------------------------------|-----|
| Policy di deprecazione del runtime .....                  | 717 |
| Modello di responsabilità condivisa .....                 | 718 |
| Utilizzo del runtime dopo la deprecazione .....           | 720 |
| Ricezione di notifiche di ritiro del runtime .....        | 722 |
| Runtime obsoleti .....                                    | 723 |
| Aggiornamenti della versione di runtime .....             | 727 |
| Ottenere dati sulle funzioni in base al runtime .....     | 740 |
| Modifiche runtime .....                                   | 747 |
| API Runtime .....                                         | 751 |
| OS-only tempi di esecuzione .....                         | 760 |
| Repository open source .....                              | 775 |
| Configurazione delle funzioni .....                       | 778 |
| archivi di file .zip .....                                | 781 |
| Immagini di container .....                               | 795 |
| Self-managed Archiviazione del codice S3 .....            | 805 |
| Memoria .....                                             | 811 |
| Archiviazione temporanea .....                            | 814 |
| Set di istruzioni (ARM/x86) .....                         | 817 |
| Timeout .....                                             | 821 |
| Variabili di ambiente .....                               | 824 |
| Collegamento delle funzioni a un VPC .....                | 838 |
| Collegare funzioni alle risorse di un altro account ..... | 854 |
| Accesso Internet per funzioni del VPC .....               | 861 |
| Rete in entrata .....                                     | 886 |
| File system .....                                         | 890 |
| Alias .....                                               | 899 |
| Versioni .....                                            | 907 |
| Tag .....                                                 | 912 |
| Streaming delle risposte .....                            | 916 |
| Endpoint dei metadati .....                               | 927 |
| Richiamo di funzioni .....                                | 932 |
| Invocazione sincrona .....                                | 934 |
| Invocazione asincrona .....                               | 938 |
| Strumento di mappatura dell'origine degli eventi .....    | 952 |
| Filtro eventi .....                                       | 964 |
| Test nella console .....                                  | 980 |

|                                                                        |      |
|------------------------------------------------------------------------|------|
| Stati delle funzioni .....                                             | 984  |
| Tentativi .....                                                        | 988  |
| Rilevamento di un ciclo ricorsivo .....                                | 990  |
| URL delle funzioni .....                                               | 999  |
| Dimensionamento della funzione .....                                   | 1050 |
| Comprendere e visualizzare la simultaneità .....                       | 1051 |
| Calcolo della simultaneità per una funzione .....                      | 1056 |
| Informazioni sulla simultaneità riservata e simultaneità fornita ..... | 1057 |
| Informazioni sulla simultaneità e le richieste al secondo .....        | 1066 |
| Quote di simultaneità .....                                            | 1068 |
| Configurazione della simultaneità riservata .....                      | 1070 |
| Configurazione della simultaneità fornita .....                        | 1075 |
| Comportamento del dimensionamento .....                                | 1086 |
| Monitoraggio della simultaneità .....                                  | 1088 |
| Monitoraggio e debug .....                                             | 1094 |
| Prezzi .....                                                           | 1094 |
| Parametri di funzione .....                                            | 1095 |
| Log delle funzioni .....                                               | 1108 |
| CloudTrail registri .....                                              | 1148 |
| AWS X-Ray .....                                                        | 1157 |
| Informazioni sulle funzioni .....                                      | 1166 |
| Visualizzazione dei parametri dell'applicazione .....                  | 1175 |
| Segnali applicativi .....                                              | 1178 |
| Esecuzione del debug con VS Code .....                                 | 1182 |
| Integrazione con altri servizi .....                                   | 1185 |
| Creazione di un trigger .....                                          | 1186 |
| Elenco dei servizi .....                                               | 1186 |
| Apache Kafka .....                                                     | 1189 |
| Gateway API .....                                                      | 1338 |
| Infrastructure Composer .....                                          | 1368 |
| CloudFormation .....                                                   | 1372 |
| Amazon DocumentDB .....                                                | 1375 |
| DynamoDB .....                                                         | 1414 |
| EC2 .....                                                              | 1478 |
| Elastic Load Balancer (Application Load Balancer) .....                | 1480 |
| Invoca utilizzando uno Scheduler EventBridge .....                     | 1483 |

|                                                    |      |
|----------------------------------------------------|------|
| IoT .....                                          | 1486 |
| Flussi di dati Kinesis .....                       | 1488 |
| Kubernetes .....                                   | 1550 |
| MQ .....                                           | 1552 |
| RDS .....                                          | 1574 |
| S3 .....                                           | 1606 |
| Secrets Manager .....                              | 1662 |
| SQS .....                                          | 1681 |
| Funzioni Step .....                                | 1743 |
| Batch S3 .....                                     | 1753 |
| SNS .....                                          | 1756 |
| Funzioni durevoli .....                            | 1778 |
| Vantaggi principali .....                          | 1779 |
| Quando utilizzare le funzioni durevoli .....       | 1779 |
| Come funziona .....                                | 1780 |
| Fasi successive .....                              | 1782 |
| Concetti di base .....                             | 1783 |
| Creazione di funzioni durature .....               | 1787 |
| Configura funzioni durevoli .....                  | 1815 |
| Funzioni durevoli o Step Functions .....           | 1817 |
| Esempi .....                                       | 1820 |
| Sicurezza e autorizzazioni .....                   | 1846 |
| Crittografia dei dati di esecuzione durevoli ..... | 1853 |
| SDK di esecuzione durevole .....                   | 1869 |
| Richiamo di funzioni durevoli .....                | 1879 |
| Test di funzioni durevoli .....                    | 1902 |
| Monitoraggio delle funzioni durature .....         | 1903 |
| Best practice .....                                | 1909 |
| SnapStart .....                                    | 1910 |
| Casi d'uso .....                                   | 1912 |
| Funzionalità e limitazioni supportate .....        | 1912 |
| Regioni supportate .....                           | 1913 |
| Considerazioni sulla compatibilità .....           | 1913 |
| Prezzi .....                                       | 1914 |
| Attivazione SnapStart .....                        | 1916 |
| Gestione dell'unicità .....                        | 1923 |

|                                                                                                        |      |
|--------------------------------------------------------------------------------------------------------|------|
| Hook di runtime .....                                                                                  | 1930 |
| Monitoraggio .....                                                                                     | 1943 |
| Modello di sicurezza .....                                                                             | 1946 |
| Best practice .....                                                                                    | 1947 |
| Risoluzione dei problemi .....                                                                         | 1953 |
| Isolamento degli inquilini .....                                                                       | 1956 |
| Quando utilizzare la modalità di isolamento dei tenant .....                                           | 1957 |
| Funzionalità e limitazioni supportate .....                                                            | 1957 |
| Supportata Regioni AWS .....                                                                           | 1958 |
| Considerazioni .....                                                                                   | 1958 |
| Prezzi .....                                                                                           | 1958 |
| Modalità di isolamento .....                                                                           | 1958 |
| Abilitare l'isolamento dei tenant .....                                                                | 1960 |
| Richiamo di funzioni con isolamento del tenant .....                                                   | 1963 |
| Accesso tramite codice .....                                                                           | 1967 |
| Monitoraggio .....                                                                                     | 1971 |
| risoluzione dei problemi .....                                                                         | 1973 |
| Istanze gestite .....                                                                                  | 1973 |
| Funzionalità chiave .....                                                                              | 1974 |
| Quando usare le istanze gestite Lambda .....                                                           | 1974 |
| Come funziona .....                                                                                    | 1975 |
| Modello di concorrenza .....                                                                           | 1975 |
| Locazione e isolamento .....                                                                           | 1975 |
| Prezzi .....                                                                                           | 1976 |
| In che modo Lambda Managed Instances differisce dal tipo di elaborazione Lambda<br>(predefinito) ..... | 1976 |
| Fasi successive .....                                                                                  | 1978 |
| Nozioni di base .....                                                                                  | 1978 |
| Concetti principali .....                                                                              | 1985 |
| Runtime .....                                                                                          | 2009 |
| Rete .....                                                                                             | 2034 |
| Monitoraggio .....                                                                                     | 2037 |
| Quote .....                                                                                            | 2054 |
| Best practice .....                                                                                    | 2057 |
| Risoluzione dei problemi .....                                                                         | 2061 |
| Livelli .....                                                                                          | 2070 |

|                                                                     |      |
|---------------------------------------------------------------------|------|
| Come usare i livelli .....                                          | 2072 |
| Livelli e versioni di livelli .....                                 | 2073 |
| Creazione di pacchetti dei livelli .....                            | 2074 |
| Creazione ed eliminazione di livelli .....                          | 2078 |
| Aggiunta di livelli .....                                           | 2081 |
| Strati con CloudFormation .....                                     | 2085 |
| Livelli con AWS SAM .....                                           | 2086 |
| Estensioni .....                                                    | 2086 |
| Ambiente di esecuzione .....                                        | 2087 |
| Impatto su prestazioni e risorse .....                              | 2088 |
| Permissions .....                                                   | 2089 |
| Configurazione delle estensioni .....                               | 2090 |
| Partner con estensioni .....                                        | 2092 |
| API estensioni .....                                                | 2094 |
| API di telemetria .....                                             | 2112 |
| Risoluzione dei problemi .....                                      | 2169 |
| Configurazione .....                                                | 2169 |
| Implementazione .....                                               | 2179 |
| Invocazione .....                                                   | 2186 |
| Esecuzione .....                                                    | 2194 |
| Strumento di mappatura dell'origine degli eventi .....              | 2205 |
| Rete .....                                                          | 2210 |
| Best practice .....                                                 | 2213 |
| Codice della funzione .....                                         | 2213 |
| Configurazione delle funzioni .....                                 | 2215 |
| Scalabilità delle funzioni .....                                    | 2216 |
| Parametri e allarmi .....                                           | 2216 |
| Utilizzo di flussi .....                                            | 2217 |
| Best practice di sicurezza .....                                    | 2219 |
| Applicazioni di esempio .....                                       | 2219 |
| Strumenti di sviluppo .....                                         | 2222 |
| Strumenti di sviluppo locale .....                                  | 2223 |
| Strumenti Infrastructure as Code (IaC) .....                        | 2223 |
| GitHub Azioni e strumenti .....                                     | 2224 |
| Powertools per AWS Lambda .....                                     | 2224 |
| Strumenti per la gestione del flusso di lavoro e degli eventi ..... | 2225 |

|                                                                                                 |      |
|-------------------------------------------------------------------------------------------------|------|
| Sviluppo locale .....                                                                           | 2225 |
| GitHub Azioni .....                                                                             | 2234 |
| Infrastructure as code (IaC) .....                                                              | 2235 |
| Powertools per Lambda AWS .....                                                                 | 2259 |
| Flussi di lavoro ed eventi .....                                                                | 2260 |
| Test in corso .....                                                                             | 2263 |
| Obiettivi aziendali specifici .....                                                             | 2264 |
| Cosa testare .....                                                                              | 2265 |
| Come testare le soluzioni serverless .....                                                      | 2266 |
| Tecniche di test .....                                                                          | 2266 |
| Best practice .....                                                                             | 2273 |
| Problematiche legate ai test locali .....                                                       | 2276 |
| Domande frequenti .....                                                                         | 2278 |
| Risorse e passaggi successivi .....                                                             | 2279 |
| Permissions .....                                                                               | 2280 |
| Ruolo di esecuzione (autorizzazioni per le funzioni per accedere ad altre risorse) .....        | 2282 |
| Autorizzazioni di accesso (autorizzazioni per altre entità di accedere alle tue funzioni) ..... | 2295 |
| Sicurezza, governance e conformità .....                                                        | 2338 |
| Protezione dei dati .....                                                                       | 2339 |
| Uso di ruoli collegati ai servizi .....                                                         | 2347 |
| Identity and Access Management .....                                                            | 2350 |
| Governance .....                                                                                | 2376 |
| Convalida della conformità .....                                                                | 2411 |
| Resilienza .....                                                                                | 2411 |
| Sicurezza dell'infrastruttura .....                                                             | 2412 |
| Protezione dei carichi di lavoro con endpoint pubblici .....                                    | 2413 |
| Firma del codice .....                                                                          | 2415 |
| MicroVM .....                                                                                   | 2424 |
| Come funziona Lambda MicroVMS .....                                                             | 2424 |
| Quando usare Lambda MicroVMS .....                                                              | 2425 |
| Funzionalità principali .....                                                                   | 2426 |
| Concetti principali .....                                                                       | 2426 |
| Concetti chiave .....                                                                           | 2426 |
| Come Lambda costruisce la tua immagine .....                                                    | 2427 |
| Ciclo di vita di MicroVM .....                                                                  | 2428 |
| Crea la tua prima microVM .....                                                                 | 2430 |

|                                                                                     |      |
|-------------------------------------------------------------------------------------|------|
| Prerequisiti .....                                                                  | 2430 |
| Creazione della tua prima immagine MicroVM .....                                    | 2432 |
| Esecuzione della prima MicroVM .....                                                | 2434 |
| Connessione alla tua MicroVM .....                                                  | 2435 |
| Eliminazione .....                                                                  | 2436 |
| Fasi successive .....                                                               | 2436 |
| Immagini MicroVM .....                                                              | 2436 |
| Come Lambda crea un'immagine MicroVM .....                                          | 2437 |
| Dimensionamento delle microVM .....                                                 | 2437 |
| Immagini base MicroVM .....                                                         | 2438 |
| ganci per la creazione di immagini MicroVM .....                                    | 2439 |
| Aggiornamento di un'immagine MicroVM .....                                          | 2441 |
| Stati dell'immagine e stati di costruzione .....                                    | 2442 |
| Variabili di ambiente .....                                                         | 2443 |
| Snapshot .....                                                                      | 2443 |
| Applicazione di patch alle immagini MicroVM .....                                   | 2445 |
| Immagini di base del contenitore .....                                              | 2445 |
| Funzionalità del sistema operativo .....                                            | 2446 |
| Esecuzione di MicroVMS .....                                                        | 2447 |
| Avvio di una MicroVM .....                                                          | 2447 |
| Connessione a una microVM .....                                                     | 2450 |
| Hook del ciclo di vita .....                                                        | 2453 |
| Sospensione e ripristino di MicroVM .....                                           | 2457 |
| Scalabilità e concorrenza .....                                                     | 2458 |
| Terminazione di una MicroVM .....                                                   | 2459 |
| Elenco delle microVMS .....                                                         | 2459 |
| Gestione degli errori .....                                                         | 2459 |
| Rete .....                                                                          | 2460 |
| Panoramica di .....                                                                 | 2461 |
| Connettività in entrata .....                                                       | 2461 |
| Connettività in uscita .....                                                        | 2465 |
| Utilizzo dei connettori di rete in uscita .....                                     | 2465 |
| Utilizzo di Lambda MicroVMS con endpoint VPC di interfaccia (AWS PrivateLink) ..... | 2467 |
| Sicurezza .....                                                                     | 2473 |
| Ruoli IAM .....                                                                     | 2473 |
| Token di autenticazione .....                                                       | 2474 |

|                                                                          |      |
|--------------------------------------------------------------------------|------|
| Token di accesso alla shell .....                                        | 2474 |
| Applicazione di tag alle risorse .....                                   | 2474 |
| Riferimento alle autorizzazioni IAM .....                                | 2475 |
| Formati ARN delle risorse .....                                          | 2476 |
| Esempio: Least-privilege politica per un operatore .....                 | 2476 |
| Monitoraggio .....                                                       | 2477 |
| Registrazione (registri) CloudWatch .....                                | 2477 |
| CloudTrail registrazione .....                                           | 2478 |
| Risoluzione dei problemi .....                                           | 2482 |
| Accesso alla shell .....                                                 | 2482 |
| Risoluzione dei problemi .....                                           | 2482 |
| Errori comuni (creazione di immagini) .....                              | 2484 |
| Risoluzione dei problemi relativi ai connettori di rete .....            | 2485 |
| Best practice .....                                                      | 2486 |
| Progettazione delle applicazioni .....                                   | 2486 |
| Performance .....                                                        | 2487 |
| Sicurezza .....                                                          | 2487 |
| Ottimizzazione dei costi .....                                           | 2487 |
| Monitoraggio .....                                                       | 2488 |
| Integrazioni .....                                                       | 2488 |
| Agenti gestiti Claude .....                                              | 2488 |
| Agenti Cursor Cloud .....                                                | 2492 |
| Lavorare con AWS gli SDK .....                                           | 2499 |
| Esempi di codice .....                                                   | 2501 |
| Nozioni di base .....                                                    | 2504 |
| Hello Lambda .....                                                       | 2505 |
| Informazioni di base .....                                               | 2514 |
| Azioni .....                                                             | 2651 |
| Scenari .....                                                            | 2788 |
| Confermare automaticamente gli utenti noti con una funzione Lambda ..... | 2789 |
| Migrare automaticamente gli utenti noti con una funzione Lambda .....    | 2830 |
| Crea un'API REST per tenere traccia COVID-19 dei dati .....              | 2853 |
| Creazione di una REST API per la libreria di prestiti .....              | 2854 |
| Creazione di un'applicazione di messaggistica .....                      | 2855 |
| Crea un'API REST con integrazione con proxy di funzioni .....            | 2856 |
| Creazione di un'applicazione serverless per gestire foto .....           | 2864 |



|                                                                                                                                        |      |
|----------------------------------------------------------------------------------------------------------------------------------------|------|
| Creazione di un'applicazione di chat websocket .....                                                                                   | 2869 |
| Crea un'applicazione per analizzare il feedback dei clienti .....                                                                      | 2869 |
| Creazione di una dashboard di monitoraggio con il nome della funzione come variabile ....                                              | 2876 |
| Creazione della tua prima funzione serverless .....                                                                                    | 2886 |
| Invocazione di una funzione Lambda da un browser .....                                                                                 | 2898 |
| Trasformare i dati con S3 Object Lambda .....                                                                                          | 2899 |
| Utilizzo di Gateway API per invocare una funzione Lambda .....                                                                         | 2900 |
| Utilizzo di Step Functions per invocare le funzioni Lambda .....                                                                       | 2902 |
| Utilizzo degli eventi pianificati per invocare una funzione Lambda .....                                                               | 2902 |
| Utilizzare l'API Neptune per eseguire query sui dati del grafico .....                                                                 | 2905 |
| Utilizzo delle variabili di proprietà nei dashboard di monitoraggio per monitorare più funzioni serverless .....                       | 2905 |
| Scrivere i dati di attività personalizzate con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito tramite un SDK ..... | 2914 |
| Esempi serverless .....                                                                                                                | 2937 |
| Connessione a un database Amazon RDS in una funzione Lambda .....                                                                      | 2938 |
| Invocare una funzione Lambda da un trigger Kinesis .....                                                                               | 2957 |
| Invocare una funzione Lambda da un trigger DynamoDB .....                                                                              | 2967 |
| Invocare una funzione Lambda da un trigger Amazon DocumentDB .....                                                                     | 2977 |
| Invocare una funzione Lambda da un trigger Amazon MSK .....                                                                            | 2989 |
| Invocazione di una funzione Lambda da un trigger Amazon S3 .....                                                                       | 2999 |
| Richiamo di una funzione Lambda da un trigger Amazon SNS .....                                                                         | 3011 |
| Invocazione di una funzione Lambda da un trigger Amazon SQS .....                                                                      | 3020 |
| Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis .....                                              | 3029 |
| Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB ...                                               | 3042 |
| Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS                                                 | 3053 |
| AWS contributi della comunità .....                                                                                                    | 3063 |
| Creare e testare un'applicazione serverless .....                                                                                      | 3063 |
| Quote di Lambda .....                                                                                                                  | 3066 |
| Calcolo e archiviazione .....                                                                                                          | 3068 |
| Configurazione, implementazione ed esecuzione della funzione .....                                                                     | 3069 |
| Richieste API Lambda .....                                                                                                             | 3072 |
| Funzioni durevoli .....                                                                                                                | 3074 |
| MicroVM .....                                                                                                                          | 3076 |
| Elaborazione e archiviazione .....                                                                                                     | 3076 |
| Immagini e versioni .....                                                                                                              | 3077 |

|                                |           |
|--------------------------------|-----------|
| Per-MicroVM produttività ..... | 3078      |
| Limiti frequenza API .....     | 3078      |
| Altri servizi .....            | 3079      |
| Cronologia dei documenti ..... | 3080      |
| Aggiornamenti precedenti ..... | 3111      |
| .....                          | mmmcxviii |

# Cosa è AWS Lambda?

AWS Lambda è un servizio di elaborazione serverless. Con Lambda, puoi eseguire codice senza effettuare il provisioning o gestire server. Lambda gestisce automaticamente l'infrastruttura sottostante, inclusa la manutenzione del server, il provisioning della capacità, la scalabilità e l'applicazione di patch, in modo che tu possa concentrarti sulla logica dell'applicazione.

Lambda fornisce due primitive di calcolo, ciascuna progettata per diversi modelli di carico di lavoro:

- [Funzioni Lambda](#) : esegue il codice in risposta a eventi o chiamate API senza gestire i server. Scrivi una funzione di gestione, la colleghi a un trigger (API Gateway, Amazon S3, Amazon SQS e oltre 200 altri AWS servizi) e Lambda la esegue. EventBridge Ogni chiamata viene eseguita in modo indipendente senza uno stato condiviso, scalando orizzontalmente per soddisfare la domanda. Lambda gestisce gli ambienti di esecuzione, il ridimensionamento, il routing e la tolleranza ai guasti.
- [Lambda MicroVMS](#) : ambienti di elaborazione isolati con avvio quasi istantaneo e conservazione dello stato per un massimo di 8 ore. Progettato per carichi di lavoro che richiedono un ambiente di elaborazione dedicato per ogni singolo utente o lavoro. Lambda gestisce l'isolamento, la capacità e il networking. La tua applicazione utilizza le API Lambda MicroVMS e gli endpoint HTTPS per connettere ciascuno al proprio ambiente di elaborazione. user/job

Per informazioni sui prezzi, consultare [Prezzi di AWS Lambda](#).

## Come si confrontano Lambda Functions e Lambda MicroVMS

Lambda Functions e Lambda MicroVMS condividono una base serverless comune:

- Nessuna gestione del server: AWS gestisce l'infrastruttura sottostante, l'applicazione di patch alle istanze e la capacità.
- Pay-per-use fatturazione: nessun impegno anticipato. Paghi solo per le risorse utilizzate.
- Rete gestita: entrambi forniscono un accesso alla rete in entrata e in uscita gestito dal servizio.
- Virtualizzazione Firecracker: isolamento tra i carichi di lavoro. VM-level

Pur condividendo queste fondamenta, soddisfano diversi casi d'uso:

|                            | <a href="#">Funzioni Lambda</a>                                                                                             | <a href="#">MicroVMS Lambda</a>                                                                                      |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| Ideale per                 | Request-response o carichi di lavoro basati su eventi (API, elaborazione dati, automazione)                                 | Ambienti persistenti che eseguono codice utente o non attendibile AI-produced                                        |
| Modello di programmazione  | Gestore di funzioni richiamato in un runtime supportato                                                                     | Qualsiasi applicazione: esegui i tuoi file binari, ascolta le porte, usa le funzionalità del sistema operativo Linux |
| Durata                     | Fino a 15 minuti per chiamata; flussi di lavoro in più fasi che durano fino a un anno con Lambda Durable Functions          | Fino a 8 ore per sessione; sospensione e ripresa tra le sessioni                                                     |
| Ambiente di runtime        | Service-provided runtime linguistici; supporto per i runtime forniti dal cliente                                            | Customer-provided immagini MicroVM                                                                                   |
| Rete in entrata            | Richiami diretti o integrazioni di sorgenti di eventi con AWS i servizi; supporto per lo streaming delle risposte           | Accesso in entrata a qualsiasi porta utilizzando i protocolli OSI Layer 7                                            |
| Concurrency (Simultaneità) | Una richiesta per ambiente di esecuzione alla volta                                                                         | Connessioni multiple simultanee per microVM                                                                          |
| Stato dell'ambiente        | Gli ambienti di esecuzione possono essere riutilizzati (avvio a caldo), ma lo stato potrebbe non persistere tra le chiamate | Lo stato della memoria e del disco viene preservato durante la sospensione e ripristinato alla ripresa               |
| Dimensionamento            | Automatico: Lambda crea e distrugge gli ambienti di esecuzione in risposta al traffico                                      | Developer-controlled — puoi creare, sospendere, riprendere e terminare MicroVMS tramite l'API                        |

|               | <a href="#">Funzioni Lambda</a>                    | <a href="#">MicroVMS Lambda</a>                                                                         |
|---------------|----------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| Ciclo di vita | Completamente gestito da Lambda                    | Developer-controlled; politiche di inattività opzionali per la sospensione e la ripresa automatica      |
| Prezzi        | Per-request + del tempo di esecuzione e GB-seconds | Per-second di elaborazione durante l'esecuzione + archiviazione delle istantanee in caso di sospensione |

# Guida alla configurazione dell'agente

Gli agenti di codifica AI possono accelerare lo sviluppo serverless fornendo assistenza intelligente per lo scaffolding delle applicazioni, la configurazione dell'implementazione, il debug e la generazione dell'infrastruttura come codice. Configurando il tuo agente con la giusta combinazione di plugin, competenze e server MCP, lo doti delle competenze necessarie per creare e gestire applicazioni serverless.

## Collabora con il tuo agente di codifica

Per una guida rapida, puoi inserire questo link direttamente nel contesto del tuo agente:

```
https://docs.aws.amazon.com/lambda/latest/dg/samples/aws-lambda-agent-setup.md
```

Scegli il metodo di installazione più adatto al tuo ambiente di sviluppo.

## Prerequisiti

Assicurati che siano presenti i seguenti prerequisiti per l'installazione del plug-in aws-core e del server MCP Serverless:

- [uv installato](#) sul tuo sistema.
- (Facoltativo) Un Account AWS con credenziali IAM configurate sul computer locale. Le credenziali sono necessarie per gli strumenti che eseguono chiamate AWS API ed eseguono script, ma non per cercare documentazione o scoprire competenze. Se non disponi di credenziali configurate, consulta [Configurazione del server AWS](#) MCP per istruzioni dettagliate.

## Codice Claude

Installa il plug-in aws-core, parte di [Agent Toolkit for AWS](#), che raggruppa la configurazione del server AWS MCP e un set selezionato di competenze per gli agenti (inclusa l'abilità serverless). AWS

## Installa il plugin aws-core da Agent Toolkit per AWS

Esegui il comando seguente per installare il plugin:

```
/plugin marketplace add aws/agent-toolkit-for-aws  
/plugin install aws-core@agent-toolkit-for-aws
```

Ciò consente di installare le competenze degli agenti, tra cui l'abilità AWS serverless, gli hook degli agenti e la configurazione del server AWS MCP in un unico passaggio.

## Installa il server MCP Serverless

Sostituiscilo AWS\_PROFILE con il nome del tuo AWS profilo locale.

```
claude mcp add awslabs-aws-serverless-mcp \  
-e AWS_PROFILE=default \  
-e AWS_REGION=us-east-1 \  
-e FASTMCP_LOG_LEVEL=ERROR \  
--scope user \  
-- uvx awslabs.aws-serverless-mcp-server@latest
```

## Codex

### Installa il plugin aws-core da Agent Toolkit per AWS

Esegui il comando seguente per installare il plugin:

```
codex plugin marketplace add aws/agent-toolkit-for-aws
```

Quindi avvia Codex ed esegui /plugins per sfogliare e installare il plug-in aws-core.

## Installa Serverless MCP Server

Sostituiscilo AWS\_PROFILE con il nome del tuo AWS profilo locale.

```
codex mcp add awslabs-aws-serverless-mcp \  
--env AWS_PROFILE=default \  
--env AWS_REGION=us-east-1 \  
--env FASTMCP_LOG_LEVEL=ERROR \  
-- uvx awslabs.aws-serverless-mcp-server@latest
```

# Cursore

## Installa AWS abilità serverless

```
npx skills add https://github.com/aws/agent-toolkit-for-aws --skill aws-serverless --yes --global
```

## Installa Serverless MCP Server

Aggiungi quanto segue a sotto. `.cursor/mcp.json` "mcpServers" Sostituiscilo `AWS_PROFILE` con il nome AWS del tuo profilo locale.

```
"awslabs.aws-serverless-mcp": {
  "command": "uvx",
  "args": ["awslabs.aws-serverless-mcp-server@latest"],
  "env": {
    "AWS_PROFILE": "default",
    "AWS_REGION": "us-east-1",
    "FASTMCP_LOG_LEVEL": "ERROR"
  }
}
```

# Kiro

## Installa AWS abilità serverless nella CLI di Kiro

```
npx skills add https://github.com/aws/agent-toolkit-for-aws --skill aws-serverless --yes --global
```

## Installa Serverless MCP Server

Aggiungi quanto segue a sotto. `~/.kiro/settings/mcp.json` "mcpServers" Sostituiscilo `AWS_PROFILE` con il nome AWS del tuo profilo locale.

```
"awslabs.aws-serverless-mcp": {
  "command": "uvx",
  "args": ["awslabs.aws-serverless-mcp-server@latest"],
  "env": {
```



```
"AWS_PROFILE": "default",
"AWS_REGION": "us-east-1",
"FASTMCP_LOG_LEVEL": "ERROR"
},
"disabled": false
}
```

## Installa Kiro powers (da Kiro IDE)

Installa i seguenti poteri di Kiro che forniscono contesti e strumenti specializzati agli agenti di Kiro su richiesta:

- Serverless Application Model (SAM): installazione tramite Kiro Powers Link. <https://kiro.dev/launch/powers/add/?name=aws-sam>
- Istanze gestite Lambda: installazione tramite [Kiro powerlink](#).
- Lambda Durable Functions: installazione tramite [Kiro powerlink](#).

## GitHub Copilota

### Installa AWS abilità serverless

```
npx skills add https://github.com/aws/agent-toolkit-for-aws --skill aws-serverless --yes --global
```

### Installa Serverless MCP Server

Aggiungi quanto segue a sotto. `.vscode/mcp.json` "mcpServers" Sostituiscilo AWS\_PROFILE con il nome AWS del tuo profilo locale.

```
"awslabs.aws-serverless-mcp": {
  "command": "uvx",
  "args": ["awslabs.aws-serverless-mcp-server@latest"],
  "env": {
    "AWS_PROFILE": "default",
    "AWS_REGION": "us-east-1",
    "FASTMCP_LOG_LEVEL": "ERROR"
  }
}
```

# Devin Desktop

## Installa AWS abilità serverless

```
npx skills add https://github.com/aws/agent-toolkit-for-aws --skill aws-serverless --yes --global
```

## Installa Serverless MCP Server

Aggiungi quanto segue a sotto. `/.codeium/windsurf/mcp_config.json` "mcpServers"  
Sostituiscilo AWS\_PROFILE con il nome AWS del tuo profilo locale.

```
"awslabs.aws-serverless-mcp": {
  "command": "uvx",
  "args": ["awslabs.aws-serverless-mcp-server@latest"],
  "env": {
    "AWS_PROFILE": "default",
    "AWS_REGION": "us-east-1",
    "FASTMCP_LOG_LEVEL": "ERROR"
  }
}
```

# OpenCode

## Installa AWS abilità serverless

```
npx skills add https://github.com/aws/agent-toolkit-for-aws --skill aws-serverless --yes --global
```

## Installa Serverless MCP Server

Aggiungi quanto segue a sotto. `/.config/opencode/opencode.jsonc` "mcpServers"  
Sostituiscilo AWS\_PROFILE con il nome AWS del tuo profilo locale.

```
"awslabs.aws-serverless-mcp": {
  "command": "uvx",
  "args": ["awslabs.aws-serverless-mcp-server@latest"],
  "env": {
    "AWS_PROFILE": "default",
```

```
"AWS_REGION": "us-east-1",  
"FASTMCP_LOG_LEVEL": "ERROR"  
}  
}
```

## Per tutti gli altri agenti compatibili con le competenze degli agenti e la configurazione del server MCP

Per qualsiasi altro agente che supporti il [formato open source delle competenze degli](#) agenti e la configurazione del server MCP, segui questi passaggi:

### Installa AWS abilità serverless

```
npx skills add https://github.com/aws/agent-toolkit-for-aws --skill aws-serverless --yes --global
```

### Installa Serverless MCP Server

Aggiungi il server MCP Serverless al file di configurazione del client MCP del tuo agente in.

"mcpServers" Sostituiscilo AWS\_PROFILE con il nome del tuo profilo locale AWS .

```
{  
  "mcpServers": {  
    "awslabs.aws-serverless-mcp": {  
      "command": "uvx",  
      "args": ["awslabs.aws-serverless-mcp-server@latest"],  
      "env": {  
        "AWS_PROFILE": "default",  
        "AWS_REGION": "us-east-1",  
        "FASTMCP_LOG_LEVEL": "ERROR"  
      }  
    }  
  }  
}
```

### Installa competenze Lambda aggiuntive

L'Agent Toolkit for AWS include competenze serverless specializzate che consentono agli agenti di intelligenza artificiale di conoscere le funzionalità Lambda. Installa queste competenze utilizzando

qualsiasi agente come Claude Code, Cursor, Kiro CLI, GitHub Copilot, Codex, Devin Desktop OpenCode o qualsiasi altro [agente compatibile con il](#) formato open source delle competenze degli agenti per consentire al tuo agente di aiutarti a creare, distribuire e gestire carichi di lavoro utilizzando Lambda MicroVMS, Lambda Managed Instances e Lambda durable functions.

1. Competenza Lambda MicroVMS: Lambda MicroVMS è una primitiva di calcolo serverless che fornisce VM-level isolamento, velocità di avvio e ripristino quasi istantanee e conservazione dello stato per l'esecuzione di utenti o AI-generated codice in ambienti di esecuzione isolati e con stato.

```
npx skills add https://github.com/aws/agent-toolkit-for-aws/tree/main/skills/specialized-skills/serverless-skills/aws-lambda-microvms --yes --global
```

2. Competenza Lambda Managed Instances: con Lambda Managed Instances, puoi eseguire funzioni Lambda sulle tue istanze Amazon EC2 mantenendo la semplicità operativa di Lambda, dandoti accesso a configurazioni di calcolo specializzate ed efficienza dei costi grazie ai vantaggi dei prezzi di Amazon EC2 senza gestire l'infrastruttura.

```
npx skills add https://github.com/aws/agent-toolkit-for-aws/tree/main/skills/specialized-skills/serverless-skills/aws-lambda-managed-instances --yes --global
```

3. Competenza Lambda durable functions: le funzioni Lambda durable consentono di creare applicazioni tolleranti ai guasti e di lunga durata in grado di verificare l'avanzamento, ripristinare automaticamente gli errori e sospendere l'esecuzione per un massimo di un anno in attesa di attività di lunga durata come i processi human-in-the-loop.

```
npx skills add https://github.com/aws/agent-toolkit-for-aws/tree/main/skills/specialized-skills/serverless-skills/aws-lambda-durable-functions --yes --global
```

# AWS Lambda Funzioni



## Tip

Unisciti agli esperti di Serverless per workshop pratici gratuiti per imparare a creare applicazioni Serverless con le migliori pratiche. [Iscriviti ai workshop](#) serverless gratuiti sul sito web di Experience. AWS

AWS Lambda è un servizio di elaborazione che esegue codice senza la necessità di gestire i server. Il codice viene eseguito automaticamente con aumento o riduzione verticale, a prezzi pay-per-use. Per iniziare, consulta [Crea la tua prima funzione](#). Per informazioni sulle [best practice](#), sui linguaggi [di programmazione](#), sull'isolamento [dei](#) tenant e sugli [strumenti per lo sviluppo e l'implementazione](#), consulta gli argomenti correlati di seguito.

Puoi utilizzare Lambda per:

- Elaborazione di file: elabora automaticamente i file quando vengono caricati su Amazon Simple Storage Service. Per ulteriori informazioni, consulta [gli esempi di elaborazione file](#).
- Long-running flussi di lavoro: utilizza funzioni Lambda [durevoli](#) per creare flussi di lavoro statici e in più fasi che possono durare fino a un anno. Perfetto per l'elaborazione degli ordini, i flussi di lavoro di approvazione, i processi integrati e le pipeline di dati complesse che devono ricordare i loro progressi.
- Esempi di operazioni e integrazione del database: rispondi alle modifiche del database e automatizza i flussi di lavoro dei dati. Per ulteriori informazioni, consulta gli [esempi di database](#).
- Attività pianificate e periodiche: esegui operazioni automatizzate secondo una pianificazione regolare utilizzando. EventBridge Per ulteriori informazioni, consulta gli [esempi di attività pianificate](#).
- Elaborazione di flussi: elabora flussi di dati in tempo reale per analisi e monitoraggio. Per ulteriori informazioni, consulta [Flussi di dati Kinesis](#).
- Applicazioni Web: crea app Web scalabili che si adattano automaticamente alla domanda.
- Backend mobili: crea backend API sicuri per applicazioni mobili e Web.
- backend IoT: gestisci richieste di API Web, dispositivi mobili, IoT e di terze parti. Per ulteriori informazioni, consulta [IoT](#).

Per informazioni sui prezzi, consultare [Prezzi di AWS Lambda](#).

## Funzioni e funzioni durature

[Le funzioni Lambda](#) durano fino a 15 minuti e sono ideali per attività basate su eventi come l'elaborazione di richieste API, la gestione dei caricamenti di file o la risposta alle modifiche del database. [Le funzioni durevoli](#) estendono questo modello per i carichi di lavoro che devono durare più a lungo e sopravvivere alle interruzioni. Possono essere eseguiti per un massimo di un anno, controllando automaticamente i loro progressi in modo da riprendere in modo affidabile dopo i guasti. Utilizza funzioni durevoli quando hai bisogno di flussi di lavoro in più fasi, approvazioni dirette o coordinamento tra i servizi per periodi prolungati.

## Come funziona Lambda

Quando si utilizza Lambda, si è responsabili solo del proprio codice. Lambda esegue il codice su un'infrastruttura di elaborazione ad alta disponibilità e gestisce tutte le risorse di calcolo, compresa la manutenzione del server e del sistema operativo, il provisioning della capacità, il dimensionamento automatico e la registrazione di log.

Poiché Lambda è un servizio di elaborazione serverless e basato su eventi, utilizza un approccio di programmazione diverso rispetto alle applicazioni web tradizionali. Nel modello seguente viene illustrato il funzionamento di Lambda:

1. Scrivi e organizzi il codice in [funzioni Lambda](#), che sono gli elementi costitutivi di base che usi per creare un'applicazione Lambda.
2. Puoi controllare la sicurezza e l'accesso tramite le autorizzazioni [Lambda](#), utilizzando i ruoli di [esecuzione](#) per gestire i AWS servizi con cui le tue funzioni possono interagire e quali politiche delle risorse possono interagire con il tuo codice.
3. Le sorgenti e AWS i servizi di eventi [attivano le](#) tue funzioni Lambda, trasmettendo i dati degli eventi in formato JSON, che le tue funzioni elaborano (incluse le mappature delle sorgenti degli eventi).
4. [Lambda esegue il codice](#) con runtime specifici del linguaggio (come Python) in ambienti di esecuzione che includono runtime, livelli Node.js ed estensioni.

 Tip

Per scoprire come creare soluzioni serverless, consulta la [Guida allo sviluppo serverless](#).

## Funzionalità principali

Configura, controlla e implementa applicazioni sicure:

- [Variabili di ambiente](#) modifica il comportamento delle applicazioni senza nuove implementazioni di codice.
- [Versioni](#) testa nuove funzionalità in modo sicuro mantenendo ambienti di produzione stabili.
- [Livelli](#) ottimizza il riutilizzo e la manutenzione del codice condividendo componenti comuni tra più funzioni.
- [Firma del codice](#) applica la conformità alla sicurezza assicurando che solo il codice approvato raggiunga i sistemi di produzione.

Scalabilità e prestazioni affidabili:

- I [controlli di simultaneità e scalabilità](#) gestiscono con precisione la reattività delle applicazioni e l'utilizzo delle risorse durante i picchi di traffico.
- [SnapStart](#) riduce significativamente i tempi di avvio a freddo. Lambda SnapStart è in grado di fornire prestazioni di avvio inferiori al secondo, in genere senza modifiche al codice della funzione.
- [Streaming delle risposte](#) ottimizza le prestazioni delle funzioni fornendo payload di grandi dimensioni in modo incrementale per l'elaborazione in tempo reale.
- [Immagini di container](#) impacchetta le funzioni con dipendenze complesse utilizzando flussi di lavoro container.

Collega e integra senza problemi:

- Le [reti VPC](#) proteggono risorse sensibili e servizi interni.
- Integrazione [File system](#) che condivide dati persistenti e gestisce le operazioni con stato durante le invocazioni di funzione.
- [URL delle funzioni](#) crea API ed endpoint rivolti al pubblico senza servizi aggiuntivi.
- [Estensioni](#) aumenta le funzioni con strumenti operativi, di monitoraggio e di sicurezza.

## Informazioni correlate

- Per ulteriori informazioni sul funzionamento di Lambda, consulta [Come funziona Lambda](#).
- Per iniziare a utilizzare Lambda, consulta [Crea la tua prima funzione Lambda](#).
- Per un elenco di applicazioni di esempio, vedere [Guida introduttiva ad applicazioni e modelli di esempio](#).

## Come funziona Lambda

Le funzioni Lambda sono gli elementi costitutivi di base utilizzati per creare applicazioni Lambda. Per scrivere funzioni, è essenziale comprendere i concetti e i componenti fondamentali che costituiscono il modello di programmazione Lambda. Questa sezione ti guida attraverso gli elementi fondamentali che devi conoscere per iniziare a creare applicazioni serverless con Lambda.

- [Funzioni Lambda e gestori di funzioni](#)- Una funzione Lambda è un piccolo blocco di codice che viene eseguito in risposta agli eventi. Le funzioni sono gli elementi costitutivi di base utilizzati per creare applicazioni. I gestori di funzioni sono il punto di ingresso per gli oggetti evento elaborati dal codice funzione Lambda.
- [Ambiente di esecuzione Lambda e runtime](#): gli ambienti di esecuzione Lambda gestiscono le risorse necessarie per eseguire la funzione. I runtime sono gli ambienti specifici della lingua in cui vengono eseguite le funzioni.
- [Eventi e trigger](#)- Altri Servizi AWS possono richiamare le tue funzioni in risposta a eventi specifici.
- [Autorizzazioni e ruoli Lambda](#)- Controlla chi può accedere alle tue funzioni e con quali altre Servizi AWS funzioni possono interagire.

### Tip

Se vuoi iniziare a comprendere lo sviluppo serverless più in generale, consulta [Informazioni sulla differenza tra sviluppo tradizionale e serverless](#) nella Guida per gli sviluppatori serverless AWS .



## Funzioni Lambda e gestori di funzioni

In Lambda, le funzioni sono gli elementi costitutivi fondamentali utilizzati per creare applicazioni. Una funzione Lambda è un pezzo di codice che viene eseguito in risposta a eventi, ad esempio un utente che seleziona un pulsante su un sito Web o un file che viene caricato in un bucket Amazon Simple Storage Service (Amazon S3). Grazie a funzioni durevoli, il codice può sospendere l'esecuzione tra una fase e l'altra, mantenendone lo stato automaticamente, rendendole ideali per flussi di lavoro di lunga durata come l'elaborazione degli ordini o la moderazione dei contenuti. Puoi pensare a una funzione come a una sorta di programma autonomo con le seguenti proprietà.

Un gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando una funzione viene eseguita in risposta a un evento, Lambda esegue il gestore delle funzioni. I dati sull'evento che ha causato l'esecuzione della funzione vengono passati direttamente al gestore. Sebbene il codice di una funzione Lambda possa contenere più di un metodo o una funzione, le funzioni Lambda possono avere un solo gestore.

Per creare una funzione Lambda, crea un pacchetto di implementazione contenente il codice funzione e le rispettive dipendenze. Lambda supporta due tipi di pacchetti di implementazione: [archivi di file .zip](#) e [immagini di container](#).

- Una funzione ha un lavoro o uno scopo specifico
- Vengono eseguite solo quando necessario in risposta a eventi specifici
- Smettono automaticamente di funzionare al termine

## Ambiente di esecuzione Lambda e runtime

Le funzioni Lambda vengono eseguite all'interno di un [ambiente di esecuzione](#) sicuro e isolato che Lambda gestisce per te. Per funzioni [durature](#), l'ambiente di esecuzione include componenti aggiuntivi per la gestione dello stato e il coordinamento del flusso di lavoro. L'ambiente di esecuzione gestisce i processi e le risorse necessari per eseguire la funzione. Quando una funzione viene invocata per la prima volta, Lambda crea un nuovo ambiente di esecuzione in cui eseguire la funzione. Al termine dell'esecuzione della funzione, Lambda non interrompe immediatamente l'ambiente di esecuzione; se la funzione viene invocata nuovamente, Lambda può riutilizzare l'ambiente di esecuzione esistente.

L'ambiente di esecuzione Lambda contiene inoltre un runtime, un ambiente specifico al linguaggio che inoltra le informazioni sugli eventi e le risposte tra Lambda e la funzione. Lambda offre una serie di [runtime gestiti](#) per i linguaggi di programmazione più diffusi, oppure puoi crearne di tuoi.

Per i runtime festiti, Lambda applica automaticamente aggiornamenti e patch alle funzioni usando il runtime.

## Eventi e trigger

Puoi anche invocare una funzione Lambda direttamente utilizzando la console Lambda, [AWS CLI](#) o uno dei [Software Development Kit \(SDK AWS\)](#). È più comune in un'applicazione di produzione che la funzione venga richiamata da un'altra Servizio AWS in risposta a un particolare evento. Ad esempio, potresti voler eseguire una funzione ogni volta che un elemento viene aggiunto a una tabella Amazon DynamoDB.

Per fare in modo che la tua funzione risponda agli eventi, configura un trigger. Un trigger collega la funzione a un'origine eventi; la funzione può avere più trigger. Quando si verifica un evento, Lambda riceve i dati dell'evento come documento JSON e li converte in un oggetto che il codice può elaborare. Puoi definire il seguente formato JSON per il tuo evento; il runtime Lambda converte questo JSON in un oggetto prima di passarlo al gestore della funzione.

Example evento Lambda personalizzato

```
{
  "Location": "SEA",
  "WeatherData": {
    "TemperaturesF": {
      "MinTempF": 22,
      "MaxTempF": 78
    },
    "PressuresHPa": {
      "MinPressureHPa": 1015,
      "MaxPressureHPa": 1027
    }
  }
}
```

I servizi di streaming e coda come Amazon Kinesis o Amazon SQS utilizzano una mappatura [???](#) dell'origine degli eventi anziché un trigger standard. Gli strumenti di mappatura dell'origine degli eventi analizzano l'origine alla ricerca di dati nuovi, raggruppano i record in batch e invocano quindi la funzione con gli eventi in batch. Per ulteriori informazioni, consulta [In che modo gli strumenti di mappatura dell'origine degli eventi differiscono dai trigger diretti](#).

Per capire come funziona un trigger, inizia completando il tutorial [Uso di un trigger Amazon S3](#) oppure, per una panoramica generale sull'uso dei trigger e le istruzioni sulla creazione di un trigger utilizzando la console Lambda, consulta [Integrazione con altri servizi](#).

## Autorizzazioni e ruoli Lambda

Per Lambda, è necessario configurare due tipi principali di [autorizzazioni](#):

- Autorizzazioni necessarie alla tua funzione per accedere ad altre Servizi AWS
- Autorizzazioni necessarie ad altri utenti per accedere alla tua funzione Servizi AWS

Le sezioni seguenti descrivono entrambi questi tipi di autorizzazione e illustrano le best practice per applicare le autorizzazioni di privilegio minimo.

### Autorizzazioni per le funzioni di accesso ad altre AWS risorse

Le funzioni Lambda spesso devono accedere ad altre AWS risorse ed eseguire azioni su di esse. Ad esempio, una funzione può leggere elementi da una tabella DynamoDB, archiviare un oggetto in un bucket S3 o scrivere in una coda Amazon SQS. Per concedere alle funzioni le autorizzazioni necessarie per eseguire queste azioni, puoi utilizzare un [ruolo di esecuzione](#).

Un ruolo di esecuzione Lambda è un tipo speciale di [ruolo AWS Identity and Access Management \(IAM\)](#), un'identità che crei nel tuo account a cui sono associate autorizzazioni specifiche definite in una policy.

Ogni funzione Lambda deve avere un ruolo di esecuzione e un singolo ruolo può essere utilizzato da più di una funzione. Quando viene invocata una funzione, Lambda assume il ruolo di esecuzione della funzione e riceve l'autorizzazione a intraprendere le azioni definite nella policy del ruolo.

Quando crei una funzione nella console Lambda, Lambda crea automaticamente un ruolo di esecuzione per la funzione. La policy del ruolo fornisce alla tua funzione le autorizzazioni di base per scrivere gli output dei log su Amazon Logs. CloudWatch Per autorizzare la tua funzione a eseguire azioni su altre AWS risorse, devi modificare il ruolo per aggiungere le autorizzazioni aggiuntive.

Il modo più semplice per aggiungere autorizzazioni è utilizzare una policy AWS [gestita](#). Le policy gestite vengono create e amministrate da AWS e forniscono autorizzazioni per molti casi d'uso comuni. Ad esempio, se la tua funzione esegue operazioni CRUD su una tabella DynamoDB, puoi aggiungere la [AmazonDynamoDBFullAccess](#) policy al tuo ruolo.

## Autorizzazioni per altri utenti e risorse per accedere alla tua funzione

Per concedere altre Servizio AWS autorizzazioni per accedere alla tua funzione Lambda, usi una policy basata sulle risorse. [???](#) In IAM, le policy basate sulle risorse sono collegate a una risorsa (in questo caso, la funzione Lambda) e definiscono chi può accedere alla risorsa e quali azioni può eseguire.

Affinché un altro Servizio AWS possa richiamare la tua funzione tramite un trigger, la politica basata sulle risorse della tua funzione deve concedere a quel servizio l'autorizzazione a utilizzare l'azione. `lambda:InvokeFunction` Se crei il trigger utilizzando la console, Lambda aggiunge automaticamente questa autorizzazione per te.

Per concedere il permesso ad altri AWS utenti di accedere alla tua funzione, puoi definirlo nella politica basata sulle risorse della tua funzione esattamente come per un'altra risorsa. Servizio AWS Puoi anche utilizzare una [policy basata sull'identità](#) associata all'utente.

## Best practice per le autorizzazioni Lambda

Quando imposti le autorizzazioni usando le policy IAM, la [best practice di sicurezza](#) consiste nel concedere le autorizzazioni richieste per eseguire un'attività. Ciò è noto come principio di privilegio minimo. Per iniziare a concedere le autorizzazioni per la tua funzione, puoi scegliere di utilizzare una policy gestita. AWS Le policy gestite possono essere il modo più rapido e semplice per concedere le autorizzazioni per eseguire un'attività, ma possono includere anche altre autorizzazioni non necessarie. Man mano che passi dallo sviluppo iniziale al test e alla produzione, ti consigliamo di ridurre le autorizzazioni solo a quelle necessarie definendo [policy personalizzate gestite dal cliente](#).

Lo stesso principio si applica quando si concedono autorizzazioni per accedere alla tua funzione utilizzando una policy basata sulle risorse. Ad esempio, se desideri autorizzare Amazon S3 a richiamare la tua funzione, la procedura migliore è limitare l'accesso a singoli bucket, o bucket in particolare Account AWS, anziché concedere autorizzazioni generiche al servizio S3.

# Esecuzione di codice con Lambda

Quando scrivi una funzione Lambda, crei codice che viene eseguito in un ambiente serverless unico. Comprendere come Lambda esegue effettivamente il codice implica due aspetti chiave: il modello di [programmazione](#) che definisce il modo in cui il codice interagisce con Lambda e il ciclo di vita dell'ambiente di esecuzione che determina il modo in cui Lambda gestisce l'ambiente di runtime del codice.

## Il modello di programmazione Lambda

Il modello di programmazione funziona come un insieme comune di regole per il funzionamento di Lambda con il codice, indipendentemente dal fatto che tu stia scrivendo in Python, Java o qualsiasi altro linguaggio supportato. Il modello di programmazione include il runtime e il gestore.

Per le funzioni standard:

1. Lambda riceve un evento.
2. Lambda utilizza il runtime per preparare l'evento in un formato utilizzabile dal codice.
3. Il runtime invia l'evento formattato al gestore.
4. Il tuo gestore elabora l'evento utilizzando il codice che hai scritto.

Essenziale per questo modello è il gestore al quale cui Lambda invia gli eventi che devono essere elaborati dal codice. Consideralo come il punto di ingresso per il tuo codice. Quando Lambda riceve un evento, lo trasmette al gestore insieme ad alcune informazioni di contesto. Il gestore esegue quindi il codice per elaborare questi eventi: ad esempio, potrebbe leggere un file quando viene caricato su Amazon S3, analizzare un'immagine o aggiornare un database. Una volta che il codice ha terminato l'elaborazione di un evento, il gestore è pronto per elaborare quello successivo.

## Il modello di esecuzione Lambda

Mentre il modello di programmazione definisce il modo in cui Lambda interagisce con il codice, l'ambiente di esecuzione è il luogo in cui Lambda esegue effettivamente la funzione: è uno spazio di calcolo sicuro e isolato creato appositamente per la funzione.

Ogni ambiente segue un ciclo di vita:

1. Inizializzazione: configurazione dell'ambiente e caricamento del codice
2. Richiamo: esecuzione singola del codice funzionale

### 3. Arresto: pulizia dell'ambiente

Questo ambiente gestisce aspetti importanti dell'esecuzione della funzione. Fornisce memoria alla funzione, insieme a una directory `/tmp` per l'archiviazione temporanea.

## Comprendere il modello di programmazione Lambda

Lambda offre due modelli di programmazione: funzioni standard che durano fino a 15 minuti e Durable Functions che possono durare fino a un anno. Sebbene entrambi condividano concetti fondamentali, Durable Functions aggiunge funzionalità per flussi di lavoro di lunga durata e con stato.

Lambda fornisce un modello di programmazione comune a tutti i tempi di esecuzione. Il modello di programmazione definisce l'interfaccia tra il codice e il sistema Lambda. Comunica a Lambda il punto di ingresso alla funzione definendo un gestore nella configurazione della funzione. Il runtime trasferisce al gestore gli oggetti che contengono l'evento di chiamata e il contesto, ad esempio il nome della funzione e l'ID della richiesta.

Per Durable Functions, il gestore riceve anche un `DurableContext` oggetto che fornisce:

- Funzionalità di checkpoint tramite `step ()`
- Gestione dello stato di attesa tramite `wait ()` e `wait ForCallback ()`
- Persistenza automatica dello stato tra le invocazioni

Quando l'handler termina l'elaborazione del primo evento, il runtime ne invia un altro. Per Durable Functions, il gestore può sospendere l'esecuzione tra una fase e l'altra e Lambda salva e ripristina automaticamente lo stato quando la funzione riprende. La classe della funzione rimane in memoria, quindi i client e le variabili dichiarate al di fuori del metodo del gestore nel codice di inizializzazione possono essere riutilizzati. Per risparmiare tempo di elaborazione sugli eventi successivi, crea risorse riutilizzabili come i client SDK AWS durante l'inizializzazione. Una volta inizializzata, ogni istanza della tua funzione può elaborare migliaia di richieste.

La funzione ha anche accesso allo spazio di archiviazione locale nella directory `/tmp`, una cache transitoria che può essere utilizzata per più invocazioni. Per ulteriori informazioni, consulta [Ambiente di esecuzione](#).

Quando la [traccia AWS X-Ray](#) è abilitata, il runtime registra sottosegmenti separati per l'inizializzazione e l'esecuzione.

Il runtime acquisisce l'output di registrazione dalla tua funzione e lo invia ad Amazon Logs. CloudWatch Oltre a registrare l'output della funzione, il runtime registra anche le voci all'avvio e alla fine dell'invocazione. Questo include un log del report con l'ID della richiesta, la durata fatturata, la durata di inizializzazione e altri dettagli. Se la funzione genera un errore, il runtime restituisce tale errore all'invoker.

#### Note

La registrazione è soggetta alle quote di Logs [CloudWatch](#) . È possibile perdere i dati di log a causa del throttling o, in alcuni casi, quando un'istanza della funzione viene interrotta.

Differenze chiave per le funzioni durevoli:

- Lo stato viene mantenuto automaticamente tra i passaggi
- Le funzioni possono sospendere l'esecuzione senza consumare risorse
- I passaggi vengono ritentati automaticamente in caso di errore
- I progressi vengono monitorati tramite checkpoint

Lambda dimensiona la funzione eseguendone altre istanze in base all'aumento della domanda e interrompendo le istanze in base alla diminuzione della domanda. Questo modello porta a variazioni nell'architettura delle applicazioni, come ad esempio:

- Salvo diversa indicazione, le richieste in entrata possono essere elaborate fuori ordine o simultaneamente.
- Non fare affidamento sulla durata delle istanze della funzione, ma archivia lo stato dell'applicazione in altri servizi.
- Utilizza lo storage locale e gli oggetti a livello di classe per migliorare le prestazioni, mantenendo al minimo le dimensioni del pacchetto di distribuzione e la quantità di dati trasferiti nell'ambiente di esecuzione.

Per un'introduzione pratica al modello di programmazione preferito, consulta i seguenti capitoli.

- [Creazione di funzioni Lambda con Node.js](#)
- [Compilazione di funzioni Lambda con Python](#)
- [Compilazione di funzioni Lambda con Ruby](#)

- [Compilazione di funzioni Lambda con Java](#)
- [Compilazione di funzioni Lambda con Go](#)
- [Compilazione di funzioni Lambda con C#](#)
- [Creazione di funzioni Lambda con PowerShell](#)

## Comprendere il ciclo di vita dell'ambiente di esecuzione Lambda

Gli ambienti di esecuzione Lambda supportano sia le funzioni standard (fino a 15 minuti) che le funzioni durature (fino a un anno). Sebbene entrambi condividano lo stesso ciclo di vita di base, Durable Functions aggiunge funzionalità di gestione dello stato per flussi di lavoro di lunga durata.

Lambda richiama la funzione in un ambiente di esecuzione, che fornisce un ambiente di runtime sicuro e isolato. L'ambiente di esecuzione gestisce le risorse necessarie per eseguire la funzione. L'ambiente di esecuzione fornisce inoltre il supporto del ciclo di vita per il runtime della funzione e per tutte le [estensioni esterne](#) associate alla funzione.

Per Durable Functions, l'ambiente di esecuzione include componenti aggiuntivi per:

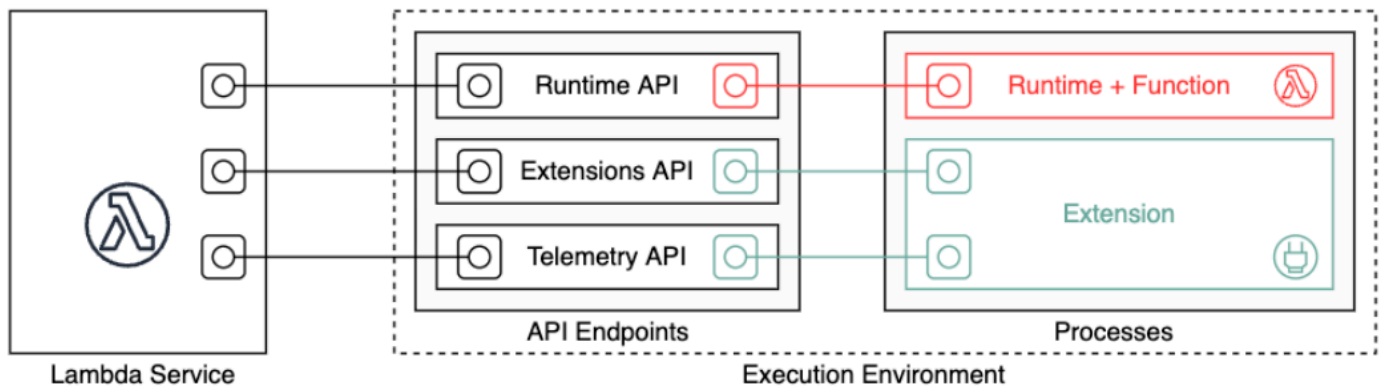
- Persistenza dello stato tra i passaggi
- Gestione dei checkpoint
- Attendi il coordinamento dello stato
- Monitoraggio dei progressi

### Ambiente di esecuzione delle istanze gestite Lambda

Se utilizzi [Lambda Managed Instances](#), l'ambiente di esecuzione presenta differenze importanti rispetto alle funzioni Lambda (predefinite). Le istanze gestite supportano invocazioni simultanee, utilizzano un modello di ciclo di vita diverso e vengono eseguite su un'infrastruttura di proprietà del cliente. Per informazioni dettagliate sull'ambiente di esecuzione delle istanze gestite, consulta [Comprendere l'ambiente di esecuzione delle istanze gestite Lambda](#)

Il runtime della funzione comunica con Lambda e usa l'[API Runtime](#). Le estensioni comunicano con Lambda utilizzando le [API estensioni](#). Le estensioni possono anche ricevere messaggi di log e altra telemetria dalla funzione tramite l'[API di telemetria](#).





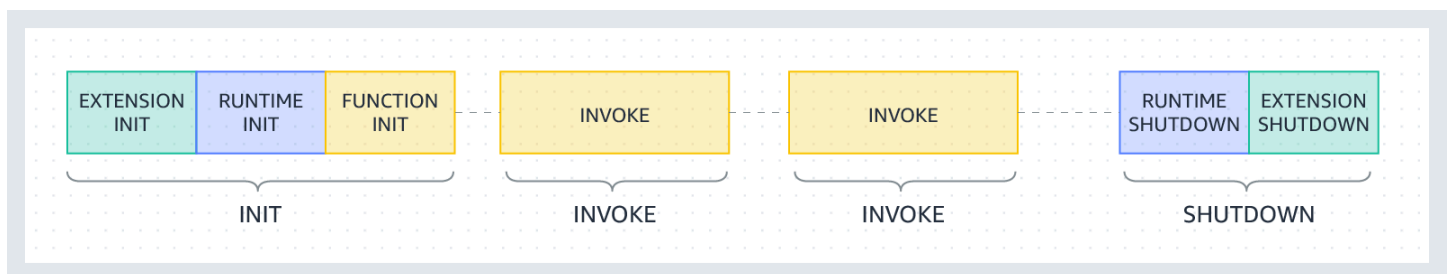
Quando si crea la funzione Lambda, si specificano le informazioni di configurazione, ad esempio la quantità di memoria disponibile e il tempo massimo di esecuzione consentito per la funzione. Lambda utilizza queste informazioni per impostare l'ambiente di esecuzione.

Il runtime della funzione e ogni estensione esterna sono processi eseguiti all'interno dell'ambiente di esecuzione. Le autorizzazioni, le risorse, le credenziali e le variabili di ambiente sono condivise tra la funzione e le estensioni.

## Argomenti

- [Ciclo di vita dell'ambiente di esecuzione Lambda](#)
- [Avvii a freddo e latenza](#)
- [Riduzione degli avvii a freddo con la simultaneità fornita](#)
- [Ottimizzazione dell'inizializzazione statica](#)

## Ciclo di vita dell'ambiente di esecuzione Lambda



Ogni fase inizia con un evento che Lambda invia al runtime e a tutte le estensioni registrate. Il runtime e ogni estensione indicano il completamento inviando una richiesta API Next. Lambda congela l'ambiente di esecuzione quando il runtime e ogni estensione sono stati completati e non ci sono eventi in sospeso.

Le fasi del ciclo di vita di Durable Functions includono:

- Init: inizializzazione standard più configurazione dello stato duraturo
- Invoke: può includere esecuzioni di più passaggi con checkpoint automatico
- Attendi: la funzione può sospendere l'esecuzione senza consumare risorse
- Riprendi: la funzione si riavvia dall'ultimo checkpoint
- Arresto: pulizia dello stato e delle risorse durevoli

## Argomenti

- [Fase di init](#)
- [Errori durante la fase di inizializzazione](#)
- [Fase di ripristino \(solo Lambda\) SnapStart](#)
- [Invoca fase](#)
- [Errori durante la fase di richiamo](#)
- [Fase di arresto](#)

## Fase di init

Nella fase Init, Lambda esegue tre incarichi:

- Avvia tutte le estensioni (Extension init)
- Esegue il bootstrap del runtime (Runtime init)
- Esegue il codice statico della funzione (Function init)
- Esegui qualsiasi hook di runtime prima del checkpoint (solo [Lambda](#) SnapStart)

La fase Init termina quando il runtime e tutte le estensioni segnalano che sono pronti inviando una richiesta Next API. La fase Init è limitata a 10 secondi. Se tutte e tre le attività non vengono completate entro 10 secondi, Lambda ritenta la fase Init al momento della prima chiamata di funzione con timeout della funzione configurata.

Quando [SnapStart](#) è attivato, la fase Init si verifica quando si pubblica una versione della funzione. Lambda salva uno snapshot della memoria e dello stato del disco dell'ambiente di esecuzione inizializzato, mantiene lo snapshot crittografato e lo memorizza nella cache per l'accesso a bassa

latenza. Se disponi di un [hook di runtime](#) prima del checkpoint, il codice viene eseguito alla fine della fase Init.

#### Note

Il timeout di 10 secondi non si applica alle funzioni che utilizzano la concorrenza predisposta o le istanze gestite Lambda. SnapStart Per le funzioni di concorrenza con provisioning e Managed Instances SnapStart, il codice di inizializzazione può essere eseguito per un massimo di 15 minuti. Il limite di tempo è 130 secondi o il timeout della funzione configurato (massimo 900 secondi), a seconda di quale dei due valori sia più elevato.

Quando utilizzi la [simultaneità fornita](#), Lambda inizializza l'ambiente di esecuzione quando configuri le impostazioni del PC per una funzione. Lambda, inoltre, garantisce che gli ambienti di esecuzione inizializzati siano sempre disponibili prima delle invocazioni. Potresti notare delle lacune tra le fasi di richiamo e inizializzazione della funzione. A seconda del runtime e della configurazione della memoria della funzione, è inoltre possibile che si verifichi una latenza variabile alla prima chiamata in un ambiente di esecuzione inizializzato.

Per le funzioni che utilizzano la concorrenza su richiesta, Lambda potrebbe occasionalmente inizializzare gli ambienti di esecuzione prima delle richieste di richiamo. Quando ciò accade, potresti anche osservare un intervallo di tempo tra le fasi di inizializzazione e invocazione della funzione. È preferibile non acquisire dipendenza da questo comportamento.

#### Errori durante la fase di inizializzazione

Se una funzione si arresta in modo anomalo o si verifica un timeout durante la fase Init, Lambda emette informazioni sull'errore nel log INIT\_REPORT.

Example— Log INIT\_REPORT per il timeout

```
INIT_REPORT Init Duration: 1236.04 ms Phase: init Status: timeout
```

Example— Log INIT\_REPORT per gli errori di estensione

```
INIT_REPORT Init Duration: 1236.04 ms Phase: init Status: error Error Type:  
Extension.Crash
```

Se la Init fase ha esito positivo, Lambda non emette il INIT\_REPORT log a meno che non sia abilitata la concorrenza predisposta [SnapStart](#). ??? SnapStart e le funzioni di concorrenza

predisposte emettono sempre. `INIT_REPORT` Per ulteriori informazioni, consulta [Monitoraggio per Lambda SnapStart](#).

### Fase di ripristino (solo Lambda) SnapStart

Quando si richiama per la prima volta una [SnapStart](#) funzione e man mano che la funzione aumenta, Lambda riprende nuovi ambienti di esecuzione dallo snapshot persistente invece di inizializzare la funzione da zero. Se disponi di un [hook di runtime](#) dopo il ripristino, il codice viene eseguito alla fine della fase Restore. Ti sarà addebitata la durata degli hook di runtime dopo il ripristino. Il runtime deve essere caricato e gli hook di runtime dopo il ripristino devono essere completati entro il limite di timeout (10 secondi). Altrimenti, otterrai un. `SnapStartTimeoutException` Al termine della fase Restore, Lambda chiama il gestore della funzione ([Invoca fase](#)).

### Errori durante la fase di ripristino

Se la fase Restore fallisce, Lambda emette informazioni sull'errore nel log `RESTORE_REPORT`.

Example— Log `RESTORE_REPORT` per il timeout

```
RESTORE_REPORT Restore Duration: 1236.04 ms Status: timeout
```

Example— Log `RESTORE_REPORT` per gli errori dell'hook di runtime

```
RESTORE_REPORT Restore Duration: 1236.04 ms Status: error Error Type: Runtime.ExitError
```

Per ulteriori informazioni sul log `RESTORE_REPORT`, consulta la pagina [Monitoraggio per Lambda SnapStart](#).

### Invoca fase

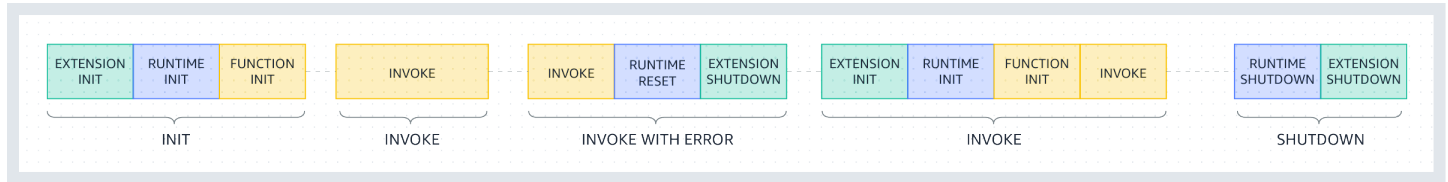
Quando una funzione Lambda viene richiamata in risposta a una richiesta API Next, Lambda invia un evento `Invoke` al runtime e a ogni estensione.

L'impostazione di timeout della funzione limita la durata dell'intera fase `Invoke`. Ad esempio, se si imposta il timeout della funzione come 360 secondi, la funzione e tutte le estensioni devono essere completate entro 360 secondi. Si noti che non esiste una fase post-richiamo indipendente. La durata è la somma di tutto il tempo di chiamata (runtime + estensioni) e non viene calcolata fino a quando la funzione e tutte le estensioni non hanno terminato l'esecuzione.

La fase di richiamo termina dopo il runtime e tutte le estensioni segnalano che vengono eseguite inviando una richiesta Next API.

## Errori durante la fase di richiamo

Se la funzione Lambda si arresta in modo anomalo o si verifica un timeout durante la fase Invoke, Lambda reimposta l'ambiente di esecuzione. Il diagramma seguente mostra il comportamento dell'ambiente di esecuzione Lambda in caso di errore di invocazione:



Nel diagramma precedente:

- La prima è la fase INIT e viene eseguita senza errori.
- La seconda è la fase INVOKE e viene eseguita senza errori.
- Ad un certo punto, supponiamo che la funzione abbia un errore di richiamo (le cause più comuni includono timeout delle funzioni, errori di runtime, esaurimento della memoria, problemi di connettività VPC, errori di autorizzazione, limiti di concorrenza e vari problemi di configurazione). Per un elenco completo dei possibili errori di richiamo, consulta [the section called “Invocazione”](#). La terza fase, denominata INVOKE WITH ERROR, mostra questo scenario.

In questo caso, il servizio Lambda esegue un ripristino. Il reset si comporta come un evento Shutdown. Innanzitutto Lambda chiude il runtime, poi invia un evento Shutdown a ogni estensione esterna registrata. L'evento include il motivo dell'arresto. Se questo ambiente viene utilizzato per una nuova chiamata, Lambda re-inizializza l'estensione e il runtime insieme alla chiamata successiva.

Nota che il ripristino di Lambda non cancella il contenuto della `/tmp` directory prima della successiva fase di inizializzazione. Questo comportamento è coerente con la normale fase di arresto.

### Note

AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS.

Se la configurazione del registro di sistema della funzione è impostata su testo normale, questa modifica influisce sui messaggi di registro acquisiti in CloudWatch Logs quando la

funzione non riesce a richiamare la funzione. Gli esempi seguenti mostrano gli output dei log nei formati vecchi e nuovi.

Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regioni, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia di nuovo formato.

#### Example CloudWatch Output del log dei log (crash di runtime o estensione) - vecchio stile

```
START RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1 Version: $LATEST
RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1 Error: Runtime exited without providing a reason
Runtime.ExitError
END RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1
REPORT RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1 Duration: 933.59 ms Billed Duration: 934 ms Memory Size: 128 MB Max Memory Used: 9 MB
```

#### Example CloudWatch Registra l'output del registro (timeout della funzione) - vecchio stile

```
START RequestId: b70435cc-261c-4438-b9b6-efe4c8f04b21 Version: $LATEST
2024-03-04T17:22:38.033Z b70435cc-261c-4438-b9b6-efe4c8f04b21 Task timed out after 3.00 seconds
END RequestId: b70435cc-261c-4438-b9b6-efe4c8f04b21
REPORT RequestId: b70435cc-261c-4438-b9b6-efe4c8f04b21 Duration: 3004.92 ms Billed Duration: 3117 ms Memory Size: 128 MB Max Memory Used: 33 MB Init Duration: 111.23 ms
```

Il nuovo formato per CloudWatch i log include un status campo aggiuntivo nella REPORT riga. In caso di arresto anomalo del runtime o dell'estensione, la riga REPORT include anche un campo **ErrorType**.

#### Example CloudWatch Output del registro dei log (errore di runtime o estensione): nuovo stile

```
START RequestId: 5b866fb1-7154-4af6-8078-6ef6ca4c2ddd Version: $LATEST
END RequestId: 5b866fb1-7154-4af6-8078-6ef6ca4c2ddd
REPORT RequestId: 5b866fb1-7154-4af6-8078-6ef6ca4c2ddd Duration: 133.61 ms Billed Duration: 214 ms Memory Size: 128 MB Max Memory Used: 31 MB Init Duration: 80.00 ms Status: error Error Type: Runtime.ExitError
```

## Example CloudWatch Registra l'output del registro (timeout della funzione) - nuovo stile

```
START RequestId: 527cb862-4f5e-49a9-9ae4-a7edc90f0fda Version: $LATEST
END RequestId: 527cb862-4f5e-49a9-9ae4-a7edc90f0fda
REPORT RequestId: 527cb862-4f5e-49a9-9ae4-a7edc90f0fda Duration: 3016.78 ms Billed
Duration: 3101 ms Memory Size: 128 MB Max Memory Used: 31 MB Init Duration: 84.00
ms Status: timeout
```

- La quarta fase rappresenta la fase INVOKE, immediatamente successiva a un errore di chiamata. In questo caso Lambda inizializza nuovamente l'ambiente eseguendo nuovamente la fase INIT. Questa operazione è chiamata inizializzazione soppressa.

Quando si verificano inizializzazioni sopresse, Lambda non segnala esplicitamente una fase INIT aggiuntiva nei log. CloudWatch Potresti invece notare che la durata nella riga REPORT include una durata INIT aggiuntiva + la durata INVOKE. Ad esempio, supponiamo di vedere i seguenti accessi: CloudWatch

```
2022-12-20T01:00:00.000-08:00 START RequestId: XXX Version: $LATEST
2022-12-20T01:00:02.500-08:00 END RequestId: XXX
2022-12-20T01:00:02.500-08:00 REPORT RequestId: XXX Duration: 3022.91 ms
Billed Duration: 3000 ms Memory Size: 512 MB Max Memory Used: 157 MB
```

In questo esempio, la differenza tra i timestamp REPORT e START è di 2,5 secondi. Ciò non corrisponde alla durata riportata di 3022,91 millisecondi, perché la durata della INIT (inizializzazione soppressa) aggiuntiva eseguita da Lambda non viene tenuta in conto. Da questo esempio puoi capire che la fase INVOKE effettiva ha richiesto 2,5 secondi.

Per ulteriori informazioni su questo comportamento, puoi utilizzare il [Accesso ai dati di telemetria in tempo reale per le estensioni tramite l'API Telemetry](#). L'API di telemetria produce gli eventi INIT\_START, INIT\_RUNTIME\_DONE e INIT\_REPORT insieme a phase=invoke ogni volta che le inizializzazioni vengono sopresse tra le fasi di invocazione.

- La quinta fase rappresenta la fase SHUTDOWN, che viene eseguita senza errori.

### Fase di arresto

Quando Lambda sta per chiudere il runtime, invia un evento Shutdown a ciascuna estensione esterna registrata. Le estensioni possono utilizzare questo tempo per le attività di pulizia finali. L'evento Shutdown è una risposta a una richiesta API Next.

Limite di durata: la durata massima della fase Shut down dipende dalla configurazione delle estensioni registrate:

- 0 ms: funzione senza estensioni registrate
- 500 ms: funzione con estensione interna registrata
- 2.000 ms: funzione con una o più estensioni esterne registrate

Se il runtime o un'estensione non risponde all'evento Shut down entro il limite, Lambda termina il processo utilizzando un segnale SIGKILL.

Dopo che la funzione e tutte le estensioni sono state completate, Lambda mantiene l'ambiente di esecuzione per qualche tempo in previsione di un'altra chiamata di funzione. Tuttavia, Lambda termina gli ambienti di esecuzione ogni poche ore per consentire gli aggiornamenti e la manutenzione del runtime, anche per le funzioni che vengono richiamate continuamente. Non si deve dare per scontato che l'ambiente di esecuzione persista all'infinito. Per ulteriori informazioni, consulta [Implementazione stateless nelle funzioni](#).

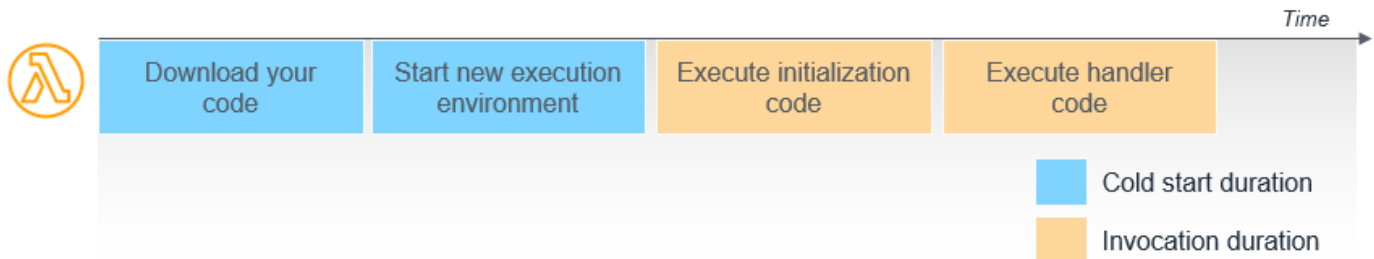
Quando la funzione viene richiamata nuovamente, Lambda disattiva l'ambiente per il riutilizzo. Il riutilizzo dell'ambiente di esecuzione ha le seguenti implicazioni:

- Gli oggetti dichiarati al di fuori del metodo del gestore della funzione rimangono inizializzati, fornendo ulteriore ottimizzazione quando la funzione viene nuovamente invocata. Ad esempio, se la funzione Lambda stabilisce la connessione a un database, nelle invocazioni successive viene utilizzata la connessione originaria anziché stabilirne un'altra. È consigliabile aggiungere al codice una logica per verificare l'esistenza di una connessione prima che ne venga stabilita una nuova.
- Ogni ambiente di esecuzione fornisce da 512 MB a 10.240 MB con incrementi di 1 MB di spazio su disco nella directory /tmp. Il contenuto della directory rimane quando il contesto di esecuzione è bloccato, fornendo così una cache transitoria utilizzabile per più invocazioni. È possibile aggiungere ulteriore codice per verificare se la cache contiene i dati memorizzati. Per ulteriori informazioni sui limiti delle dimensioni della distribuzione, vedere [Quote di Lambda](#).
- Processi in background o callback che sono stati avviati dalla funzione Lambda e che non sono stati completati quando la funzione è terminata; riprendere se Lambda riutilizza l'ambiente di esecuzione. È necessario accertarsi che tutti i processi in background o le callback nel codice vengano completate prima che il codice sia terminato.



## Avvii a freddo e latenza

Quando Lambda riceve una richiesta per eseguire una funzione tramite l'API Lambda, il servizio prepara innanzitutto un ambiente di esecuzione. Durante questa fase di inizializzazione, il servizio scarica il codice, avvia l'ambiente ed esegue qualsiasi codice di inizializzazione all'esterno del gestore principale. Infine, Lambda esegue il codice del gestore.



In questo diagramma, i primi due passaggi di download del codice e configurazione dell'ambiente vengono spesso definiti «avvio a freddo». Questo tempo [ti](#) viene addebitato e questo aggiunge latenza alla durata complessiva della chiamata.

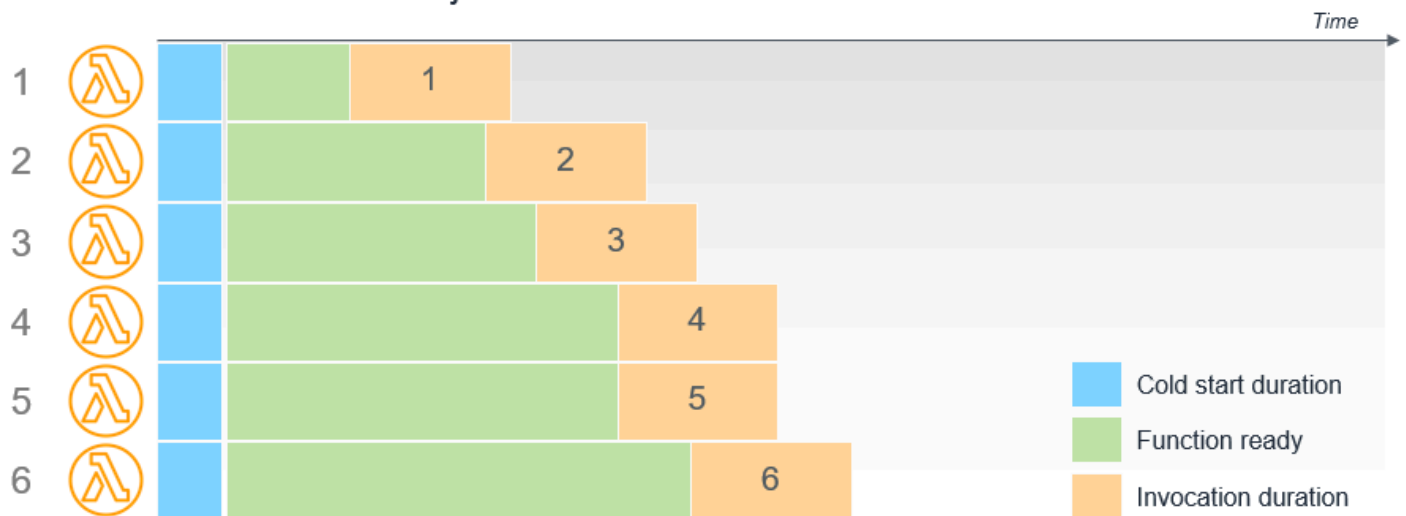
Al termine dell'invocazione, l'ambiente di esecuzione viene bloccato. Per migliorare la gestione e le prestazioni delle risorse, Lambda mantiene l'ambiente di esecuzione per un periodo di tempo. Durante questo periodo, se arriva un'altra richiesta per la stessa funzione, Lambda può riutilizzare l'ambiente. Questa seconda richiesta in genere termina più rapidamente, poiché l'ambiente di esecuzione è già completamente configurato. Questo è chiamato «avvio a caldo».

Le partenze a freddo si verificano in genere in meno dell'1% delle invocazioni. La durata di un avvio a freddo varia da meno di 100 ms a oltre 1 secondo. In generale, gli avviamenti a freddo sono in genere più comuni nelle funzioni di sviluppo e test rispetto ai carichi di lavoro di produzione. Questo perché le funzioni di sviluppo e test vengono in genere richiamate meno frequentemente.

### Riduzione degli avvii a freddo con la simultaneità fornita

Se hai bisogno di orari di inizio delle funzioni prevedibili per il tuo carico di lavoro, la [simultaneità fornita](#) è la soluzione consigliata per garantire la latenza più bassa possibile. Questa funzionalità preinizializza gli ambienti di esecuzione, riducendo gli avviamenti a freddo.

Ad esempio, una funzione con una concorrenza prevista pari a 6 ha 6 ambienti di esecuzione preriscaldati.



### Ottimizzazione dell'inizializzazione statica

L'inizializzazione statica avviene prima che il codice dell'handler inizi a essere eseguito in una funzione. Questo è il codice di inizializzazione fornito, che si trova all'esterno del gestore principale. Questo codice viene spesso utilizzato per importare librerie e dipendenze, impostare configurazioni e inizializzare le connessioni ad altri servizi.

Il seguente esempio in Python mostra l'importazione e la configurazione dei moduli e la creazione del client Amazon S3 durante la fase di inizializzazione, prima che la funzione `lambda_handler` venga eseguita durante la fase di invocazione.

```
import os
import json
import cv2
import logging
import boto3

s3 = boto3.client('s3')
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Handler logic...
```

Il maggior contributo alla latenza prima dell'esecuzione della funzione proviene dal codice di inizializzazione. Questo codice viene eseguito quando viene creato un nuovo ambiente di esecuzione

per la prima volta. Il codice di inizializzazione non viene eseguito nuovamente se una invocazione utilizza un ambiente di esecuzione caldo. I fattori che influiscono sulla latenza del codice di inizializzazione includono:

- La dimensione del pacchetto di funzioni, in termini di librerie e dipendenze importate e livelli Lambda.
- La quantità di codice e la procedura di inizializzazione.
- Le prestazioni delle librerie e di altri servizi nella configurazione di connessioni e altre risorse.

Esistono diversi passaggi che gli sviluppatori possono adottare per ottimizzare la latenza di inizializzazione statica. Se una funzione ha molti oggetti e connessioni, potresti essere in grado di riprogettare una singola funzione in più funzioni specializzate. Queste sono singolarmente più piccole e ognuna ha meno codice di inizializzazione.

È importante che le funzioni importino solo le librerie e le dipendenze di cui hanno bisogno. Ad esempio, se utilizzi Amazon DynamoDB solo nell' AWS SDK, puoi richiedere un singolo servizio anziché l'intero SDK. Confronta i tre esempi seguenti:

```
// Instead of const AWS = require('aws-sdk'), use:  
const DynamoDB = require('aws-sdk/clients/dynamodb')  
  
// Instead of const AWSXRay = require('aws-xray-sdk'), use:  
const AWSXRay = require('aws-xray-sdk-core')  
  
// Instead of const AWS = AWSXRay.captureAWS(require('aws-sdk')), use:  
const dynamodb = new DynamoDB.DocumentClient()  
AWSXRay.captureAWSClient(dynamodb.service)
```

L'inizializzazione statica è spesso anche il posto migliore per aprire connessioni al database per consentire a una funzione di riutilizzare le connessioni su più invocazioni allo stesso ambiente di esecuzione. Tuttavia, potreste avere un numero elevato di oggetti che vengono utilizzati solo in determinati percorsi di esecuzione della funzione. In questo caso, puoi caricare lentamente le variabili nell'ambito globale per ridurre la durata dell'inizializzazione statica.

Evitate le variabili globali per informazioni specifiche del contesto. Se la funzione ha una variabile globale che viene utilizzata solo per la durata di una singola invocazione e viene reimpostata per l'invocazione successiva, utilizza un ambito di variabile locale per l'handler. In questo modo non solo si evitano perdite di variabili globali tra le invocazioni, ma si migliorano anche le prestazioni dell'inizializzazione statica.

# Creazione di architetture basate sugli eventi con Lambda

Un evento è tutto ciò che attiva l'esecuzione di una funzione Lambda. Gli eventi possono attivare una funzione Lambda in due modi: tramite invocazione diretta (push) e mappatura dell'origine degli eventi (pull).

Molti AWS servizi possono richiamare direttamente le funzioni Lambda. Questi servizi inviano eventi alla tua funzione Lambda. Gli eventi che attivano le funzioni possono essere praticamente qualsiasi cosa, da una richiesta HTTP tramite API Gateway, una pianificazione gestita da EventBridge Scheduler, un AWS IoT evento o un evento Amazon S3. Con lo strumento di mappatura dell'origine degli eventi, Lambda recupera (pull) attivamente gli eventi da una coda o da un flusso. Puoi configurare Lambda per verificare la presenza di eventi da un servizio supportato e Lambda gestisce il polling e l'invocazione della tua funzione.

Quando passati alla funzione, gli eventi sono strutturati nel formato JSON. La struttura JSON varia a seconda del servizio che la genera e del tipo di evento. Mentre le chiamate alle funzioni Lambda standard possono durare fino a 15 minuti, Lambda è più adatta per chiamate brevi che durano un secondo o meno. Ciò è particolarmente vero per le architetture basate sugli eventi, in cui ogni funzione Lambda viene trattata come un microservizio responsabile dell'esecuzione di una serie ristretta di istruzioni specifiche.

## Note

Event-driven le architetture comunicano tra diversi sistemi utilizzando reti, che introducono una latenza variabile. Per carichi di lavoro che richiedono una latenza molto bassa, come i sistemi di trading in tempo reale, questo design potrebbe non essere la scelta migliore. Tuttavia, per carichi di lavoro altamente scalabili e disponibili o con schemi di traffico imprevedibili, le architetture basate sugli eventi possono fornire un modo efficace per soddisfare queste esigenze.

## Argomenti

- [Vantaggi delle architetture basate sugli eventi](#)
- [Trade-offs di architetture basate sugli eventi](#)
- [Anti-patterns nelle applicazioni basate sugli eventi Lambda-based](#)

## Vantaggi delle architetture basate sugli eventi

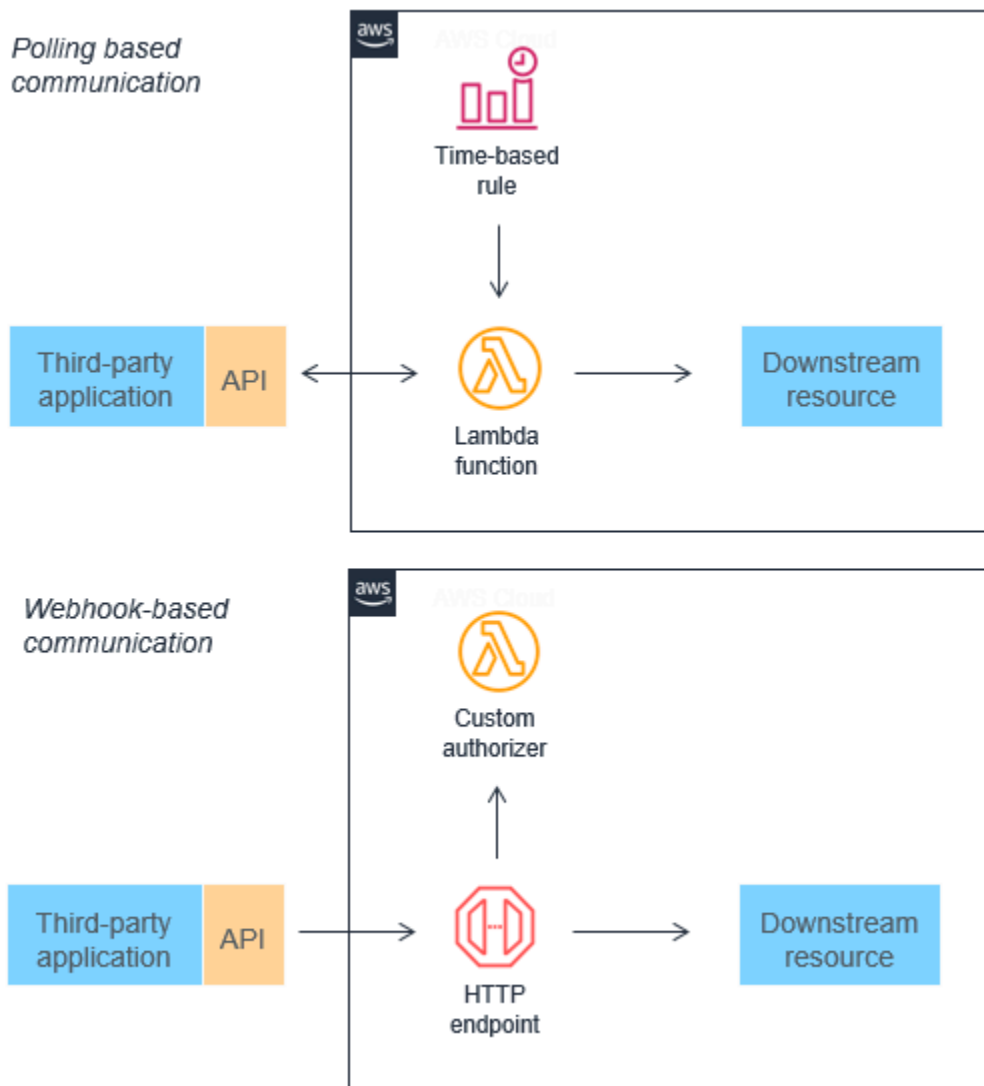
Lambda supporta due metodi di invocazione nelle architetture basate sugli eventi:

1. Richiamo diretto (metodo push): i AWS servizi attivano direttamente le funzioni Lambda. Ad esempio:
  - Amazon S3 attiva una funzione quando viene caricato un file
  - API Gateway attiva una funzione quando riceve una richiesta HTTP
2. Strumento di mappatura dell'origine degli eventi (metodo pull): Lambda recupera gli eventi e richiama le funzioni. Ad esempio:
  - Lambda recupera i messaggi da una coda Amazon SQS e invoca una funzione
  - Lambda legge i record da un flusso DynamoDB e invoca una funzione

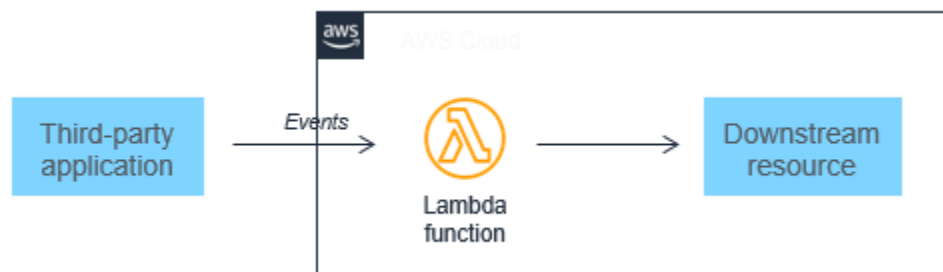
Entrambi i metodi contribuiscono ai vantaggi delle architetture basate sugli eventi, come descritto di seguito.

### Sostituzione di polling e webhook con eventi

Molte architetture tradizionali utilizzano meccanismi di polling e webhook per comunicare lo stato tra diversi componenti. Il polling può essere molto inefficiente per il recupero degli aggiornamenti poiché esiste un ritardo tra la disponibilità di nuovi dati e la sincronizzazione con i servizi a valle. I webhook non sono sempre supportati da altri microservizi con cui si desidera integrarsi. Potrebbero inoltre richiedere configurazioni di autorizzazione e autenticazione personalizzate. In entrambi i casi, questi metodi di integrazione sono difficili da scalare su richiesta senza ulteriore lavoro da parte dei team di sviluppo.



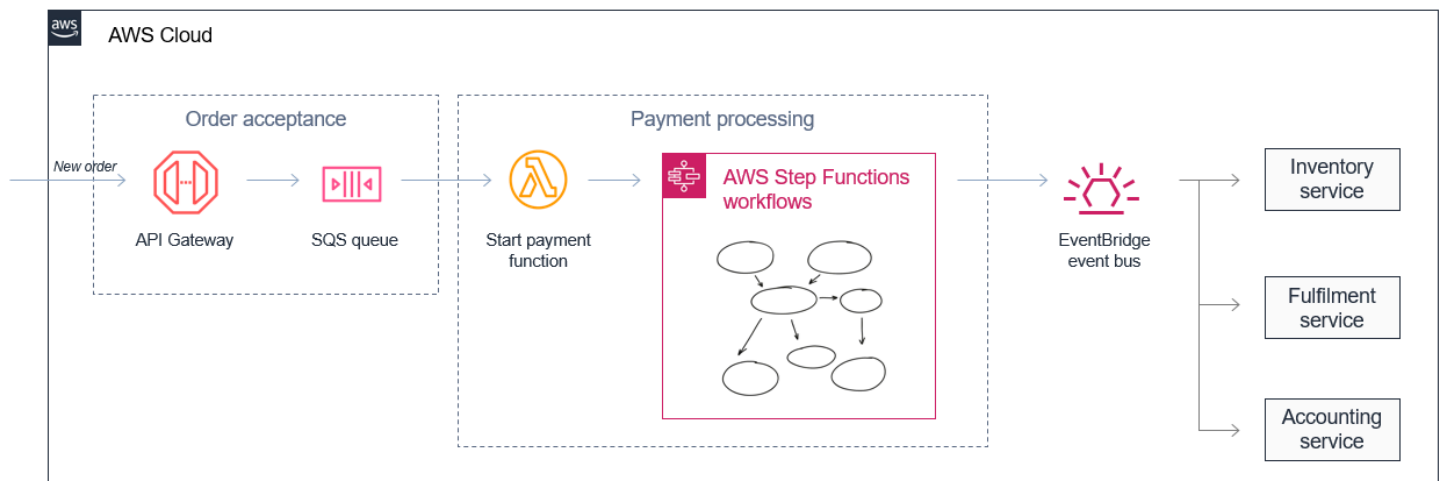
Entrambi questi meccanismi possono essere sostituiti da eventi che possono essere filtrati, indirizzati e trasferiti a valle verso l'utilizzo di microservizi. Questo approccio può comportare un minore consumo di larghezza di banda, un minore utilizzo della CPU e, potenzialmente, una riduzione dei costi. Queste architetture possono anche ridurre la complessità, poiché ogni unità funzionale è più piccola e spesso contiene meno codice.



Event-driven le architetture possono anche semplificare la progettazione di sistemi quasi in tempo reale, aiutando le organizzazioni ad abbandonare l'elaborazione basata su batch. Gli eventi vengono generati nel momento in cui lo stato dell'applicazione cambia, quindi il codice personalizzato di un microservizio deve essere progettato per gestire l'elaborazione di un singolo evento. Poiché la scalabilità è gestita dal servizio Lambda, questa architettura può gestire aumenti significativi del traffico senza modificare il codice personalizzato. Man mano che gli eventi aumentano, aumenta anche il livello di elaborazione che elabora gli eventi.

### Riduzione della complessità

I microservizi consentono a sviluppatori e architetti di semplificare flussi di lavoro complessi. Ad esempio, un monolite di e-commerce può essere suddiviso in processi di accettazione degli ordini e pagamento con servizi separati di inventario, evasione degli ordini e contabilità. Ciò che potrebbe essere complesso da gestire e orchestrare in un monolite diventa una serie di servizi disaccoppiati che comunicano in modo asincrono con gli eventi.



Questo approccio consente inoltre di assemblare servizi che elaborano i dati a velocità diverse. In questo caso, un microservizio di accettazione degli ordini può archiviare elevati volumi di ordini in arrivo bufferizzando i messaggi in una coda Amazon SQS.

Un servizio di elaborazione dei pagamenti, che in genere è più lento a causa della complessità della gestione dei pagamenti, può ricevere un flusso costante di messaggi dalla coda di Amazon SQS. È in grado di orchestrare complesse logiche di gestione dei tentativi e degli errori utilizzando AWS Step Functions e coordinare i flussi di lavoro di pagamento attivi per centinaia di migliaia di ordini.

Approccio alternativo: per l'orchestrazione utilizzando linguaggi di programmazione standard, puoi utilizzare le funzioni serverless Lambda. ??? Le funzioni serverless consentono di scrivere in codice la logica di accettazione dell'ordine, elaborazione dei pagamenti e notifica con checkpoint

e nuovo tentativo automatici. Questo approccio funziona bene quando il flusso di lavoro coinvolge principalmente funzioni Lambda e si preferisce mantenere la logica di orchestrazione nel codice.

## Migliorare la scalabilità e l'estensibilità

I microservizi generano eventi che in genere vengono pubblicati su servizi di messaggistica come Amazon SNS e Amazon SQS. Questi si comportano come un buffer elastico tra i microservizi e aiutano a gestire la scalabilità quando il traffico aumenta. Servizi come Amazon EventBridge possono quindi filtrare e indirizzare i messaggi in base al contenuto dell'evento, come definito nelle regole. Di conseguenza, le applicazioni basate sugli eventi possono essere più scalabili e offrire una maggiore ridondanza rispetto alle applicazioni monolitiche.

Questo sistema è inoltre altamente estensibile e consente ad altri team di estendere le funzionalità e aggiungere funzionalità senza influire sui microservizi di elaborazione degli ordini e di elaborazione dei pagamenti. Pubblicando eventi utilizzando EventBridge, questa applicazione si integra con i sistemi esistenti, come il microservizio di inventario, ma consente anche l'integrazione di qualsiasi applicazione futura come consumatore di eventi. I produttori di eventi non conoscono gli utenti di eventi, il che può contribuire a semplificare la logica dei microservizi.

## Trade-offs di architetture basate sugli eventi

### Latenza variabile

A differenza delle applicazioni monolitiche, che potrebbero elaborare tutto all'interno dello stesso spazio di memoria su un singolo dispositivo, le applicazioni basate sugli eventi comunicano tra le reti. Questo design introduce una latenza variabile. Sebbene sia possibile progettare applicazioni per ridurre al minimo la latenza, le applicazioni monolitiche possono quasi sempre essere ottimizzate per una latenza inferiore a scapito della scalabilità e della disponibilità.

I carichi di lavoro che richiedono prestazioni costanti a bassa latenza, come le applicazioni di trading ad alta frequenza nelle banche o l'automazione robotica inferiore al millisecondo nei magazzini, non sono buoni candidati per l'architettura basata sugli eventi.

### Consistenza finale

Un evento rappresenta un cambiamento di stato e, poiché molti eventi fluiscono attraverso diversi servizi di un'architettura in un dato momento, tali carichi di lavoro alla fine sono [spesso coerenti](#). Ciò rende più complessa l'elaborazione delle transazioni, la gestione dei duplicati o la determinazione dell'esatto stato generale di un sistema.



Molti carichi di lavoro contengono una combinazione di requisiti che alla fine sono coerenti (ad esempio, gli ordini totali nell'ora corrente) o fortemente coerenti (ad esempio, l'inventario corrente). Per i carichi di lavoro che richiedono una forte coerenza dei dati, esistono modelli di architettura a supporto di questa funzionalità. Ad esempio:

- DynamoDB è in grado di fornire una [elevata consistenza di lettura](#), a volte con una latenza più elevata, consumando un throughput maggiore rispetto alla modalità predefinita. DynamoDB può anche [supportare le transazioni](#) per aiutare a mantenere la coerenza dei dati.
- Puoi utilizzare Amazon RDS per funzionalità che richiedono [proprietà ACID](#), sebbene qualsiasi database relazionale sia generalmente meno scalabile di database NoSQL come DynamoDB. [Server proxy per Amazon RDS](#) può aiutare a gestire il pooling e la scalabilità delle connessioni per utenti effimeri come le funzioni Lambda.

Event-based le architetture sono generalmente progettate in base a singoli eventi anziché a grandi quantità di dati. In genere, i flussi di lavoro sono progettati per gestire le fasi di un singolo evento o flusso di esecuzione anziché operare su più eventi contemporaneamente. In modalità serverless, l'elaborazione degli eventi in tempo reale è preferita all'elaborazione in batch: i batch devono essere sostituiti con molti aggiornamenti incrementali più piccoli. Se da un lato ciò può rendere i carichi di lavoro più disponibili e scalabili, dall'altro rende più difficile per gli eventi conoscere altri eventi.

### Restituzione di valori ai chiamanti

In molti casi, le applicazioni basate su eventi sono asincrone. Ciò significa che i servizi di chiamata non attendono le richieste da altri servizi prima di continuare con altre attività. Questa è una caratteristica fondamentale delle architetture basate sugli eventi che consente scalabilità e flessibilità. Significa anche che il passaggio dei valori restituiti o del risultato di un flusso di lavoro è più complesso rispetto ai flussi di esecuzione sincroni.

La maggior parte delle invocazioni Lambda nei sistemi di produzione sono [asincrone](#) e rispondono agli eventi di servizi come Amazon S3 o Amazon SQS. In questi casi, il successo o il fallimento dell'elaborazione di un evento è spesso più importante della restituzione di un valore. In Lambda sono fornite funzionalità come le [code DLQ](#) per garantire che sia possibile identificare e riprovare gli eventi non riusciti senza dover avvisare il chiamante.

### Esecuzione del debug tra servizi e funzioni

Il debug di sistemi basati sugli eventi è inoltre diverso rispetto a un'applicazione monolitica. Poiché diversi sistemi e servizi trasmettono eventi, non è possibile registrare e riprodurre lo stato esatto

di più servizi quando si verifica un errore. Poiché ogni invocazione di servizio e funzione ha file di log separati, può essere più complicato determinare cosa è successo a un evento specifico che ha causato un errore.

Esistono tre requisiti importanti per creare un approccio di debug efficace nei sistemi basati sugli eventi. Innanzitutto, è fondamentale disporre di un sistema di registrazione robusto, fornito da Amazon su tutti AWS i servizi e integrato nelle funzioni Lambda. CloudWatch In secondo luogo, in questi sistemi, è importante assicurarsi che ogni evento abbia un identificatore di transazione che viene registrato in ogni fase della transazione, per facilitare la ricerca dei log.

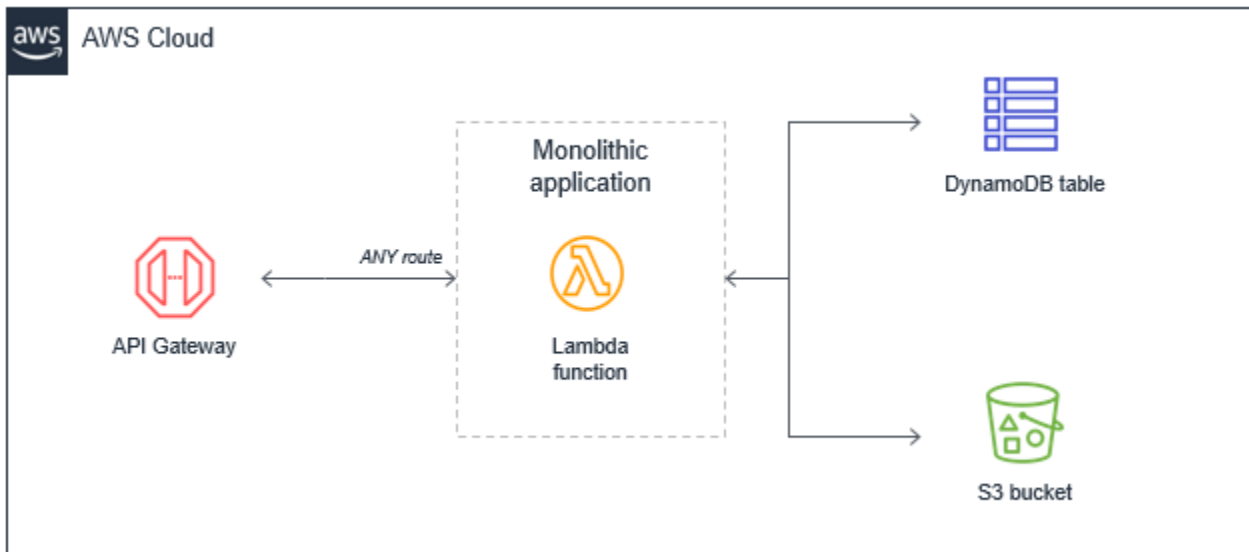
Infine, si consiglia vivamente di automatizzare l'analisi e l'analisi dei log utilizzando un servizio di debug e monitoraggio come AWS X-Ray. Ciò può consumare i log di più invocazioni Lambda e servizi, rendendo molto più semplice individuare la causa principale dei problemi. Consulta la procedura dettagliata [per la risoluzione dei problemi per una descrizione approfondita dell'utilizzo](#) per la risoluzione dei problemi. X-Ray

## Anti-patterns nelle applicazioni basate sugli eventi Lambda-based

Quando crei architetture basate sugli eventi con Lambda, evita i seguenti anti-pattern comuni. Questi modelli funzionano ma possono aumentare i costi e la complessità.

### Il monolite Lambda

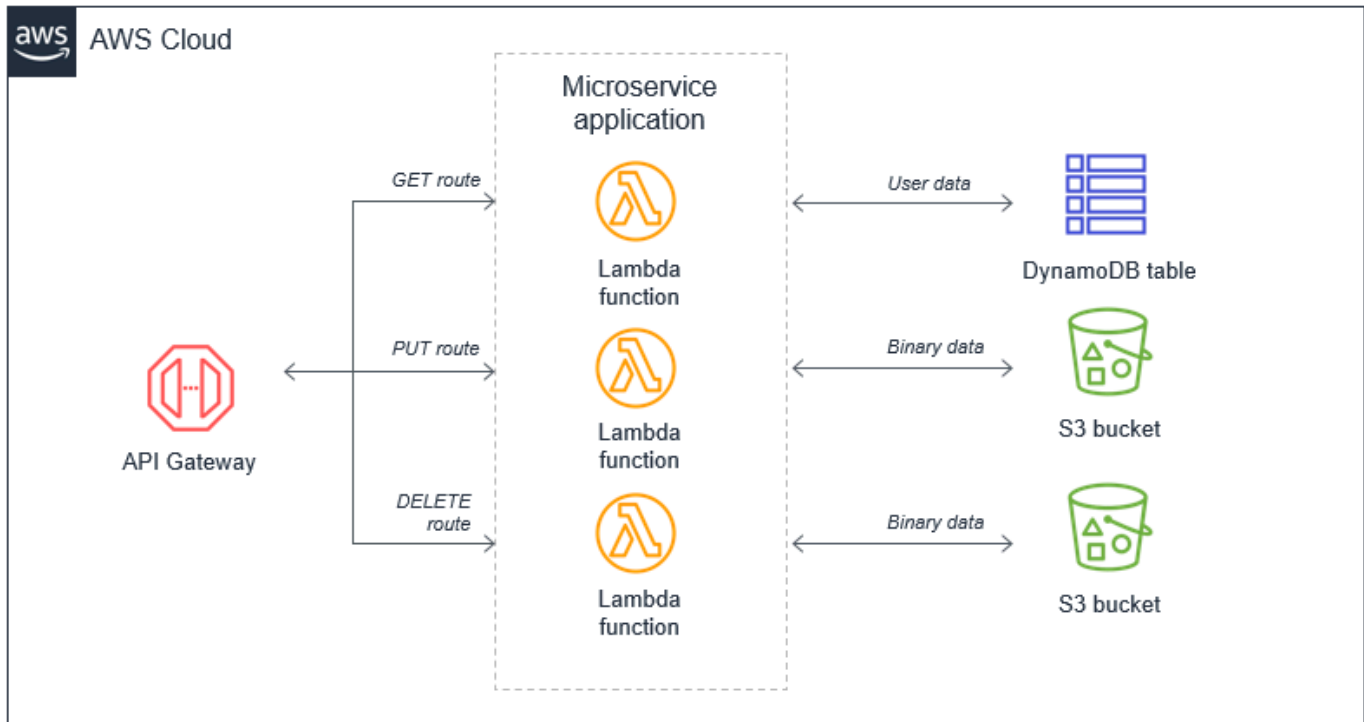
In molte applicazioni migrate da server tradizionali, come le istanze Amazon EC2 o le applicazioni Elastic Beanstalk, gli sviluppatori «estraggono e spostano» il codice esistente. Spesso, ciò si traduce in una singola funzione Lambda che contiene tutta la logica dell'applicazione attivata per tutti gli eventi. Per un'applicazione Web di base, una funzione Lambda monolitica gestirebbe tutti i percorsi API Gateway e si integrerebbe con tutte le risorse downstream necessarie.



Questo approccio presenta diversi inconvenienti:

- Dimensione del pacchetto: la funzione Lambda potrebbe essere molto più grande perché contiene tutto il codice possibile per tutti i percorsi, il che rende più lenta l'esecuzione del servizio Lambda.
- Difficile applicare il privilegio minimo: il ruolo di [esecuzione della funzione](#) deve consentire le autorizzazioni a tutte le risorse necessarie per tutti i percorsi, il che rende le autorizzazioni molto ampie. Si tratta di un problema di sicurezza. Molti percorsi nel monolite funzionale non necessitano di tutte le autorizzazioni concesse.
- Più difficile da aggiornare: in un sistema di produzione, qualsiasi aggiornamento alla singola funzione è più rischioso e potrebbe causare l'interruzione dell'intera applicazione. L'aggiornamento di un singolo percorso nella funzione Lambda è un aggiornamento dell'intera funzione.
- Più difficile da mantenere: è più difficile avere più sviluppatori che lavorano sul servizio poiché si tratta di un repository di codice monolitico. Inoltre, ciò aumenta il carico cognitivo per gli sviluppatori e rende più difficile creare una copertura di test adeguata per il codice.
- Codice più difficile da riutilizzare: può essere più difficile separare le librerie riutilizzabili dai monoliti, il che rende più difficile il riutilizzo del codice. Man mano che sviluppate e supportate più progetti, ciò può rendere più difficile il supporto del codice e aumentare la velocità del team.
- Più difficile da testare: all'aumentare delle righe di codice, diventa più difficile eseguire test di unità di tutte le possibili combinazioni di input e punti di ingresso nella codebase. In genere è più semplice implementare i test unitari per servizi più piccoli con meno codice.

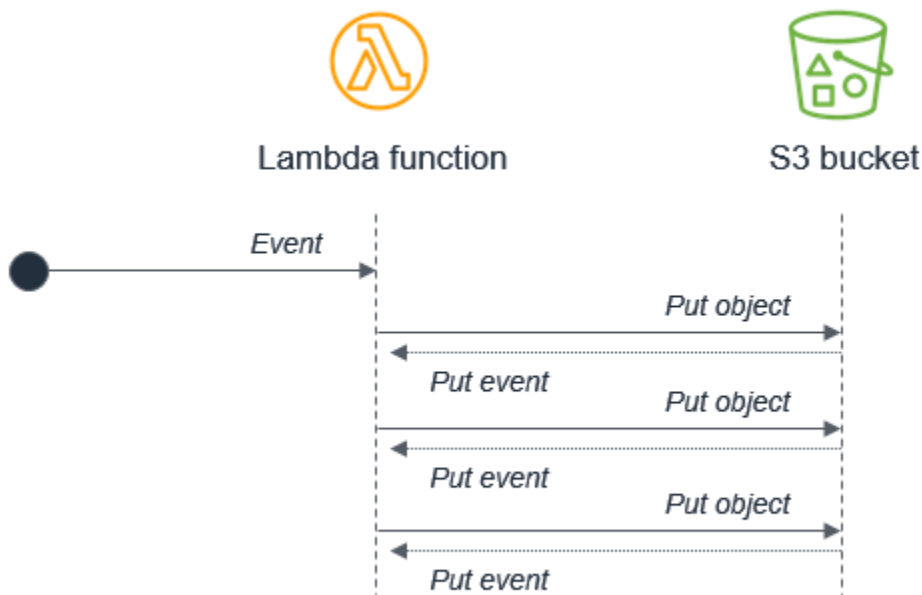
L'alternativa preferita è suddividere la funzione Lambda monolitica in singoli microservizi, mappando una singola funzione Lambda a un'unica attività ben definita. In questa semplice applicazione Web con pochi endpoint API, l'architettura risultante basata su microservizi può essere basata sui percorsi API Gateway.



### Schemi ricorsivi che causano l'inattività delle funzioni Lambda

AWS i servizi generano eventi che richiamano funzioni Lambda e le funzioni Lambda possono inviare messaggi ai servizi. AWS In genere, il servizio o la risorsa che richiama una funzione Lambda deve essere diverso dal servizio o dalla risorsa a cui la funzione invia l'output. La mancata gestione di questa situazione può comportare cicli infiniti.

Ad esempio, una funzione Lambda scrive un oggetto su un oggetto Amazon S3, che a sua volta richiama la stessa funzione Lambda utilizzando un evento put. L'invocazione fa sì che un secondo oggetto venga scritto nel bucket, che richiama la stessa funzione Lambda:



Sebbene il potenziale dei cicli infiniti esista nella maggior parte dei linguaggi di programmazione, questo anti-modello ha il potenziale di consumare più risorse nelle applicazioni serverless. Sia Lambda che Amazon S3 si scalano automaticamente in base al traffico, quindi il loop può far sì che la scalabilità di Lambda utilizzi tutta la concorrenza disponibile e Amazon S3 continui a scrivere oggetti e generare più eventi per Lambda.

Questo esempio utilizza S3 ma il rischio di cicli ricorsivi esiste anche in Amazon SNS, Amazon SQS, DynamoDB e altri servizi. È possibile utilizzare il [rilevamento ricorsivo del loop](#) per trovare ed evitare questo anti-modello.

### Funzioni Lambda che richiamano funzioni Lambda

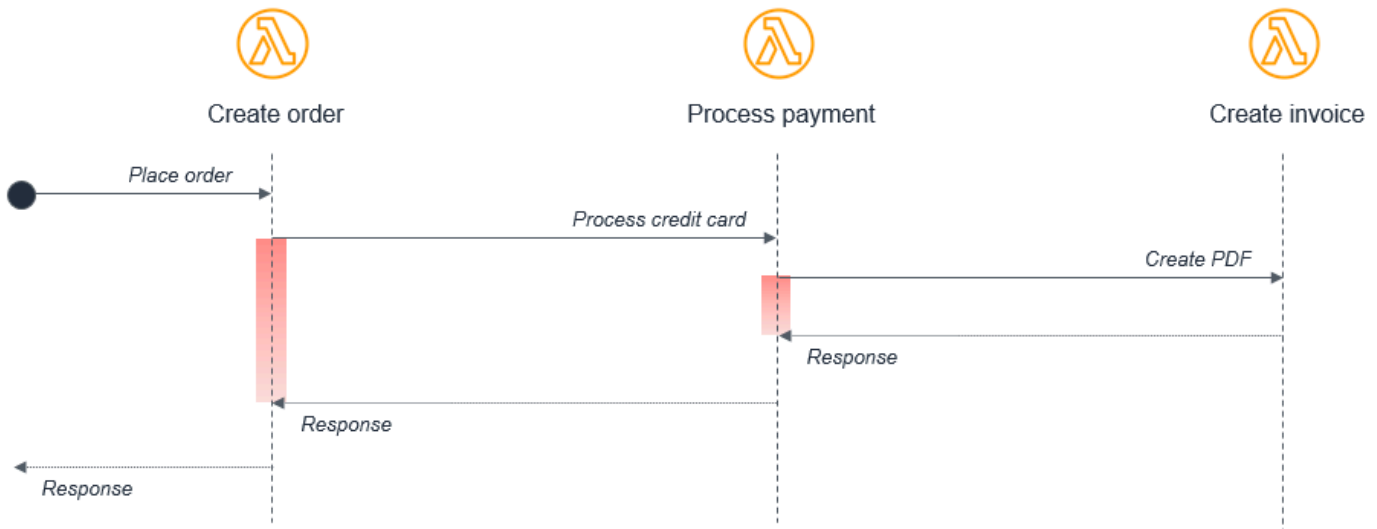
Le funzioni consentono l'incapsulamento e il riutilizzo del codice. La maggior parte dei linguaggi di programmazione supporta il concetto di codice che richiama in modo sincrono le funzioni all'interno di una codebase. In questo caso, il chiamante attende che la funzione restituisca una risposta.

#### Note

Sebbene le funzioni Lambda che richiamano direttamente altre funzioni Lambda siano generalmente contrarie a causa di problemi di costi e complessità, ciò non si applica alle funzioni [durevoli](#), che sono specificamente progettate per orchestrare flussi di lavoro in più fasi richiamando altre funzioni.

Quando ciò accade su un server tradizionale o su un'istanza virtuale, lo scheduler del sistema operativo passa ad altre attività disponibili. Il fatto che la CPU funzioni allo 0% o al 100% non influisce sul costo complessivo dell'applicazione, poiché si paga il costo fisso di proprietà e gestione di un server.

Questo modello spesso non si adatta bene allo sviluppo serverless. Ad esempio, considera una semplice applicazione di e-commerce composta da tre funzioni Lambda che elaborano un ordine:



In questo caso, la funzione Crea ordine richiama la funzione Elabora pagamento, che a sua volta richiama la funzione Crea fattura. Sebbene questo flusso sincrono possa funzionare all'interno di una singola applicazione su un server, introduce diversi problemi evitabili in un'architettura serverless distribuita:

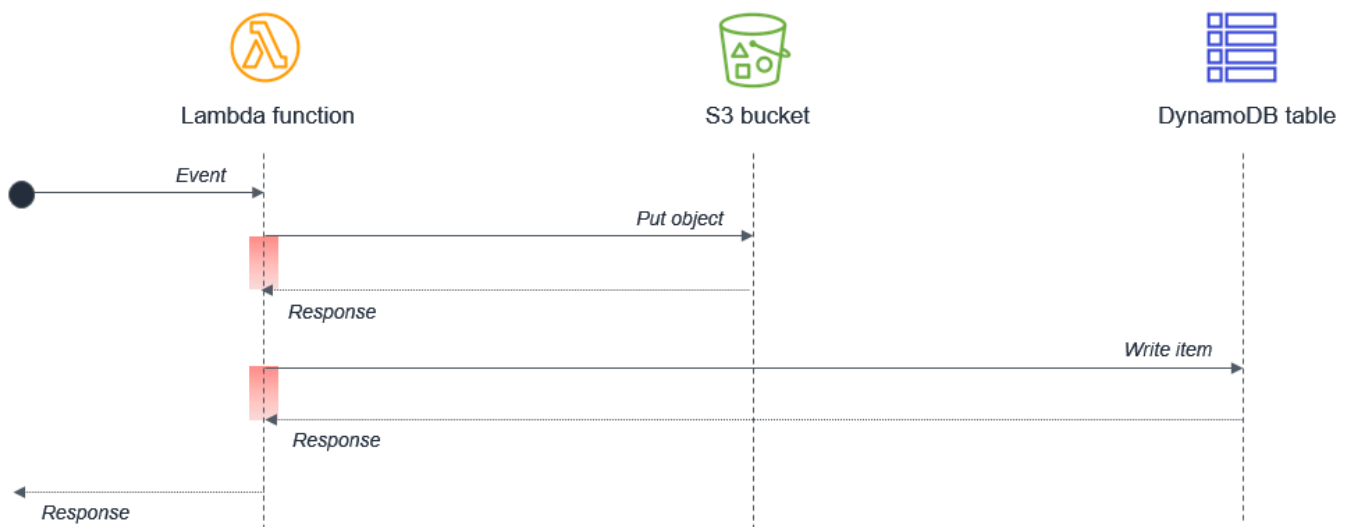
- **Costo:** con Lambda, paghi per la durata di una invocazione. In questo esempio, mentre le funzioni Crea fattura sono in esecuzione, anche altre due funzioni sono in esecuzione in stato di attesa, mostrate in rosso nel diagramma.
- **Gestione degli errori:** nelle chiamate nidificate, la gestione degli errori può diventare molto più complessa. Ad esempio, un errore in Crea fattura potrebbe richiedere la funzione Elabora pagamento per stornare l'addebito oppure potrebbe invece riprovare il processo di creazione della fattura.
- **Accoppiamento stretto:** l'elaborazione di un pagamento richiede in genere più tempo rispetto alla creazione di una fattura. In questo modello, la disponibilità dell'intero flusso di lavoro è limitata dalla funzione più lenta.
- **Scalabilità:** la [simultaneità](#) di tutte e tre le funzioni deve essere uguale. In un sistema affollato, ciò utilizza più simultaneità di quanto sarebbe altrimenti necessario.

Nelle applicazioni serverless, esistono due approcci comuni per evitare questo modello. Innanzitutto, usa una coda Amazon SQS tra le funzioni Lambda. Se un processo a valle è più lento di un processo a monte, la coda mantiene i messaggi in modo duraturo e disaccoppia le due funzioni. In questo esempio, la funzione *Crea ordine* pubblicherebbe un messaggio in una coda Amazon SQS e la funzione *Elaborazione pagamento* consuma i messaggi dalla coda.

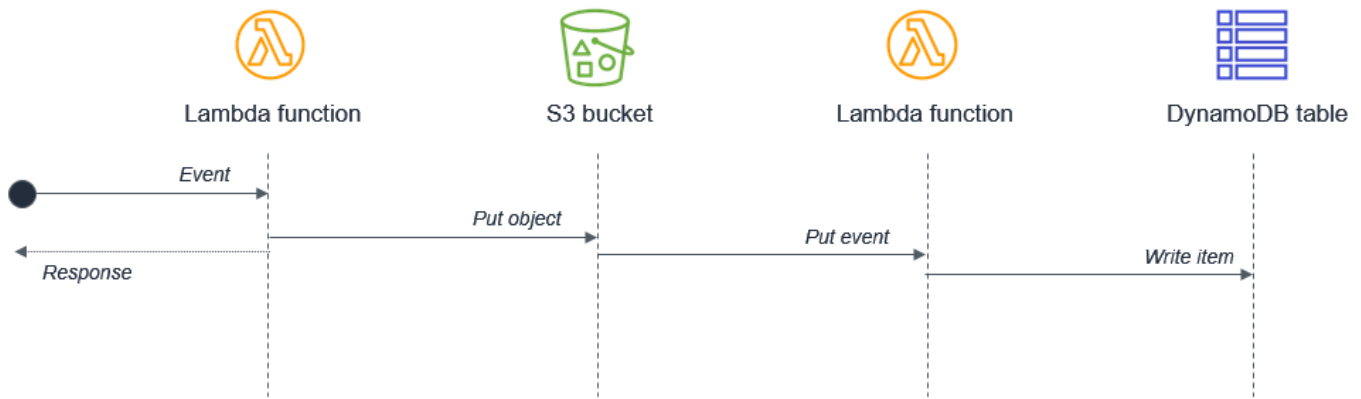
Il secondo approccio consiste nell'utilizzare AWS Step Functions. Per processi complessi con diversi tipi di logica di errore e nuovo tentativo, Step Functions può aiutare a ridurre la quantità di codice personalizzato necessario per orchestrare il flusso di lavoro. Di conseguenza, Step Functions orchestra il lavoro e gestisce in modo efficace errori e nuovi tentativi, mentre le funzioni Lambda contengono solo la logica aziendale.

Attesa sincrona all'interno di una singola funzione Lambda

Assicurati che eventuali attività potenzialmente concorrenti non siano pianificate in modo sincrono all'interno di una singola funzione Lambda. Ad esempio, una funzione Lambda potrebbe scrivere su un bucket S3 e quindi scrivere su una tabella DynamoDB:



In questa progettazione, i tempi di attesa sono composti perché le attività sono sequenziali. Nei casi in cui la seconda attività dipende dal completamento della prima, puoi ridurre il tempo di attesa totale e il costo di esecuzione utilizzando due funzioni Lambda distinte:



In questo design, la prima funzione Lambda risponde immediatamente dopo aver inserito l'oggetto nel bucket Amazon S3. Il servizio S3 richiama la seconda funzione Lambda, che quindi scrive i dati nella tabella DynamoDB. Questo approccio riduce al minimo il tempo di attesa totale nelle esecuzioni delle funzioni Lambda.



# Progettazione di applicazioni Lambda

Un'applicazione basata sugli eventi ben progettata utilizza una combinazione di AWS servizi e codice personalizzato per elaborare e gestire richieste e dati. Questo capitolo si concentra sugli Lambda-specific argomenti relativi alla progettazione di applicazioni. Ci sono numerose considerazioni importanti per gli architetti serverless quando progettano applicazioni per sistemi di produzione affollati.

Molte delle best practice che si applicano allo sviluppo di software e ai sistemi distribuiti si applicano anche allo sviluppo di applicazioni serverless. L'obiettivo generale è sviluppare carichi di lavoro che siano:

- **Affidabili:** offrendo agli utenti finali un elevato livello di disponibilità, i servizi serverless AWS sono affidabili perché progettati anche per evitare guasti.
- **Durevoli:** offrendo opzioni di archiviazione che soddisfano le esigenze di durabilità del carico di lavoro.
- **Sicuri:** seguendo le best practice e utilizzando gli strumenti forniti per proteggere l'accesso ai carichi di lavoro e limitare il raggio di esplosione.
- **Performanti:** utilizzando le risorse di elaborazione in modo efficiente e soddisfacendo le esigenze di prestazioni degli utenti finali.
- **Cost-efficient—** progettazione di architetture che evitino costi inutili, scalabili senza spese eccessive e smantellate senza spese generali significative.

I seguenti principi di progettazione possono aiutarti a creare carichi di lavoro che soddisfino questi obiettivi. Non tutti i principi potrebbero applicarsi a tutte le architetture, ma dovrebbero guidarti nelle decisioni architettoniche generali.

## Argomenti

- [Utilizzare i servizi anziché il codice personalizzato](#)
- [Informazioni sui livelli di astrazione Lambda](#)
- [Implementazione stateless nelle funzioni](#)
- [Riduzione dell'associazione](#)
- [Creazione di dati on demand anziché in batch](#)
- [Scegli un'opzione di orchestrazione per flussi di lavoro complessi](#)
- [Implementa l'idempotenza](#)

- [Usa multipli AWS account per la gestione delle quote](#)

## Utilizzare i servizi anziché il codice personalizzato

Le applicazioni serverless di solito comprendono diversi AWS servizi, integrati con codice personalizzato eseguito nelle funzioni Lambda. Sebbene Lambda possa essere integrato con la maggior parte dei AWS servizi, i servizi più comunemente usati nelle applicazioni serverless sono:

| Categoria                       | AWS servizio                       |
|---------------------------------|------------------------------------|
| Calcolo                         | AWS Lambda                         |
| Archiviazione di dati           | Simple Storage Service (Amazon S3) |
|                                 | Amazon DynamoDB                    |
|                                 | Amazon RDS                         |
| "Hello, World!"                 | Gateway Amazon API                 |
| Integrazione delle applicazioni | Amazon EventBridge                 |
|                                 | Amazon SNS                         |
|                                 | Amazon SQS                         |
| Orchestrazione                  | Funzioni durevoli Lambda           |
|                                 | AWS Step Functions                 |
| Streaming di dati e analisi     | Amazon Data Firehose               |

### Note

Molti servizi serverless forniscono replica e supporto per più regioni, tra cui DynamoDB e Amazon S3. Le funzioni Lambda possono essere implementate in più regioni come parte di una pipeline di implementazione e API Gateway può essere configurato per supportare

questa configurazione. Guarda questo [esempio di architettura](#) che mostra come raggiungere questo obiettivo.

Esistono molti modelli comuni e consolidati nelle architetture distribuite che puoi creare tu stesso o implementare utilizzando i servizi. AWS Per la maggior parte dei clienti, investire tempo per sviluppare questi modelli partendo da zero ha poco valore commerciale. Quando l'applicazione richiede uno di questi pattern, utilizzate il servizio corrispondente: AWS

| Pattern                            | AWS servizio                                   |
|------------------------------------|------------------------------------------------|
| Queue                              | Amazon SQS                                     |
| Router di eventi                   | Amazon EventBridge                             |
| Publish/subscribe (uscita ventola) | Amazon SNS                                     |
| Orchestrazione                     | Funzioni durevoli Lambda<br>AWS Step Functions |
| "Hello, World!"                    | Gateway Amazon API                             |
| Flussi di eventi                   | Amazon Kinesis                                 |

Questi servizi sono progettati per integrarsi con Lambda e puoi utilizzare l'infrastructure as code (IaC) per creare ed eliminare risorse nei servizi. Puoi utilizzare uno qualsiasi di questi servizi tramite l'[AWS SDK](#) senza dover installare applicazioni o configurare server. Acquisire esperienza nell'uso di questi servizi tramite il codice nelle funzioni Lambda è un passo importante per produrre applicazioni serverless ben progettate.

## Informazioni sui livelli di astrazione Lambda

Il servizio Lambda limita l'accesso ai sistemi operativi, agli hypervisor e all'hardware sottostanti che eseguono le funzioni Lambda. Il servizio migliora e modifica continuamente l'infrastruttura per aggiungere funzionalità, ridurre i costi e rendere il servizio più performante. Il codice non deve presupporre alcuna conoscenza dell'architettura di Lambda né presupporre alcuna affinità hardware.

Allo stesso modo, le integrazioni di Lambda con altri servizi sono gestite da AWS, con solo un numero limitato di opzioni di configurazione esposte all'utente. Ad esempio, quando API Gateway e Lambda interagiscono, non sussiste il concetto di bilanciamento del carico poiché è interamente gestito dai servizi. Inoltre, non hai alcun controllo diretto sulle [zone di disponibilità](#) utilizzate dai servizi per invocare le funzioni in qualsiasi momento, o su come Lambda determina quando aumentare o ridurre verticalmente il numero di ambienti di esecuzione.

Questa astrazione ti aiuta a concentrarti sugli aspetti di integrazione dell'applicazione, sul flusso di dati e sulla logica aziendale in cui il carico di lavoro fornisce valore agli utenti finali. Consentire ai servizi di gestire i meccanismi sottostanti consente di sviluppare applicazioni più rapidamente con meno codice personalizzato da gestire.

## Implementazione stateless nelle funzioni

Per le funzioni Lambda standard, è necessario presupporre che l'ambiente esista solo per una singola chiamata. La funzione dovrebbe inizializzare qualsiasi stato richiesto al primo avvio. Ad esempio, la tua funzione potrebbe richiedere il recupero di dati da una tabella DynamoDB. Deve eseguire eventuali modifiche permanenti ai dati in un archivio durevole come Amazon S3, DynamoDB o Amazon SQS prima dell'uscita. Non dovrebbe basarsi su strutture di dati o file temporanei esistenti o su qualsiasi stato interno che verrebbe gestito da più invocazioni.

Per inizializzare le connessioni e le librerie del database o lo stato di caricamento, è possibile sfruttare l'[inizializzazione statica](#). Poiché gli ambienti di esecuzione vengono riutilizzati ove possibile per migliorare le prestazioni, è possibile ammortizzare il tempo impiegato per inizializzare queste risorse su più invocazioni. Tuttavia, non è necessario archiviare alcuna variabile o dato utilizzato nella funzione all'interno di questo ambito globale.

## Riduzione dell'associazione

La maggior parte delle architetture dovrebbe preferire molte funzioni più brevi rispetto a un numero inferiore di funzioni più grandi. Lo scopo di ogni funzione dovrebbe essere quello di gestire l'evento trasmesso alla funzione, senza alcuna conoscenza o aspettativa del flusso di lavoro complessivo o del volume delle transazioni. Ciò rende la funzione indipendente dall'origine eventi con un abbinamento minimo ad altri servizi.

Tutte le costanti di ambito globale che cambiano di rado devono essere implementate come variabili di ambiente per consentire aggiornamenti senza implementazioni. Eventuali informazioni segrete o sensibili devono essere archiviate in [AWS Systems Manager Parameter Store](#) o [AWS Secrets Manager](#) e caricate dalla funzione. Poiché queste risorse sono specifiche dell'account, puoi creare

pipeline di compilazione su più account. Le pipeline caricano i segreti appropriati per ambiente, senza esporli agli sviluppatori o richiedere modifiche al codice.

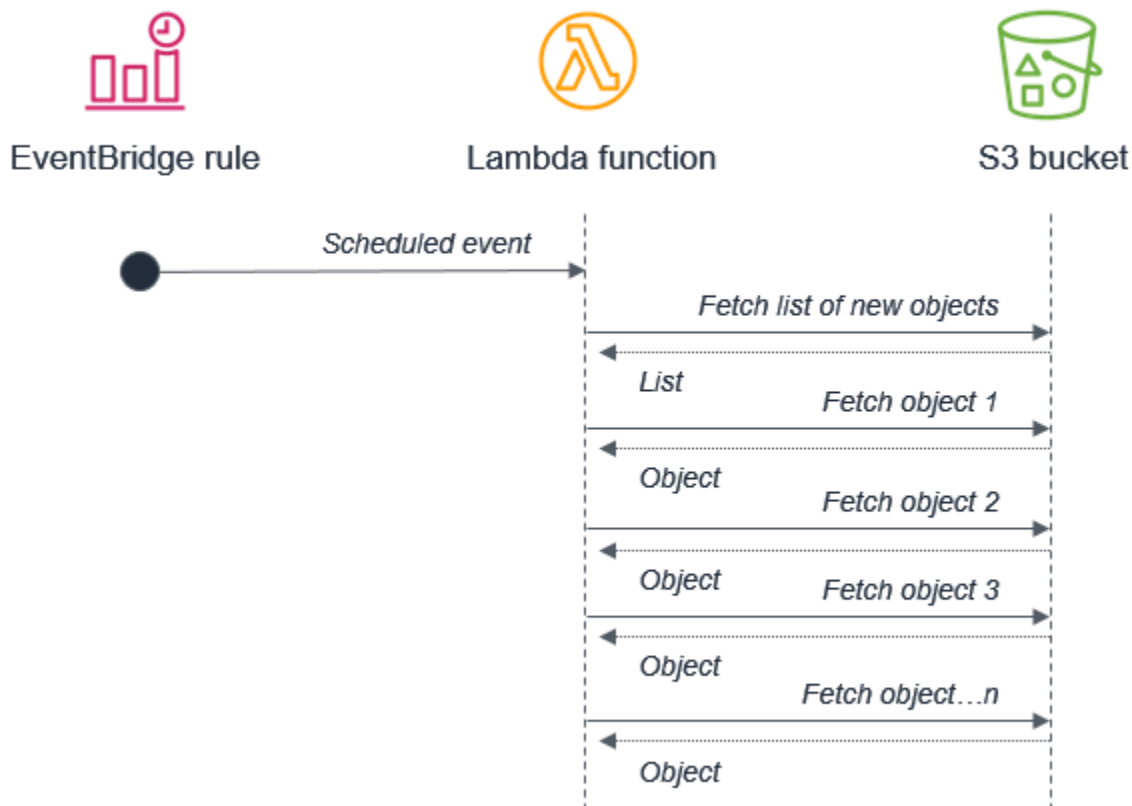
## Creazione di dati on demand anziché in batch

Molti sistemi tradizionali sono progettati per funzionare periodicamente ed elaborare batch di transazioni accumulati nel tempo. Ad esempio, un'applicazione bancaria potrebbe essere eseguita ogni ora per elaborare le transazioni ATM nei registri centrali. Nelle Lambda-based applicazioni, l'elaborazione personalizzata dovrebbe essere attivata da ogni evento, consentendo al servizio di aumentare la concorrenza in base alle esigenze, per fornire un'elaborazione delle transazioni quasi in tempo reale.

Mentre le funzioni Lambda standard sono limitate a 15 minuti di esecuzione, le Durable Functions possono essere eseguite fino a un anno, il che le rende adatte a esigenze di elaborazione più lunghe. Tuttavia, quando possibile, dovresti comunque preferire l'elaborazione basata sugli eventi rispetto all'elaborazione in batch.

Sebbene sia possibile eseguire [attività](#) cron in applicazioni serverless utilizzando espressioni [pianificate](#) per le regole in Amazon EventBridge, queste dovrebbero essere utilizzate con parsimonia o come ultima risorsa. In qualsiasi attività pianificata che elabora un batch, esiste la possibilità che il volume delle transazioni cresca oltre quanto può essere elaborato entro il limite di durata Lambda di 15 minuti. Se le limitazioni dei sistemi esterni ti obbligano a utilizzare uno scheduler, in genere dovresti programmare per il periodo di tempo ricorrente più breve e ragionevole.

Ad esempio, non è consigliabile utilizzare un processo batch che attiva una funzione Lambda per recuperare un elenco di nuovi oggetti Amazon S3. Questo perché il servizio potrebbe ricevere più nuovi oggetti tra un batch e l'altro di quanti possano essere elaborati in una funzione Lambda di 15 minuti.



Invece, Amazon S3 deve invocare la funzione Lambda ogni volta che un nuovo oggetto viene inserito nel bucket. Questo approccio è significativamente più scalabile e funziona quasi in tempo reale.



## Scegli un'opzione di orchestrazione per flussi di lavoro complessi

I flussi di lavoro che coinvolgono la logica di ramificazione, diversi tipi di modelli di errore e la logica dei tentativi in genere utilizzano un orchestratore per tenere traccia dello stato dell'esecuzione complessiva. Non creare un'orchestrazione ad hoc nelle funzioni Lambda standard. Ciò comporta un accoppiamento stretto, un codice di routing complesso e nessun ripristino automatico dello stato.

Utilizzate invece una di queste opzioni di orchestrazione appositamente create:

- [Funzioni durevoli Lambda](#): Application-centric orchestrazione utilizzando linguaggi di programmazione standard con checkpoint automatico, riprova integrata e ripristino dell'esecuzione.

Ideale per gli sviluppatori che preferiscono mantenere la logica del flusso di lavoro nel codice insieme alla logica aziendale all'interno di Lambda.

- [AWS Step Functions](#): Orchestrazione visiva del flusso di lavoro con integrazioni native per oltre 220 servizi. AWS Ideale per il coordinamento multiservizio, l'infrastruttura a manutenzione zero e la progettazione visiva del flusso di lavoro.

Per indicazioni sulla scelta tra queste opzioni, consulta Funzioni [durevoli o Step Functions](#).

Con [Step Functions](#), si utilizzano macchine a stati per gestire l'orchestrazione. Questo estrae la gestione degli errori, il routing e la logica di ramificazione dal codice, sostituendola con macchine a stati dichiarate utilizzando JSON. Oltre a rendere i flussi di lavoro più robusti e osservabili, puoi anche aggiungere il controllo delle versioni ai flussi di lavoro e rendere la macchina a stati una risorsa codificata da aggiungere a un archivio di codice.

È normale che i flussi di lavoro più semplici nelle funzioni Lambda diventino più complessi nel tempo. Quando si utilizza un'applicazione serverless di produzione, è importante identificare quando ciò accade, in modo da poter migrare questa logica su una macchina a stati o su una funzione durevole.

## Implementa l'idempotenza

AWS i servizi serverless, incluso Lambda, sono tolleranti ai guasti e progettati per gestire i guasti. Ad esempio, se un servizio richiama una funzione Lambda e si verifica un'interruzione del servizio, Lambda invoca la funzione in una zona di disponibilità diversa. Se la funzione emette un errore, Lambda ritenta l'invocazione.

Poiché lo stesso evento potrebbe essere ricevuto più di una volta, le funzioni devono essere progettate per essere idempotenti. <https://en.wikipedia.org/wiki/Idempotence> Ciò significa che ricevere lo stesso evento più volte non modifica il risultato dopo la prima ricezione dell'evento.

Puoi implementare l'idempotenza nelle funzioni Lambda utilizzando una tabella DynamoDB per tenere traccia degli identificatori elaborati di recente per determinare se la transazione è già stata gestita in precedenza. La tabella DynamoDB di solito implementa un valore [Time To Live \(TTL\)](#) per gli elementi con scadenza per limitare lo spazio di archiviazione utilizzato.

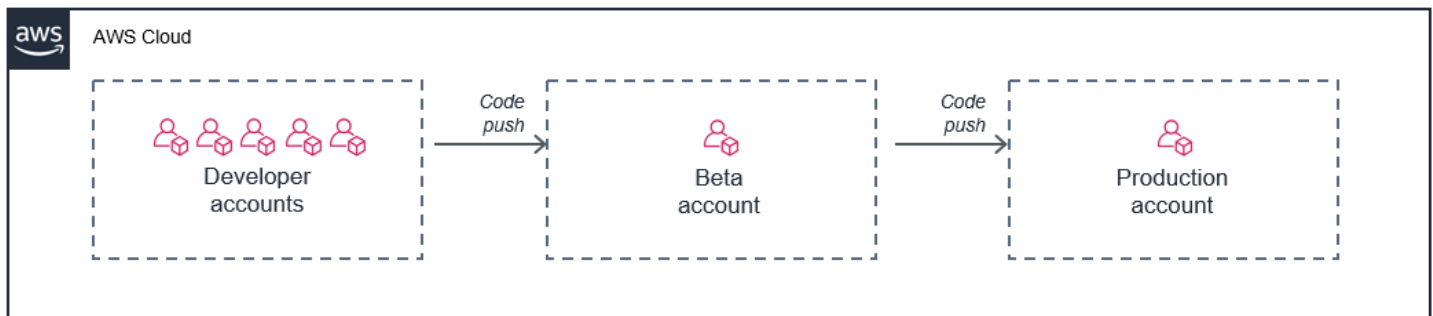
## Usa multipli AWS account per la gestione delle quote

Molte quote [di servizio](#) in AWS sono impostate a livello di account. Ciò significa che man mano che aggiungi più carichi di lavoro, puoi esaurire rapidamente i tuoi limiti.

Un modo efficace per risolvere questo problema è utilizzare più AWS account, dedicando ogni carico di lavoro al proprio account. Ciò impedisce che le quote vengano condivise con altri carichi di lavoro o risorse non di produzione.

In aggiunta, utilizzando [Organizzazioni AWS](#), puoi gestire centralmente la fatturazione, la conformità e la sicurezza di questi account. È possibile collegare policy a gruppi di account per evitare script personalizzati e processi manuali.

Un approccio comune consiste nel fornire a ogni sviluppatore un AWS account e quindi utilizzare account separati per la fase di distribuzione beta e la produzione:



In questo modello, ogni sviluppatore ha i propri limiti per l'account, quindi il loro utilizzo non ha alcun impatto sull'ambiente di produzione. Questo approccio consente inoltre agli sviluppatori di testare le funzioni Lambda localmente sulle proprie macchine di sviluppo con risorse cloud attive nei propri account individuali.



# Crea la tua prima funzione Lambda

Per iniziare a utilizzare Lambda, usa la console Lambda per creare una funzione. In pochi minuti puoi creare e implementare una funzione, e testarla nella console.

Mano a mano che esegui il tutorial, apprendrai alcuni concetti fondamentali di Lambda, ad esempio come passare argomenti a una funzione tramite l'oggetto evento di Lambda. Scoprirai anche come restituire gli output dei log dalla tua funzione e come visualizzare i log di invocazione della tua funzione in Amazon Logs. CloudWatch

Per semplificare le cose, crei la tua funzione usando Python o il runtime. Node.js Con questi linguaggi interpretati, puoi modificare il codice della funzione direttamente nell'editor del codice integrato della console. Con linguaggi compilati come Java e C#, è necessario creare un pacchetto di distribuzione sulla macchina di compilazione locale e caricarlo su Lambda. Per informazioni sull'implementazione di funzioni in Lambda tramite altri runtime, consulta i link nella sezione [the section called “Fasi successive”](#).



Per scoprire come creare soluzioni serverless, consulta la [Guida allo sviluppo serverless](#).

## Registrati per un Account AWS

Per iniziare AWS, hai bisogno di un Account AWS. Per informazioni sulla creazione di un Account AWS, vedere Guida [introduttiva a Account AWS](#) nella Guida Gestione dell'account AWS di riferimento.

## Creare una funzione Lambda con la console

In questo esempio, la funzione acquisisce un oggetto JSON contenente due valori interi etichettati "length" e "width". La funzione moltiplica tali valori per calcolare un'area e la restituisce come stringa JSON.

La tua funzione stampa anche l'area calcolata, insieme al nome del relativo gruppo di CloudWatch log. Più avanti nel tutorial, imparerai a usare [CloudWatch i log](#) per visualizzare i record delle chiamate delle tue funzioni.

Per creare una funzione Lambda Hello world con la console

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli Crea funzione.
3. Scegli Crea da zero.
4. Nel riquadro Informazioni base, per Nome funzione inserisci **myLambdaFunction**.
5. Per Runtime, scegli Node.js 24 o Python 3.14.
6. Lascia l'architettura impostata su x86\_64 e scegli Crea funzione.

Oltre a una funzione semplice che restituisce il messaggio `Hello from Lambda!`, Lambda crea anche un [ruolo di esecuzione](#) per la tua funzione. Un ruolo di esecuzione è un ruolo AWS Identity and Access Management (IAM) che concede a una funzione Lambda l'autorizzazione all'accesso e alle risorse. Servizi AWS Per la tua funzione, il ruolo creato da Lambda concede le autorizzazioni di base per scrivere su Logs. CloudWatch

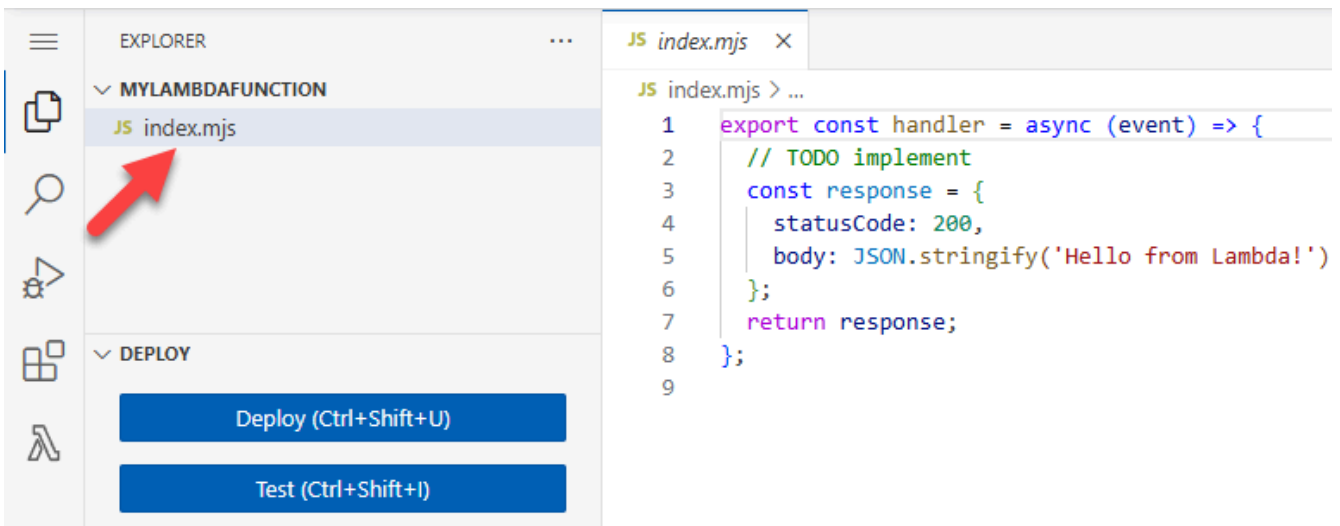
Utilizza l'editor di codice integrato della console per sostituire il codice Hello world creato da Lambda con il codice della tua funzione.

Node.js

Per modificare il codice nella console

1. Scegli la scheda Codice.

Nell'editor di codice integrato della console, dovresti vedere il codice della funzione creato da Lambda. Se non vedi la scheda `index.js` nell'editor di codice, seleziona `index.js` in Esplora file, come illustrato nel diagramma seguente.



2. Incolla il codice seguente nella scheda index.js, sostituendo il codice creato da Lambda.

```
export const handler = async (event, context) => {

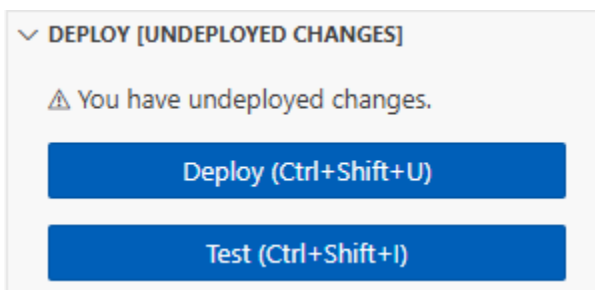
  const length = event.length;
  const width = event.width;
  let area = calculateArea(length, width);
  console.log(`The area is ${area}`);

  console.log('CloudWatch log group: ', context.logGroupName);

  let data = {
    "area": area,
  };
  return JSON.stringify(data);

  function calculateArea(length, width) {
    return length * width;
  }
};
```

3. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



## Comprendere il codice della funzione

Prima di passare alla fase successiva, esaminiamo il codice della funzione e apprendiamo alcuni concetti chiave di Lambda.

- Il gestore Lambda:

La tua funzione Lambda contiene una funzione denominata `Node.js handler`. Una funzione Lambda in Node.js può contenere più di una Node.js funzione, ma la funzione `handler` è sempre il punto di ingresso al codice. Quando viene richiamata la funzione, Lambda esegue questo metodo.

Una volta creata la funzione Hello world utilizzando la console, Lambda imposta automaticamente su `handler` il nome del metodo del gestore per la funzione. Assicurati di non modificare il nome di questa Node.js funzione. In tal caso, Lambda non sarà in grado di eseguire il codice quando richiami la funzione.

Per saperne di più sul gestore Lambda in, consulta. Node.js [the section called “Gestore”](#)

- L'oggetto evento Lambda:

La funzione `handler` acquisisce due argomenti, `event` e `context`. Un evento in Lambda è un documento in formato JSON contenente i dati che la funzione deve elaborare.

Se la tua funzione viene richiamata da un'altra Servizio AWS, l'oggetto evento contiene informazioni sull'evento che ha causato l'invocazione. Ad esempio, se la funzione viene richiamata quando un oggetto viene caricato in un bucket Amazon Simple Storage Service (Amazon S3), l'evento conterrà il nome del bucket e la chiave dell'oggetto.

In questo esempio, creerai un evento nella console inserendo un documento in formato JSON con due coppie chiave-valore.

- L'oggetto contestuale Lambda:

Il secondo argomento acquisito dalla funzione è `context`. Lambda passa l'oggetto contestuale alla tua funzione automaticamente. L'oggetto contestuale contiene informazioni sull'invocazione della funzione e sull'ambiente di esecuzione.

Puoi utilizzare l'oggetto contestuale per generare informazioni sull'invocazione della funzione a scopo di monitoraggio. In questo esempio, la funzione utilizza il `logGroupName` parametro per restituire il nome del relativo gruppo di log. CloudWatch

Per saperne di più sull'oggetto di contesto Lambda in Node.js, vedi [the section called “Contesto”](#).

- Accesso a Lambda:

Con Node.js, puoi utilizzare metodi da console come `console.log` e `console.error` per inviare informazioni al registro della tua funzione. Il codice di esempio utilizza `console.log` le istruzioni per restituire l'area calcolata e il nome del gruppo CloudWatch Logs della funzione. Puoi utilizzare anche qualunque libreria di log che scrive in `stdout` o `stderr`.

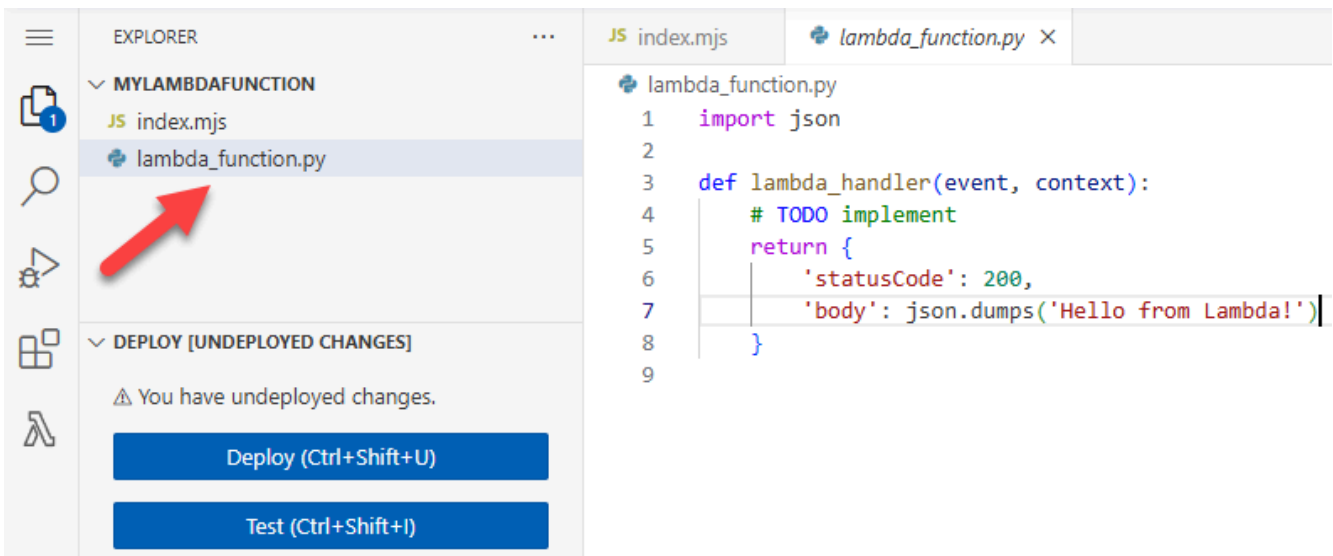
Per ulteriori informazioni, consulta [the section called “Registrazione dei log”](#). Per informazioni sui log in altri runtime, consulta le pagine “Creazione con” per i runtime a cui sei interessato.

## Python

Per modificare il codice nella console

1. Scegli la scheda Codice.

Nell'editor di codice integrato della console, dovresti vedere il codice della funzione creato da Lambda. Se non vedi la scheda `lambda_function.py` nell'editor di codice, seleziona `lambda_function.py` in Esplora file, come illustrato nel diagramma seguente.



2. Incolla il codice seguente nella scheda `lambda_function.py`, sostituendo il codice creato da Lambda.

```
import json
import logging
```

```
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Get the length and width parameters from the event object. The
    # runtime converts the event object to a Python dictionary
    length = event['length']
    width = event['width']

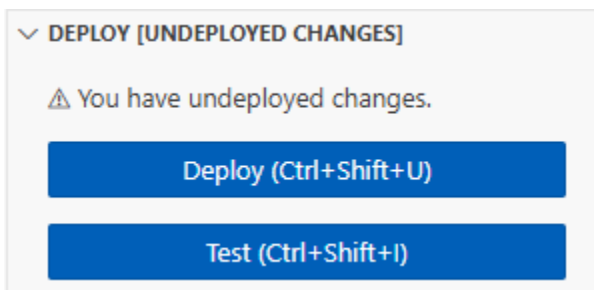
    area = calculate_area(length, width)
    print(f"The area is {area}")

    logger.info(f"CloudWatch logs group: {context.log_group_name}")

    # return the calculated area as a JSON string
    data = {"area": area}
    return json.dumps(data)

def calculate_area(length, width):
    return length*width
```

3. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



## Comprendere il codice della funzione

Prima di passare alla fase successiva, esaminiamo il codice della funzione e apprendiamo alcuni concetti chiave di Lambda.

- Il gestore Lambda:

La funzione Lambda contiene una funzione Python denominata `lambda_handler`. Una funzione Lambda in Python può contenere più di una funzione Python, ma la funzione del

gestore è sempre il punto di ingresso al codice. Quando viene richiamata la funzione, Lambda esegue questo metodo.

Una volta creata la funzione Hello world utilizzando la console, Lambda imposta automaticamente su `lambda_handler` il nome del metodo del gestore per la funzione. Non modificare il nome di questa funzione Python. In tal caso, Lambda non sarà in grado di eseguire il codice quando richiami la funzione.

Per ulteriori informazioni sul gestore Lambda in Python, consulta [the section called “Gestore”](#).

- L'oggetto evento Lambda:

La funzione `lambda_handler` acquisisce due argomenti, `event` e `context`. Un evento in Lambda è un documento in formato JSON contenente i dati che la funzione deve elaborare.

Se la tua funzione viene richiamata da un'altra Servizio AWS, l'oggetto evento contiene informazioni sull'evento che ha causato l'invocazione. Ad esempio, se la funzione viene richiamata quando un oggetto viene caricato in un bucket Amazon Simple Storage Service (Amazon S3), l'evento conterrà il nome del bucket e la chiave dell'oggetto.

In questo esempio, creerai un evento nella console inserendo un documento in formato JSON con due coppie chiave-valore.

- L'oggetto contestuale Lambda:

Il secondo argomento acquisito dalla funzione è `context`. Lambda passa l'oggetto contestuale alla tua funzione automaticamente. L'oggetto contestuale contiene informazioni sull'invocazione della funzione e sull'ambiente di esecuzione.

Puoi utilizzare l'oggetto contestuale per generare informazioni sull'invocazione della funzione a scopo di monitoraggio. In questo esempio, la funzione utilizza il `log_group_name` parametro per restituire il nome del relativo gruppo di log. CloudWatch

Per ulteriori informazioni sull'oggetto contestuale Lambda in Python, consulta [the section called “Contesto”](#).

- Accesso a Lambda:

Con Python, puoi usare un'istruzione `print` o una libreria di log Python per inviare informazioni al log della funzione. Per illustrare la differenza tra i dati acquisiti, il codice di esempio utilizza entrambi i metodi. In un'applicazione in produzione, è preferibile utilizzare una libreria di log.

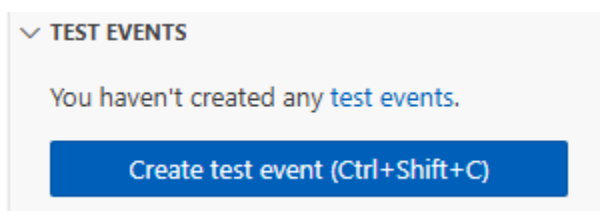
Per ulteriori informazioni, consulta [the section called “Registrazione dei log”](#). Per informazioni sui log in altri runtime, consulta le pagine “Creazione con” per i runtime a cui sei interessato.

## Richiamare la funzione Lambda tramite l'editor di codice della console

Per richiamare la funzione utilizzando l'editor di codice della console Lambda, devi prima creare un evento di test da inviare alla tua funzione. L'evento è un documento in formato JSON contenente due coppie chiave-valore con le chiavi "length" e "width".

Per creare un evento di test

1. Nella sezione TEST EVENTS dell'editor di codice della console, scegli Crea evento di test.



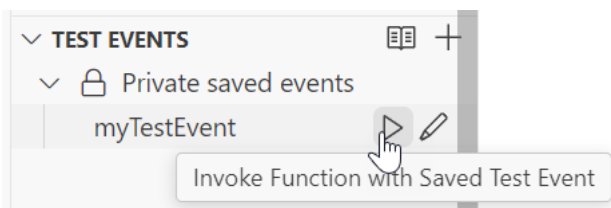
2. Per Event name (Nome evento), immettere **myTestEvent**.
3. Nella sezione JSON dell'evento, sostituisci il codice JSON predefinito con il seguente:

```
{
  "length": 6,
  "width": 7
}
```

4. Selezionare Salva.

Per testare la funzione e visualizzare i record di invocazione nella console

Nella sezione TEST EVENTS dell'editor di codice della console, scegli l'icona di esecuzione accanto all'evento di test:





Al termine dell'esecuzione della funzione, verranno visualizzati i log della risposta e della funzione nella scheda OUTPUT. Sono visualizzati risultati simili ai seguenti:

## Node.js

```
Status: Succeeded
Test Event Name: myTestEvent

Response
"{\"area\":42}"

Function Logs
START RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a Version: $LATEST
2024-08-31T23:39:45.313Z 5c012b0a-18f7-4805-b2f6-40912935034a INFO The area is 42
2024-08-31T23:39:45.331Z 5c012b0a-18f7-4805-b2f6-40912935034a INFO CloudWatch log
group: /aws/lambda/myLambdaFunction
END RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a
REPORT RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a Duration: 20.67 ms Billed
Duration: 21 ms Memory Size: 128 MB Max Memory Used: 66 MB Init Duration: 163.87 ms

Request ID
5c012b0a-18f7-4805-b2f6-40912935034a
```

## Python

```
Status: Succeeded
Test Event Name: myTestEvent

Response
"{\"area\": 42}"

Function Logs
START RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b Version: $LATEST
The area is 42
[INFO] 2024-08-31T23:43:26.428Z 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b CloudWatch logs
group: /aws/lambda/myLambdaFunction
END RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b
REPORT RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b Duration: 1.42 ms Billed
Duration: 2 ms Memory Size: 128 MB Max Memory Used: 39 MB Init Duration: 123.74 ms

Request ID
2d0b1579-46fb-4bf7-a6e1-8e08840eae5b
```

Quando richiami la tua funzione all'esterno della console Lambda, devi usare CloudWatch Logs per visualizzare i risultati di esecuzione della funzione.

Per visualizzare i record di chiamata della funzione in Logs CloudWatch

1. Apri la [pagina dei gruppi di log](#) della CloudWatch console.
2. Scegli il nome del gruppo di log per la funzione (/aws/lambda/myLambdaFunction). Questo è il nome del gruppo di log che la funzione ha stampato sulla console.
3. Scorri verso il basso e scegli il flusso di log per le invocazioni delle funzioni che desideri esaminare.

| Log streams (14)                                                     |                                           |                                      | Delete                                | Create log stream    | Search all log streams |
|----------------------------------------------------------------------|-------------------------------------------|--------------------------------------|---------------------------------------|----------------------|------------------------|
| <input type="text" value="Filter log streams or try prefix search"/> |                                           | <input type="checkbox"/> Exact match | <input type="checkbox"/> Show expired | <a href="#">Info</a> | < 1 >                  |
| <input type="checkbox"/>                                             | Log stream                                | Last event time                      |                                       |                      |                        |
| <input type="checkbox"/>                                             | <a href="#">2024/04/30/[\$LATEST]e0fa</a> | 2024-04-30 17:24:16 (UTC)            |                                       |                      |                        |
| <input type="checkbox"/>                                             | <a href="#">2024/04/19/[\$LATEST]e9a</a>  | 2024-04-19 20:59:06 (UTC)            |                                       |                      |                        |
| <input type="checkbox"/>                                             | <a href="#">2024/02/22/[\$LATEST]cf0</a>  | 2024-02-22 18:38:41 (UTC)            |                                       |                      |                        |
| <input type="checkbox"/>                                             | <a href="#">2024/02/21/[1]d132c4d</a>     | 2024-02-21 21:37:01 (UTC)            |                                       |                      |                        |
| <input type="checkbox"/>                                             | <a href="#">2024/02/21/[1]5ad</a>         | 2024-02-21 21:37:01 (UTC)            |                                       |                      |                        |

Verrà visualizzato un output simile al seguente:

Node.js

```
INIT_START Runtime Version: nodejs:22.v13    Runtime Version ARN:
arn:aws:lambda:us-
west-2::runtime:e3aaabf6b92ef8755eaae2f4bfdbc7eb8c4536a5e044900570a42bdba7b869d9
START RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20 Version: $LATEST
2024-08-23T22:04:15.809Z    5c012b0a-18f7-4805-b2f6-40912935034a    INFO The area
is 42
2024-08-23T22:04:15.810Z    aba6c0fc-cf99-49d7-a77d-26d805dacd20    INFO
CloudWatch log group: /aws/lambda/myLambdaFunction
END RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20
REPORT RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20    Duration: 17.77 ms
Billed Duration: 18 ms    Memory Size: 128 MB    Max Memory Used: 67 MB    Init
Duration: 178.85 ms
```

## Python

```
INIT_START Runtime Version: python:3.13.v16    Runtime Version ARN:
arn:aws:lambda:us-
west-2::runtime:ca202755c87b9ec2b58856efb7374b4f7b655a0ea3deb1d5acc9aee9e297b072
START RequestId: 9d4096ee-acb3-4c25-be10-8a210f0a9d8e Version: $LATEST
The area is 42
[INFO] 2024-09-01T00:05:22.464Z 9315ab6b-354a-486e-884a-2fb2972b7d84 CloudWatch
logs group: /aws/lambda/myLambdaFunction
END RequestId: 9d4096ee-acb3-4c25-be10-8a210f0a9d8e
REPORT RequestId: 9d4096ee-acb3-4c25-be10-8a210f0a9d8e    Duration: 1.15 ms
Billed Duration: 2 ms    Memory Size: 128 MB    Max Memory Used: 40 MB
```

## Eliminazione

Quando hai terminato il lavoro con la funzione di esempio, eliminala. Puoi anche eliminare il gruppo di log che memorizza i log della funzione e il [ruolo di esecuzione](#) creato dalla console.

Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **confirm** nel campo di immissione del testo, quindi scegli Elimina.

Come eliminare il gruppo di log

1. Apri la pagina [Log groups page \(Pagina dei gruppi di log\)](#) della console CloudWatch.
2. Scegli il gruppo di log della funzione (/aws/lambda/myLambdaFunction).
3. Scegli Actions (Azioni), Delete log group(s) (Elimina gruppo/i di log).
4. Nella finestra di dialogo Delete log group(s) (Elimina gruppo/i di log) scegli Delete (Elimina).

Come eliminare il ruolo di esecuzione

1. Apri la pagina [Ruoli](#) della console AWS Identity and Access Management (IAM).

2. Seleziona il ruolo di esecuzione della funzione (ad esempio, `myLambdaFunction-role-31exxmpl`).
3. Scegliere Elimina.
4. Nella finestra di dialogo Elimina ruolo, immetti il nome del ruolo, quindi scegli Elimina.

## Risorse aggiuntive e fasi successive

Ora che hai creato e testato una semplice funzione Lambda utilizzando la console, procedi nel seguente modo:

- Scopri come aggiungere dipendenze alla tua funzione e implementala utilizzando un pacchetto di implementazione .zip. Scegli la lingua preferita dai seguenti link.

Node.js

[the section called “Implementazione di archivi di file .zip”](#)

Typescript

[the section called “Implementare archivi di file .zip”](#)

Python

[the section called “Implementazione di archivi di file .zip”](#)

Ruby

[the section called “Implementazione di archivi di file .zip”](#)

Java

[the section called “Implementazione di archivi di file .zip”](#)

Go

[the section called “Implementazione di archivi di file .zip”](#)

C#

[the section called “Pacchetto di implementazione”](#)

- Per sapere come richiamare una funzione Lambda usandone un'altra Servizio AWS, consulta. [Tutorial: uso di un trigger Amazon S3 per richiamare una funzione Lambda](#)
- Scegli uno dei seguenti tutorial per esempi più complessi di utilizzo di Lambda con altri Servizi AWS.

- [Tutorial: uso di Lambda con Amazon API Gateway](#): crea una REST API di Gateway Amazon API che richiama una funzione Lambda.
- [Utilizzo di una funzione Lambda per accedere a un database Amazon RDS](#): utilizza una funzione Lambda per scrivere dati in un database Amazon Relational Database Service (Amazon RDS) tramite un proxy RDS.
- [Utilizzo di un trigger Amazon S3 per creare anteprime](#): utilizza una funzione Lambda per creare un'anteprima ogni volta che un file immagine viene caricato in un bucket Amazon S3.

## Guida introduttiva ad applicazioni e modelli di esempio

Le seguenti risorse possono essere utilizzate per creare e distribuire rapidamente app serverless che implementano alcuni casi d'uso comuni di Lambda. Per ciascuna delle app di esempio, forniamo istruzioni per creare e configurare le risorse manualmente utilizzando o per Console di gestione AWS utilizzarle AWS Serverless Application Model per distribuire le risorse utilizzando IaC. Segui le istruzioni della console per saperne di più sulla configurazione delle singole AWS risorse per ciascuna app o usala per distribuire rapidamente le risorse come faresti in un ambiente di produzione.

AWS SAM

### Elaborazione dei file

- [Applicazione di crittografia PDF](#): crea un'applicazione serverless che crittografa i file PDF quando vengono caricati in un bucket Amazon Simple Storage Service e li salva in un altro bucket, utile per proteggere i documenti sensibili al momento del caricamento.
- [Applicazione di analisi delle immagini](#): crea un'applicazione serverless che estrae il testo dalle immagini utilizzando Amazon Rekognition, utile per l'elaborazione dei documenti, la moderazione dei contenuti e l'analisi automatica delle immagini.

### Integrazione del database

- [Queue-to-Database Applicazione](#): crea un'applicazione serverless che scrive messaggi di coda su un database Amazon RDS, utile per elaborare le registrazioni degli utenti e gestire l'invio degli ordini.

- [Database Event Handler](#) : crea un'applicazione serverless che risponda alle modifiche alle tabelle di Amazon DynamoDB, utile per la registrazione di audit, la replica dei dati e i flussi di lavoro automatizzati.

## Attività pianificate

- [Applicazione di manutenzione del database](#) : crea un'applicazione serverless che elimina automaticamente le voci più vecchie di 12 mesi da una tabella Amazon DynamoDB in base a una pianificazione cron. Ciò è utile per la manutenzione automatizzata del database e la gestione del ciclo di vita dei dati.
- [Crea una EventBridge pianificazione dello Scheduler per le funzioni Lambda](#) : usa EventBridge Scheduler per richiamare una funzione Lambda su una pianificazione unica o ricorrente. EventBridge Scheduler supporta espressioni cron e rate, finestre temporali flessibili, criteri per i nuovi tentativi e code di lettere morte. Puoi creare una pianificazione direttamente dalla console Lambda utilizzando il flusso di trigger Add o dalla console Scheduler. EventBridge

## Risorse aggiuntive

Usa le seguenti risorse per esplorare ulteriormente lo sviluppo di applicazioni Lambda e serverless:

- [Serverless Land](#) : una libreria di modelli pronti all'uso per la creazione di app serverless. Aiuta gli sviluppatori a creare applicazioni più velocemente utilizzando AWS servizi come Lambda, API Gateway e EventBridge. Il sito offre soluzioni predefinite e best practice, che semplificano lo sviluppo di sistemi serverless.
- [Applicazioni di esempio Lambda](#) : applicazioni disponibili nel GitHub repository di questa guida. Questi esempi dimostrano l'uso di vari linguaggi e AWS servizi. Ogni applicazione di esempio include script per semplificare l'implementazione, la pulizia e il supporto delle risorse.
- [Esempi di codice per Lambda tramite AWS SDK](#) : esempi che mostrano come utilizzare Lambda con kit di sviluppo AWS software (SDK). Questi esempi includono nozioni di base, azioni, scenari e contributi della community. AWS Gli esempi riguardano le operazioni essenziali, le singole funzioni di servizio e le attività specifiche che utilizzano più funzioni o AWS servizi.

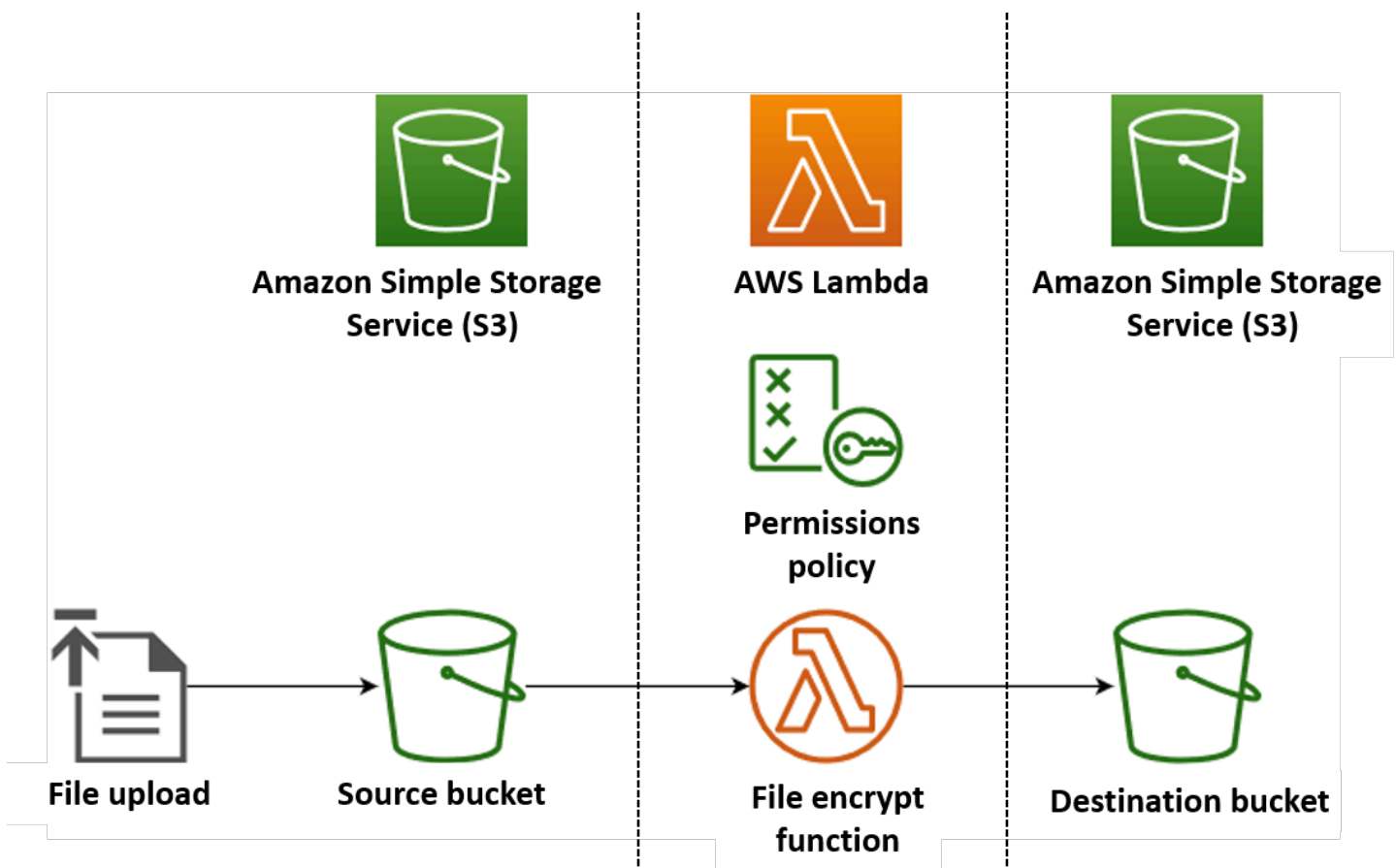
## Creare un'app di elaborazione file serverless

Uno dei casi d'uso più comuni di Lambda è l'esecuzione di attività di elaborazione dei file. Ad esempio, è possibile utilizzare una funzione Lambda per creare automaticamente file PDF da file o immagini HTML o per creare miniature quando un utente carica un'immagine.

In questo esempio, viene creata un'app che crittografa automaticamente i file PDF quando vengono caricati in un bucket Amazon Simple Storage Service (Amazon S3). Per creare questa app, crea le risorse seguenti:

- Un bucket S3 in cui gli utenti possono caricare i file PDF
- Una funzione Lambda in Python che legge il file caricato e ne crea una versione crittografata e protetta da password
- Un secondo bucket S3 per Lambda in cui salvare il file crittografato

Crei anche una policy AWS Identity and Access Management (IAM) per autorizzare la tua funzione Lambda a eseguire operazioni di lettura e scrittura sui tuoi bucket S3.



 Tip

Se non conosci Lambda, ti consigliamo di iniziare con il tutorial [the section called “Crea la tua prima funzione”](#) prima di creare questa app di esempio.

Puoi distribuire la tua app manualmente creando e configurando le risorse con Console di gestione AWS o the AWS Command Line Interface ().AWS CLI Puoi anche distribuire l'app utilizzando (). AWS Serverless Application Model AWS SAM AWS SAM è uno strumento Infrastructure as code (IaC). Con IaC, non crei risorse manualmente, ma le definisci in codice e poi le distribuisce automaticamente.

Se desideri saperne di più sull'utilizzo di Lambda con IaC prima di implementare questa app di esempio, consulta [the section called “Infrastructure as code \(IaC\)”](#).

## Crea i file di codice sorgente della funzione Lambda

Create i seguenti file nella cartella del progetto:

- `lambda_function.py`: il codice della funzione Python per la funzione Lambda che esegue la crittografia del database
- `requirements.txt`: un file manifesto che definisce le dipendenze richieste dal codice della funzione Python

Espandi le seguenti sezioni per visualizzare il codice e per saperne di più sul ruolo di ciascun file. Per creare i file sul tuo computer locale, copia e incolla il codice seguente oppure scarica i file dal repository [GitHub aws-lambda-developer-guide](#).

### Codice della funzione Python

Copia e incolla il codice seguente in un nuovo file `lambda_function.py`.

```
from pypdf import PdfReader, PdfWriter
import uuid
import os
from urllib.parse import unquote_plus
import boto3

# Create the S3 client to download and upload objects from S3
s3_client = boto3.client('s3')
```



```

def lambda_handler(event, context):
    # Iterate over the S3 event object and get the key for all uploaded files
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key']) # Decode the S3 object key to
        remove any URL-encoded characters
        download_path = f'/tmp/{uuid.uuid4()}.pdf' # Create a path in the Lambda tmp
        directory to save the file to
        upload_path = f'/tmp/converted-{uuid.uuid4()}.pdf' # Create another path to
        save the encrypted file to

        # If the file is a PDF, encrypt it and upload it to the destination S3 bucket
        if key.lower().endswith('.pdf'):
            s3_client.download_file(bucket, key, download_path)
            encrypt_pdf(download_path, upload_path)
            encrypted_key = add_encrypted_suffix(key)
            s3_client.upload_file(upload_path, f'{bucket}-encrypted', encrypted_key)

# Define the function to encrypt the PDF file with a password
def encrypt_pdf(file_path, encrypted_file_path):
    reader = PdfReader(file_path)
    writer = PdfWriter()

    for page in reader.pages:
        writer.add_page(page)

    # Add a password to the new PDF
    writer.encrypt("my-secret-password")

    # Save the new PDF to a file
    with open(encrypted_file_path, "wb") as file:
        writer.write(file)

# Define a function to add a suffix to the original filename after encryption
def add_encrypted_suffix(original_key):
    filename, extension = original_key.rsplit('.', 1)
    return f'{filename}_encrypted.{extension}'

```

### Note

In questo codice di esempio, una password per il file crittografato (my-secret-password) è inserita nel codice della funzione. In un'applicazione di produzione, non includere informazioni

sensibili come le password nel codice funzione. Invece, [crea un AWS Secrets Manager segreto](#) e poi [usa l'estensione Lambda AWS Parameters and Secrets per recuperare le tue credenziali nella tua funzione Lambda](#).

Il codice della funzione python contiene tre funzioni: la [funzione handler](#) che Lambda esegue quando la funzione viene richiamata e due funzioni separate denominate `add_encrypted_suffix` e `encrypt_pdf` che l'handler chiama per eseguire la crittografia del PDF.

Quando la funzione viene richiamata da Amazon S3, Lambda passa un argomento di evento in formato JSON alla funzione che contiene dettagli sull'evento che ha causato l'invocazione. In questo caso, le informazioni includono il nome del bucket S3 e le chiavi oggetto per i file caricati. Per maggiori informazioni sul formato dell'oggetto evento per Amazon S3, consulta [the section called "S3"](#).

La tua funzione utilizza quindi AWS SDK per Python (Boto3) per scaricare i file PDF specificati nell'oggetto evento nella sua directory di archiviazione temporanea locale, prima di crittografarli utilizzando la libreria pypdf. <https://pypi.org/project/pypdf/>

Infine, la funzione utilizza l'SDK Boto3 per archiviare il file crittografato nel bucket di destinazione S3.

### **file manifest requirements.txt**

Copia e incolla il codice seguente in un nuovo file `requirements.txt`.

```
boto3
pypdf
```

Per questo esempio, il codice della funzione ha solo due dipendenze che non fanno parte della libreria Python standard: l'SDK per Python (Boto3) e il pacchetto pypdf utilizzato dalla funzione per eseguire la crittografia PDF.

#### Note

Una versione dell'SDK for Python (Boto3) è inclusa come parte del runtime Lambda, quindi il codice può essere eseguito senza aggiungere Boto3 al pacchetto di implementazione della funzione. Tuttavia, per mantenere il pieno controllo delle tue dipendenze ed evitare possibili problemi di disallineamento della versione, la best practice per Python è includere tutte le

dipendenze della funzione nel pacchetto di implementazione della tua funzione. Per ulteriori informazioni, consulta [the section called “Dipendenze di runtime in Python”](#).

## Distribuisci l'app

Puoi creare e distribuire le risorse per questa app di esempio manualmente o utilizzando AWS SAM. In un ambiente di produzione, ti consigliamo di utilizzare uno strumento IaC, AWS SAM ad esempio, per distribuire in modo rapido e ripetibile intere applicazioni serverless senza utilizzare processi manuali.

### Implementare le risorse manualmente

Per distribuire la tua app manualmente:

- Creare bucket Amazon S3 di origine e di destinazione
- Creare una funzione Lambda che crittografa un file PDF e salva la versione crittografata in un bucket S3
- Configurare un trigger Lambda che richiama la tua funzione quando gli oggetti vengono caricati nel bucket di origine

Prima di iniziare, assicurati che [Python](#) sia installato sulla tua macchina di compilazione.

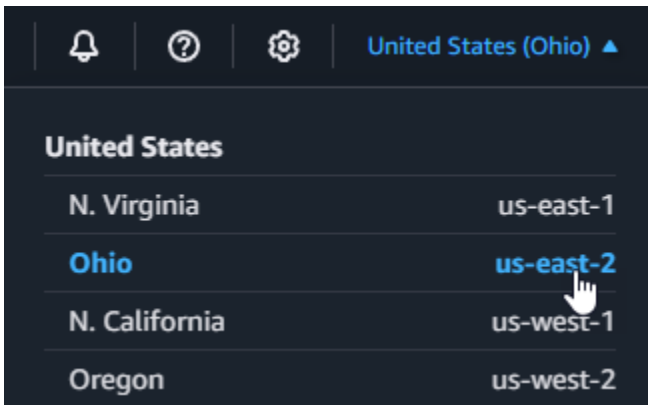
### Creare due bucket S3

Per prima cosa, crea due bucket S3. Il primo bucket è il bucket sorgente in cui carichi i tuoi file PDF. Il secondo bucket è utilizzato da Lambda per salvare il file crittografato quando richiami la tua funzione.

### Console

Per creare i bucket S3 (console)

1. Apri la [pagina](#) General Purpose buckets della console Amazon S3.
2. Seleziona quello più Regione AWS vicino alla tua posizione geografica. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo.



3. Scegliere Create bucket (Crea bucket).
4. In General configuration (Configurazione generale), eseguire le operazioni seguenti:
  - a. Per il tipo di secchio, assicurati che sia selezionato Uso generale.
  - b. Per Nome bucket, inserisci un nome univoco globale che soddisfi le regole di denominazione dei [bucket di Amazon S3](#). I nomi dei bucket possono contenere solo lettere minuscole, numeri, punti (.) e trattini (-).
5. Lascia tutte le altre opzioni impostate sui valori predefiniti e scegli Crea bucket.
6. Ripeti i passaggi da 1 a 4 per creare il bucket di destinazione. Per Nome del bucket, inserisci **amzn-s3-demo-bucket-encrypted**, dove **amzn-s3-demo-bucket** è il nome del bucket di origine che hai appena creato.

## AWS CLI

Prima di iniziare, assicurati che [AWS CLI sia installato](#) sulla tua macchina di compilazione.

Per creare i bucket Amazon S3 (AWS CLI)

1. Esegui il comando della CLI sotto riportato per creare il bucket di origine. Il nome scelto per il bucket deve essere univoco a livello globale e seguire le regole di denominazione dei bucket di Amazon S3. I nomi possono contenere solo lettere minuscole, numeri, punti (.) e trattini (-). Per region e LocationConstraint, scegli la [Regione AWS](#) più vicina alla tua posizione geografica.

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket --region us-east-2 \
--create-bucket-configuration LocationConstraint=us-east-2
```

Più avanti nel tutorial, devi creare la tua funzione Lambda nello stesso Regione AWS bucket di origine, quindi prendi nota della regione che hai scelto.

2. Esegui il comando sotto riportato per creare il bucket di destinazione. Per il nome del bucket, devi usare **amzn-s3-demo-bucket-encrypted**, dove **amzn-s3-demo-bucket** è il nome del bucket di origine che hai creato nel passaggio 1. Per `region eLocationConstraint`, scegli lo stesso Regione AWS che hai usato per creare il tuo bucket sorgente.

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket-encrypted --region us-east-2 \
--create-bucket-configuration LocationConstraint=us-east-2
```

## Creazione di un ruolo di esecuzione

Un ruolo di esecuzione è un ruolo IAM che concede a una funzione Lambda l'autorizzazione all'accesso Servizi AWS e alle risorse. Per concedere alla funzione l'accesso in lettura e scrittura ad Amazon S3, è necessario collegare la [policy gestita da AWS](#) `AmazonS3FullAccess`.

## Console

Per creare un ruolo di esecuzione e allegare la policy **AmazonS3FullAccess** gestita (console)

1. Aprire la pagina [Roles \(Ruoli\)](#) nella console IAM.
2. Scegli Crea ruolo.
3. Per il tipo di entità attendibile, seleziona AWS il servizio e, per Caso d'uso, seleziona Lambda.
4. Scegli Next (Successivo).
5. Aggiungi la policy `AmazonS3FullAccess` gestita procedendo come segue:
  - a. In Politiche di autorizzazione, **AmazonS3FullAccess** accedi alla barra di ricerca.
  - b. Seleziona la casella di controllo accanto alla policy.
  - c. Scegli Next (Successivo).
6. Nei dettagli del ruolo, immettere **LambdaS3Role** il nome del ruolo.
7. Selezionare Crea ruolo.

## AWS CLI

Per creare un ruolo di esecuzione e allegare la policy **AmazonS3FullAccess** gestita (AWS CLI)

1. Salva il seguente JSON in un file denominato `trust-policy.json`. Questa politica di fiducia consente a Lambda di utilizzare le autorizzazioni del ruolo concedendo al responsabile del servizio il `lambda.amazonaws.com` permesso di chiamare l'azione Servizio di token di sicurezza AWS (AWS STS) `AssumeRole`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Nella directory in cui hai salvato il documento della policy di attendibilità JSON, esegui il comando della CLI sotto riportato per creare il ruolo di esecuzione.

```
aws iam create-role --role-name LambdaS3Role --assume-role-policy-document
file://trust-policy.json
```

3. Per collegare la policy gestita da **AmazonS3FullAccess**, esegui il comando della CLI seguente.

```
aws iam attach-role-policy --role-name LambdaS3Role --policy-arn
arn:aws:iam::aws:policy/AmazonS3FullAccess
```

## Creazione del pacchetto di implementazione della funzione

Per creare la funzione, occorre creare un pacchetto di implementazione contenente la funzione e le rispettive dipendenze. Per questa applicazione, il codice della funzione utilizza una libreria separata per la crittografia dei PDF.

## Per creare il pacchetto di implementazione

1. Passa alla directory del progetto contenente i `lambda_function.py` `requirements.txt` file che hai creato o scaricato in GitHub precedenza e crea una nuova directory denominata `package`
2. Installa le dipendenze specificate nel file `requirements.txt` nella tua directory `package` eseguendo il comando seguente.

```
pip install -r requirements.txt --target ./package/
```

3. Crea un file `.zip` contenente il codice dell'applicazione e le relative dipendenze. Su Linux o MacOS, esegui i comandi riportati di seguito dall'interfaccia della linea di comando.

```
cd package
zip -r ../lambda_function.zip .
cd ..
zip lambda_function.zip lambda_function.py
```

Su Windows, usa il tuo strumento di compressione preferito per creare il file `lambda_function.zip`. Assicurati che il tuo file `lambda_function.py` e le cartelle contenenti le tue dipendenze si trovino tutti nella directory principale del file `.zip`.

Puoi creare il tuo pacchetto di implementazione anche utilizzando un ambiente virtuale Python. Per informazioni, consultare [Utilizzo di archivi di file .zip per le funzioni Lambda in Python](#).

## Creazione della funzione Lambda

Ora usi il pacchetto di implementazione creato nel passaggio precedente per implementare la tua funzione Lambda.

## Console

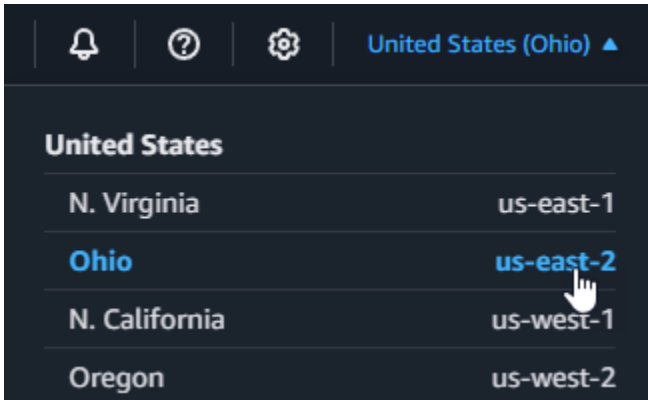
### Creazione della funzione (console)

Per creare la tua funzione Lambda usando la console, devi prima creare una funzione di base contenente del codice «Hello world». Quindi, sostituisci questo codice con il codice della tua funzione caricando il file `.zip` creato nel passaggio precedente.

Per garantire che la funzione non scada durante la crittografia di file PDF di grandi dimensioni, configura le impostazioni di memoria e timeout della funzione. Puoi anche impostare il formato di

log della funzione su JSON. La configurazione dei log in formato JSON è necessaria quando si utilizza lo script di test fornito, in modo che possa leggere lo stato di richiamo della funzione da Logs per confermare l'avvenuta invocazione. CloudWatch

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Assicurati di lavorare nello stesso Regione AWS ambiente in cui hai creato il bucket S3. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo.



3. Selezionare Create function (Crea funzione).
4. Scegli Author from scratch (Crea da zero).
5. In Basic information (Informazioni di base) eseguire queste operazioni:
  - a. Nel campo Function name (Nome funzione), immettere **EncryptPDF**.
  - b. In Runtime, scegli Python 3.12.
  - c. In Architecture (Architettura), scegli x86\_64.
6. Allega il ruolo di esecuzione che hai creato nel passaggio precedente procedendo come segue:
  - a. Espandi la sezione Change default execution role (Cambia ruolo di esecuzione predefinito).
  - b. Seleziona Utilizza un ruolo esistente.
  - c. In Ruolo esistente, seleziona il tuo ruolo (LambdaS3Role).
7. Scegli Crea funzione.

Caricamento del codice della funzione (console)

1. Nel riquadro Origine del codice, scegli Carica da.



2. Scegli File .zip.
3. Scegli Carica.
4. Nel selettore di file, seleziona il tuo file .zip e scegli Apri.
5. Scegli Save (Salva).

Per configurare la memoria e il timeout della funzione (console)

1. Seleziona la scheda Configurazione per la tua funzione.
2. Nel riquadro Configurazione generale, scegli Modifica.
3. Imposta Memoria su 256 MB e Timeout su 15 secondi.
4. Scegli Save (Salva).

Per configurare il formato di log (console)

1. Seleziona la scheda Configurazione per la tua funzione.
2. Seleziona Strumenti di monitoraggio e operazioni.
3. Nel riquadro Configurazione della registrazione, scegli Modifica.
4. Per Configurazione della registrazione, seleziona JSON.
5. Scegli Save (Salva).

## AWS CLI

Per creare la funzione (AWS CLI)

- Esegui il comando seguente dalla directory contenente il file `lambda_function.zip`. Per il parametro `region`, sostituisci `us-east-2` con la regione in cui hai creato i bucket S3.

```
aws lambda create-function --function-name EncryptPDF \
--zip-file fileb://lambda_function.zip --handler lambda_function.lambda_handler \
--runtime python3.12 --timeout 15 --memory-size 256 \
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-east-2 \
--logging-config LogFormat=JSON
```

## Configurare un trigger Amazon S3 per richiamare la funzione

Affinché la funzione Lambda venga eseguita quando carichi un file nel bucket di origine, devi configurare un trigger per la funzione. È possibile configurare il trigger Amazon S3 utilizzando la console Lambda o la AWS CLI.

### Important

Questa procedura configura il bucket S3 per richiamare la funzione ogni volta che un oggetto viene creato nel bucket. Assicurati di configurare questa opzione solo sul bucket di origine. Se la tua funzione Lambda crea oggetti nello stesso bucket che la richiama, la tua funzione può essere [richiamata continuamente in un ciclo ricorsivo \(loop\)](#). Ciò può comportare la fatturazione di addebiti imprevisti al tuo Account AWS.

## Console

### Configurazione del trigger Amazon S3 (console)

1. Apri la pagina [Funzioni](#) della console Lambda e scegli la tua funzione (EncryptPDF).
2. Selezionare Add trigger (Aggiungi trigger).
3. Seleziona S3.
4. In Bucket, seleziona il tuo bucket di origine.
5. In Tipi di eventi, seleziona Tutti gli eventi di creazione di oggetti.
6. In Invocazione ricorsiva, seleziona la casella di controllo per confermare che non è consigliabile utilizzare lo stesso bucket S3 per input e output. Per maggiori informazioni sui modelli di invocazione ricorsivi in Lambda, consulta [Schemi ricorsivi che causano loop indeterminati delle funzioni Lambda](#) in Serverless Land.
7. Scegliere Aggiungi.

Quando crei un trigger utilizzando la console Lambda, Lambda crea automaticamente una [policy basata sulle risorse](#) per concedere al servizio selezionato l'autorizzazione a richiamare la funzione.

## AWS CLI

Per configurare il trigger di Amazon S3 (AWS CLI)

1. Aggiungi una policy basata sulle [risorse](#) alla tua funzione che consenta al bucket di origine di Amazon S3 di richiamare la tua funzione quando aggiungi un file. Una dichiarazione di policy basata sulle risorse fornisce altre Servizi AWS autorizzazioni per invocare la tua funzione. Per autorizzare Amazon S3 a richiamare la tua funzione, esegui il comando della CLI comando sotto riportato. Assicurati di sostituire il `source-account` parametro con il tuo Account AWS ID e di utilizzare il tuo nome di bucket di origine.

```
aws lambda add-permission --function-name EncryptPDF \
--principal s3.amazonaws.com --statement-id s3invoke --action
"lambda:InvokeFunction" \
--source-arn arn:aws:s3:::amzn-s3-demo-bucket \
--source-account 123456789012
```

La policy che definisci con questo comando consente ad Amazon S3 di richiamare la tua funzione solo quando viene eseguita un'operazione sul tuo bucket di origine.

### Note

Sebbene i nomi dei bucket S3 siano univoci a livello globale, quando utilizzi policy basate sulle risorse è consigliabile specificare che il bucket deve appartenere al tuo account. Questo perché se elimini un bucket, è possibile che un altro ne Account AWS crei uno con lo stesso Amazon Resource Name (ARN).

2. Salva il seguente JSON in un file denominato `notification.json`. Quando viene applicato al tuo bucket di origine, questo JSON configura il bucket in modo che invii una notifica alla funzione Lambda ogni volta che viene aggiunto un nuovo oggetto. Sostituisci il Account AWS numero e Regione AWS nella funzione Lambda ARN con il tuo numero di account e la tua regione.

```
{
  "LambdaFunctionConfigurations": [
    {
      "Id": "EncryptPDFEventConfiguration",
      "LambdaFunctionArn": "arn:aws:lambda:us-
east-2:123456789012:function:EncryptPDF",

```

```

    "Events": [ "s3:ObjectCreated:Put" ]
  }
]
}

```

3. Esegui il comando della CLI comando sotto riportato per applicare le impostazioni di notifica nel file JSON che hai creato al tuo bucket di origine. Sostituisci `amzn-s3-demo-bucket` con il nome del tuo bucket di origine.

```

aws s3api put-bucket-notification-configuration --bucket amzn-s3-demo-bucket \
--notification-configuration file://notification.json

```

Per ulteriori informazioni sul comando `put-bucket-notification-configuration` e sull'opzione `notification-configuration`, consulta [put-bucket-notification-configuration](#) in Riferimento ai comandi di AWS CLI.

Distribuisci le risorse utilizzando AWS SAM

Prima di iniziare, assicurati che [Docker](#) e [la versione più recente AWS SAMCLI](#) siano installati sulla macchina di compilazione.

1. Nella directory del progetto, copia e incolla il codice seguente in un file denominato `template.yaml`. Sostituisci i nomi dei bucket segnaposto:
  - Per il bucket sorgente, sostituiscilo `amzn-s3-demo-bucket` con qualsiasi nome conforme alle regole di denominazione dei [bucket S3](#).
  - Per il bucket di destinazione, sostituisci `amzn-s3-demo-bucket-encrypted` con `<source-bucket-name>-encrypted`, dove `<source-bucket>` è il nome che hai scelto per il bucket di origine.

```

AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Resources:
  EncryptPDFFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: EncryptPDF
      Architectures: [x86_64]

```

```
CodeUri: ./
Handler: lambda_function.lambda_handler
Runtime: python3.12
Timeout: 15
MemorySize: 256
LoggingConfig:
  LogFormat: JSON
Policies:
  - AmazonS3FullAccess
Events:
  S3Event:
    Type: S3
    Properties:
      Bucket: !Ref PDFSourceBucket
      Events: s3:ObjectCreated:*

PDFSourceBucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketName: amzn-s3-demo-bucket

EncryptedPDFBucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketName: amzn-s3-demo-bucket-encrypted
```

Il AWS SAM modello definisce le risorse che crei per la tua app. In questo esempio, il modello definisce una funzione Lambda utilizzando il tipo `AWS::Serverless::Function` e due bucket S3 che utilizzano il tipo `AWS::S3::Bucket`. I nomi dei bucket specificati nel modello sono segnaposti. Prima di distribuire l'app utilizzando AWS SAM, è necessario modificare il modello per rinominare i bucket con nomi univoci a livello globale che soddisfino le regole di denominazione dei bucket [S3](#). Questo passaggio viene spiegato ulteriormente in [the section called “Distribuisci le risorse utilizzando AWS SAM”](#).

La definizione della risorsa della funzione Lambda configura un trigger per la funzione utilizzando la proprietà dell'evento `S3Event`. Questo trigger fa sì che la funzione venga richiamata ogni volta che viene creato un oggetto nel bucket di origine.

La definizione della funzione specifica anche una policy AWS Identity and Access Management (IAM) da allegare al ruolo di esecuzione della funzione. ??? La [policy gestita da AWS](#)

AmazonS3FullAccess fornisce alla tua funzione le autorizzazioni necessarie per leggere e scrivere oggetti su Amazon S3.

2. Eseguire il comando seguente nella directory in cui sono stati salvati i file `template.yaml`, `lambda_function.py` e `requirements.txt`.

```
sam build --use-container
```

Questo comando raccoglie gli artefatti di compilazione per l'applicazione e li colloca nel formato e nella posizione corretti per l'implementazione. Specificando l'opzione `--use-container` si crea la funzione all'interno di un Lambda-like contenitore Docker. Lo usiamo qui quindi non è necessario che Python 3.12 sia installato sul computer locale per far funzionare la build.

Durante il processo di compilazione, AWS SAM cerca il codice della funzione Lambda nella posizione specificata con la `CodeUri` proprietà nel modello. In questo caso, abbiamo specificato la directory corrente come posizione (`.`).

Se è presente un `requirements.txt` file, lo AWS SAM utilizza per raccogliere le dipendenze specificate. Per impostazione predefinita, AWS SAM crea un pacchetto di distribuzione .zip con il codice della funzione e le dipendenze. Puoi anche scegliere di distribuire la tua funzione come immagine del contenitore utilizzando la proprietà. [PackageType](#)

3. Per distribuire l'applicazione e creare le risorse Lambda e Amazon S3 specificate nel AWS SAM modello, esegui il comando seguente.

```
sam deploy --guided
```

L'uso del `--guided` flag significa che ti AWS SAM verranno mostrate istruzioni per guidarti nel processo di distribuzione. Per questa implementazione, accetta le opzioni predefinite premendo Invio.

Durante il processo di distribuzione, AWS SAM crea le seguenti risorse nel tuo Account AWS

- Una CloudFormation [pila](#) denominata `sam-app`
  - Una funzione Lambda con il nome `EncryptPDF`
  - Due bucket S3 con i nomi che hai scelto quando hai modificato il file modello `template.yaml`
- AWS SAM

- Un ruolo di esecuzione IAM per la tua funzione con il formato del nome `sam-app-EncryptPDFFunctionRole-2qGaapHFW0Q8`

Al AWS SAM termine della creazione delle risorse, dovresti visualizzare il seguente messaggio:

```
Successfully created/updated stack - sam-app in us-east-2
```

## Esecuzione del test dell'app

Per testare la tua app, carica un file PDF nel bucket di origine e conferma che Lambda crei una versione crittografata del file nel bucket di destinazione. In questo esempio, puoi testarlo manualmente utilizzando la console o il AWS CLI, oppure utilizzando lo script di test fornito.

Per le applicazioni di produzione, puoi utilizzare metodi e tecniche di test tradizionali, come il test di unità, per confermare il corretto funzionamento del codice di funzione Lambda. La best practice consiste anche nell'eseguire test come quelli dello script di test fornito, che eseguono test di integrazione con risorse reali basate sul cloud. I test di integrazione nel cloud confermano che l'infrastruttura è stata implementata correttamente e che gli eventi fluiscono tra i diversi servizi come previsto. Per ulteriori informazioni, consulta [the section called “Test in corso”](#).

### Test manuale dell'app

Puoi testare la tua funzione manualmente aggiungendo un file PDF al bucket di origine Amazon S3. Quando aggiungi un file al bucket di origine, la tua funzione Lambda dovrebbe essere richiamata automaticamente e dovrebbe memorizzare una versione crittografata del file nel bucket di destinazione.

### Console

Per testare l'app caricando un file (console)

1. Per caricare un file PDF nel bucket S3, procedi come segue:
  - a. Apri la pagina [Bucket](#) della console Amazon S3 e scegli il bucket di origine.
  - b. Scegli Carica.
  - c. Scegli Aggiungi file e utilizza il selettore di file per scegliere il file PDF da caricare.
  - d. Seleziona Apri, quindi Carica.
2. Verifica che Lambda abbia salvato una versione crittografata del tuo file PDF nel bucket di destinazione effettuando le seguenti operazioni:

- a. Torna alla pagina [Bucket](#) della console Amazon S3 e scegli il bucket di destinazione.
- b. Nel riquadro Oggetti, ora dovresti vedere un file con il formato del nome `filename_encrypted.pdf` (dove `filename.pdf` era il nome del file che hai caricato nel bucket di origine). Per scaricare il PDF crittografato, seleziona il file, quindi scegli Scarica.
- c. Conferma di poter aprire il file scaricato con la password con cui la funzione Lambda lo ha protetto (`my-secret-password`).

## AWS CLI

Per testare la tua app caricando un file (AWS CLI)

1. Dalla directory contenente il file PDF che desideri caricare, esegui il comando della CLI sotto riportato. Sostituisci il parametro `--bucket` con il nome del bucket di origine. Per i parametri `--key` e `--body`, usa il nome del file di test.

```
aws s3api put-object --bucket amzn-s3-demo-bucket --key test.pdf --body ./  
test.pdf
```

2. Verifica che la funzione abbia creato una versione crittografata del file e l'abbia salvata nel bucket S3 di destinazione. Esegui il comando della CLI sotto riportato sostituendo `amzn-s3-demo-bucket-encrypted` con il nome del tuo bucket di destinazione.

```
aws s3api list-objects-v2 --bucket amzn-s3-demo-bucket-encrypted
```

Se la funzione viene eseguita correttamente, vedrai un risultato simile al seguente. Il bucket di destinazione deve contenere un file con il formato del nome `<your_test_file>_encrypted.pdf`, dove `<your_test_file>` dov'è il nome del file che hai caricato.

```
{  
  "Contents": [  
    {  
      "Key": "test_encrypted.pdf",  
      "LastModified": "2023-06-07T00:15:50+00:00",  
      "ETag": "\"7781a43e765a8301713f533d70968a1e\"",  
      "Size": 2763,  
      "StorageClass": "STANDARD"    }  
  ]  
}
```



```
    }  
  ]  
}
```

3. Per scaricare il file salvato da Lambda nel bucket di destinazione, esegui il comando della CLI sotto riportato. Sostituire il parametro `--bucket` con il nome del tuo bucket di destinazione. Per il parametro `--key`, usa il nome del file `<your_test_file>_encrypted.pdf`, dove `<your_test_file>` è il nome del file di test che hai caricato.

```
aws s3api get-object --bucket amzn-s3-demo-bucket-encrypted --  
key test_encrypted.pdf my_encrypted_file.pdf
```

Questo comando scarica il file nella directory corrente e lo salva come `my_encrypted_file.pdf`.

4. Conferma di poter aprire il file scaricato con la password con cui la funzione Lambda lo ha protetto (`my-secret-password`).

## Test dell'app con lo script automatico

Create i seguenti file nella cartella del progetto:

- `test_pdf_encrypt.py`: uno script di test che puoi utilizzare per testare automaticamente l'applicazione
- `pytest.ini`: un file di configurazione per lo script di test

Espandi le seguenti sezioni per visualizzare il codice e per saperne di più sul ruolo di ciascun file.

## Script di test automatizzato

Copia e incolla il codice seguente in un nuovo file `test_pdf_encrypt.py`. Assicurati di sostituire i nomi dei bucket segnaposto:

- Nella funzione `test_source_bucket_available`, sostituisci `amzn-s3-demo-bucket` con il nome del tuo bucket di origine.
- Nella funzione `test_encrypted_file_in_bucket`, sostituisci `amzn-s3-demo-bucket-encrypted` con `source-bucket-encrypted`, dove `source-bucket` è il nome del tuo bucket di origine.

- Nella cleanup funzione, sostituisci `amzn-s3-demo-bucket` con il nome del bucket di origine e sostituisci `amzn-s3-demo-bucket-encrypted` con il nome del bucket di destinazione.

```
import boto3
import json
import pytest
import time
import os

@pytest.fixture
def lambda_client():
    return boto3.client('lambda')

@pytest.fixture
def s3_client():
    return boto3.client('s3')

@pytest.fixture
def logs_client():
    return boto3.client('logs')

@pytest.fixture(scope='session')
def cleanup():
    # Create a new S3 client for cleanup
    s3_client = boto3.client('s3')

    yield

    # Cleanup code will be executed after all tests have finished

    # Delete test.pdf from the source bucket
    source_bucket = 'amzn-s3-demo-bucket'
    source_file_key = 'test.pdf'
    s3_client.delete_object(Bucket=source_bucket, Key=source_file_key)
    print(f"\nDeleted {source_file_key} from {source_bucket}")

    # Delete test_encrypted.pdf from the destination bucket
    destination_bucket = 'amzn-s3-demo-bucket-encrypted'
    destination_file_key = 'test_encrypted.pdf'
    s3_client.delete_object(Bucket=destination_bucket, Key=destination_file_key)
    print(f"Deleted {destination_file_key} from {destination_bucket}")
```

```
@pytest.mark.order(1)
def test_source_bucket_available(s3_client):
    s3_bucket_name = 'amzn-s3-demo-bucket'
    file_name = 'test.pdf'
    file_path = os.path.join(os.path.dirname(__file__), file_name)

    file_uploaded = False
    try:
        s3_client.upload_file(file_path, s3_bucket_name, file_name)
        file_uploaded = True
    except:
        print("Error: couldn't upload file")

    assert file_uploaded, "Could not upload file to S3 bucket"

@pytest.mark.order(2)
def test_lambda_invoked(logs_client):

    # Wait for a few seconds to make sure the logs are available
    time.sleep(5)

    # Get the latest log stream for the specified log group
    log_streams = logs_client.describe_log_streams(
        logGroupName='/aws/lambda/EncryptPDF',
        orderBy='LastEventTime',
        descending=True,
        limit=1
    )

    latest_log_stream_name = log_streams['logStreams'][0]['logStreamName']

    # Retrieve the log events from the latest log stream
    log_events = logs_client.get_log_events(
        logGroupName='/aws/lambda/EncryptPDF',
        logStreamName=latest_log_stream_name
    )

    success_found = False
    for event in log_events['events']:
        message = json.loads(event['message'])
        status = message.get('record', {}).get('status')
        if status == 'success':
```

```
        success_found = True
        break

    assert success_found, "Lambda function execution did not report 'success' status in logs."

@pytest.mark.order(3)
def test_encrypted_file_in_bucket(s3_client):
    # Specify the destination S3 bucket and the expected converted file key
    destination_bucket = 'amzn-s3-demo-bucket-encrypted'
    converted_file_key = 'test_encrypted.pdf'

    try:
        # Attempt to retrieve the metadata of the converted file from the destination
        # S3 bucket
        s3_client.head_object(Bucket=destination_bucket, Key=converted_file_key)
    except s3_client.exceptions.ClientError as e:
        # If the file is not found, the test will fail
        pytest.fail(f"Converted file '{converted_file_key}' not found in the
        destination bucket: {str(e)}")

def test_cleanup(cleanup):
    # This test uses the cleanup fixture and will be executed last
    pass
```

Lo script di test automatizzato esegue tre funzioni di test per confermare il corretto funzionamento dell'app:

- Il test `test_source_bucket_available` conferma che il bucket di origine è stato creato correttamente caricando un file PDF di prova nel bucket.
- Il test `test_lambda_invoked` interroga il flusso di CloudWatch log di Logs più recente per la funzione per confermare che quando hai caricato il file di test, la funzione Lambda è stata eseguita e ha riportato un esito positivo.
- Il test `test_encrypted_file_in_bucket` conferma che il bucket di destinazione contiene il file crittografato `test_encrypted.pdf`.

Dopo l'esecuzione di tutti questi test, lo script esegue un ulteriore passaggio di pulizia per eliminare i file `test.pdf` e `test_encrypted.pdf` dai bucket di origine e di destinazione.

Come per il AWS SAM modello, i nomi dei bucket specificati in questo file sono segnaposto. Prima di eseguire il test, devi modificare questo file con i nomi reali dei bucket dell'app. Questo passaggio viene spiegato ulteriormente in [the section called “Test dell'app con lo script automatico”](#).

File di configurazione dello script di test

Copia e incolla il codice seguente in un nuovo file `pytest.ini`.

```
[pytest]
markers =
    order: specify test execution order
```

È necessario per specificare l'ordine in cui vengono eseguiti i test nello script `test_pdf_encrypt.py`.

Per eseguire i test, procedere come segue:

1. Assicuratevi che il `pytest` modulo sia installato nel vostro ambiente locale. È possibile installare `pytest` eseguendo il comando seguente:

```
pip install pytest
```

2. Salvate un file PDF denominato `test.pdf` nella directory contenente i `pytest.ini` file `test_pdf_encrypt.py` e.
3. Apri un programma di terminale o di shell ed esegui il comando sotto riportato dalla directory contenente i file di test.

```
pytest -s -v
```

Una volta completato il test, l'output dovrebbe essere simile al seguente:

```
===== test session starts
=====
platform linux -- Python 3.12.2, pytest-7.2.2, pluggy-1.0.0 -- /usr/bin/python3
cachedir: .pytest_cache
hypothesis profile 'default' -> database=DirectoryBasedExampleDatabase('/home/
pdf_encrypt_app/.hypothesis/examples')
Test order randomisation NOT enabled. Enable with --random-order or --random-order-
bucket=<bucket_type>
```

```

rootdir: /home/pdf_encrypt_app, configfile: pytest.ini
plugins: anyio-3.7.1, hypothesis-6.70.0, localserver-0.7.1, random-order-1.1.0
collected 4 items

test_pdf_encrypt.py::test_source_bucket_available PASSED
test_pdf_encrypt.py::test_lambda_invoked PASSED
test_pdf_encrypt.py::test_encrypted_file_in_bucket PASSED
test_pdf_encrypt.py::test_cleanup PASSED
Deleted test.pdf from amzn-s3-demo-bucket
Deleted test_encrypted.pdf from amzn-s3-demo-bucket-encrypted

===== 4 passed in 7.32s
=====

```

## Fasi successive

Ora che hai creato questa app di esempio, puoi utilizzare il codice fornito come base per creare altri tipi di applicazioni per l'elaborazione di file. Modifica il codice nel file `lambda_function.py` per implementare la logica di elaborazione dei file per il tuo caso d'uso.

Numerosi casi d'uso tipici dell'elaborazione di file riguardano l'elaborazione delle immagini. Quando si usa Python, le librerie di elaborazione delle immagini più popolari come [pillow](#) contengono in genere componenti C o C++. Per garantire che il pacchetto di distribuzione della funzione sia compatibile con l'ambiente di esecuzione Lambda, è importante utilizzare il binario di distribuzione dei sorgenti corretto.

Quando distribuisce le tue risorse con AWS SAM, devi eseguire alcuni passaggi aggiuntivi per includere la giusta distribuzione dei sorgenti nel pacchetto di distribuzione. Poiché AWS SAM non installa le dipendenze per una piattaforma diversa da quella della macchina di compilazione, specificare la corretta distribuzione dei sorgenti (`.whl`file) nel `requirements.txt` file non funzionerà se la macchina di compilazione utilizza un sistema operativo o un'architettura diversi dall'ambiente di esecuzione Lambda. Invece, effettua una delle seguenti operazioni:

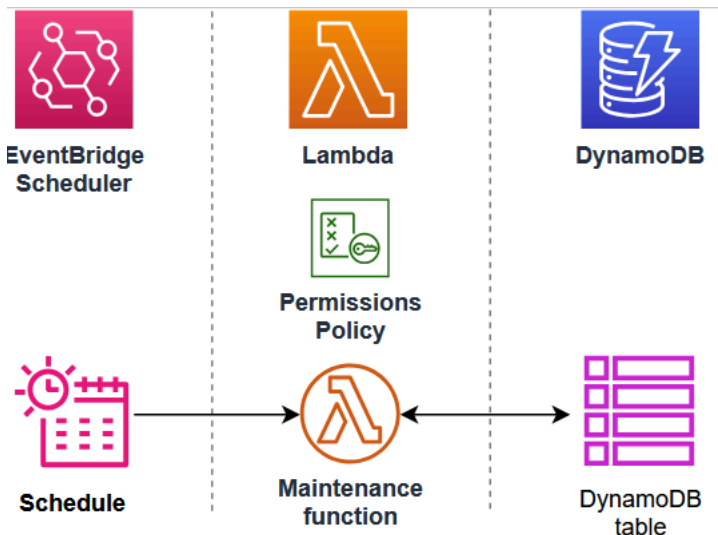
- Usa l'opzione `--use-container` durante l'esecuzione di `sam build`. Quando specifichi questa opzione, AWS SAM scarica un'immagine di base del contenitore compatibile con l'ambiente di esecuzione Lambda e crea il pacchetto di distribuzione della funzione in un contenitore Docker utilizzando quell'immagine. Per ulteriori informazioni, consulta [Creazione di una funzione Lambda all'interno di un container fornito](#).

- Crea tu stesso il pacchetto di distribuzione .zip della tua funzione utilizzando il binario di distribuzione dei sorgenti corretto e salva il file .zip nella directory specificata come nel modello. CodeUri AWS SAM Per ulteriori informazioni sulla creazione di pacchetti di implementazione .zip per Python utilizzando distribuzioni binarie, consulta [the section called “Creazione di un pacchetto di implementazione .zip con dipendenze”](#) e [the section called “Creazione di un pacchetto di implementazione .zip con librerie native”](#).

## Creare un'app per eseguire la manutenzione pianificata del database

È possibile utilizzarla AWS Lambda per sostituire processi pianificati come backup automatici del sistema, conversioni di file e attività di manutenzione. In questo esempio, viene creata un'applicazione serverless che esegue una manutenzione programmata regolare su una tabella DynamoDB eliminando le voci precedenti. L'app utilizza EventBridge Scheduler per richiamare una funzione Lambda su una pianificazione cron. Quando viene richiamata, la funzione interroga la tabella per individuare gli elementi più vecchi di un anno e li elimina. La funzione registra ogni elemento eliminato in Logs. CloudWatch

Per implementare questo esempio, crea prima una tabella DynamoDB e completala con alcuni dati di test per la funzione da interrogare. Quindi, crea una funzione Python Lambda con un trigger EventBridge Scheduler e un ruolo di esecuzione IAM che dia alla funzione il permesso di leggere ed eliminare gli elementi dalla tabella.



 Tip

Se non conosci Lambda, ti consigliamo di completare il tutorial [the section called “Crea la tua prima funzione”](#) prima di creare questa app di esempio.

Puoi distribuire la tua app manualmente creando e configurando le risorse con. Console di gestione AWS. Puoi anche distribuire l'app utilizzando (). AWS Serverless Application Model AWS SAM AWS SAM è uno strumento Infrastructure as code (IaC). Con IaC, non crei risorse manualmente, ma le definisci in codice e poi le distribuisce automaticamente.

Se desideri saperne di più sull'utilizzo di Lambda con IaC prima di implementare questa app di esempio, consulta [the section called “Infrastructure as code \(IaC\)”](#).

## Prerequisiti

Prima di creare l'app di esempio, assicurati di aver installato gli strumenti e i programmi da riga di comando necessari.

- Python

Per popolare la tabella DynamoDB creata per testare l'app, questo esempio utilizza uno script Python e un file CSV per scrivere i dati nella tabella. Assicurati che Python versione 3.8 o successiva sia installato sulla tua macchina.

- AWS SAM CLI

Se desideri creare la tabella DynamoDB e distribuire l'app di esempio utilizzando AWS SAM, devi installare la CLI. AWS SAM Segui le [istruzioni di installazione](#) nella Guida per l'utente di AWS SAM .

- AWS CLI

Per utilizzare lo script Python fornito per popolare la tabella di test, è necessario aver installato e configurato il AWS CLI. Questo perché lo script utilizza il AWS SDK per Python (Boto3), che richiede l'accesso alle tue credenziali AWS Identity and Access Management (IAM). È inoltre necessario che AWS CLI sia installato per distribuire le risorse utilizzando. AWS SAM Installa la CLI seguendo le [istruzioni di installazione](#) riportate nella Guida per l'utente di AWS Command Line Interface .

- Docker



Per distribuire l'app utilizzando AWS SAM, Docker deve essere installato anche sulla macchina di compilazione. Segui le istruzioni in [Installa Docker Engine sul sito](#) Web della documentazione di Docker.

## Download dei file dell'app di esempio

Per creare il database di esempio e l'app di manutenzione programmata, è necessario creare i seguenti file nella directory del progetto:

### File di database di esempio

- `template.yaml`- un AWS SAM modello che puoi usare per creare la tabella DynamoDB
- `sample_data.csv`: un file CSV contenente dati di esempio da caricare nella tabella
- `load_sample_data.py`: uno script Python che scrive i dati del file CSV nella tabella

### Scheduled-maintenance file dell'app

- `lambda_function.py`: il codice della funzione Python per la funzione Lambda che esegue la manutenzione del database
- `requirements.txt`: un file manifesto che definisce le dipendenze richieste dal codice della funzione Python
- `template.yaml`- un AWS SAM modello che puoi usare per distribuire l'app

### File di test

- `test_app.py`: uno script Python che scansiona la tabella e conferma il corretto funzionamento della funzione emettendo tutti i record più vecchi di un anno

Espandi le seguenti sezioni per visualizzare il codice e per saperne di più sul ruolo di ciascun file nella creazione e nel test dell'app. Per creare i file sul tuo computer locale, copia e incolla il codice seguente.

### AWS SAM modello (esempio tabella DynamoDB)

Copia e incolla il codice seguente in un nuovo file `template.yaml`.

```
AWSTemplateFormatVersion: '2010-09-09'
```

Transform: AWS::Serverless-2016-10-31

Description: SAM Template for DynamoDB Table with Order\_number as Partition Key and Date as Sort Key

Resources:

MyDynamoDBTable:

Type: AWS::DynamoDB::Table

DeletionPolicy: Retain

UpdateReplacePolicy: Retain

Properties:

TableName: MyOrderTable

BillingMode: PAY\_PER\_REQUEST

AttributeDefinitions:

- AttributeName: Order\_number

AttributeType: S

- AttributeName: Date

AttributeType: S

KeySchema:

- AttributeName: Order\_number

KeyType: HASH

- AttributeName: Date

KeyType: RANGE

SSESpecification:

SSEEnabled: true

GlobalSecondaryIndexes:

- IndexName: Date-index

KeySchema:

- AttributeName: Date

KeyType: HASH

Projection:

ProjectionType: ALL

PointInTimeRecoverySpecification:

PointInTimeRecoveryEnabled: true

Outputs:

TableName:

Description: DynamoDB Table Name

Value: !Ref MyDynamoDBTable

TableArn:

Description: DynamoDB Table ARN

Value: !GetAtt MyDynamoDBTable.Arn

 Note

AWS SAM i modelli utilizzano una convenzione di denominazione standard di `template.yaml`. In questo esempio, hai due file modello: uno per creare il database di esempio e l'altro per creare l'app stessa. Salvali in sottodirectory separate nella cartella di progetto.

Questo AWS SAM modello definisce la risorsa di tabella DynamoDB che crei per testare la tua app. La tabella utilizza una chiave primaria di `Order_number` con una chiave di ordinamento di `Date`. Affinché la funzione Lambda trovi gli elementi direttamente in base alla data, definiamo anche un [indice secondario globale](#) denominato `Date-index`.

Per ulteriori informazioni sulla creazione e la configurazione di una tabella DynamoDB tramite la risorsa `AWS::DynamoDB::Table`, consulta [AWS::DynamoDB::Table](#) nella Guida per l'utente di AWS CloudFormation .

File di dati del database di esempio

Copia e incolla il codice seguente in un nuovo file `sample_data.csv`.

```
Date,Order_number,CustomerName,ProductID,Quantity,TotalAmount
2023-09-01,ORD001,Alejandro Rosalez,PROD123,2,199.98
2023-09-01,ORD002,Akua Mansa,PROD456,1,49.99
2023-09-02,ORD003,Ana Carolina Silva,PROD789,3,149.97
2023-09-03,ORD004,Arnav Desai,PROD123,1,99.99
2023-10-01,ORD005,Carlos Salazar,PROD456,2,99.98
2023-10-02,ORD006,Diego Ramirez,PROD789,1,49.99
2023-10-03,ORD007,Efua Owusu,PROD123,4,399.96
2023-10-04,ORD008,John Stiles,PROD456,2,99.98
2023-10-05,ORD009,Jorge Souza,PROD789,3,149.97
2023-10-06,ORD010,Kwaku Mensah,PROD123,1,99.99
2023-11-01,ORD011,Li Juan,PROD456,5,249.95
2023-11-02,ORD012,Marcia Oliveria,PROD789,2,99.98
2023-11-03,ORD013,Maria Garcia,PROD123,3,299.97
2023-11-04,ORD014,Martha Rivera,PROD456,1,49.99
2023-11-05,ORD015,Mary Major,PROD789,4,199.96
2023-12-01,ORD016,Mateo Jackson,PROD123,2,199.99
2023-12-02,ORD017,Nikki Wolf,PROD456,3,149.97
2023-12-03,ORD018,Pat Candella,PROD789,1,49.99
2023-12-04,ORD019,Paulo Santos,PROD123,5,499.95
```

```
2023-12-05,ORD020,Richard Roe,PROD456,2,99.98
2024-01-01,ORD021,Saanvi Sarkar,PROD789,3,149.97
2024-01-02,ORD022,Shirley Rodriguez,PROD123,1,99.99
2024-01-03,ORD023,Sofia Martinez,PROD456,4,199.96
2024-01-04,ORD024,Terry Whitlock,PROD789,2,99.98
2024-01-05,ORD025,Wang Xiulan,PROD123,3,299.97
```

Questo file contiene alcuni esempi di dati di test con cui compilare la tabella DynamoDB in formato CSV (valori separati da virgola) standard.

Script Python per caricare dati di esempio

Copia e incolla il codice seguente in un nuovo file `load_sample_data.py`.

```
import boto3
import csv
from decimal import Decimal

# Initialize the DynamoDB client
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('MyOrderTable')
print("DDB client initialized.")

def load_data_from_csv(filename):
    with open(filename, 'r') as file:
        csv_reader = csv.DictReader(file)
        for row in csv_reader:
            item = {
                'Order_number': row['Order_number'],
                'Date': row['Date'],
                'CustomerName': row['CustomerName'],
                'ProductID': row['ProductID'],
                'Quantity': int(row['Quantity']),
                'TotalAmount': Decimal(str(row['TotalAmount']))
            }
            table.put_item(Item=item)
            print(f"Added item: {item['Order_number']} - {item['Date']}")

if __name__ == "__main__":
    load_data_from_csv('sample_data.csv')
    print("Data loading completed.")
```

Questo script Python utilizza innanzitutto il AWS SDK per Python (Boto3) per creare una connessione alla tabella DynamoDB. Quindi esegue un'iterazione su ogni riga del file CSV di dati di esempio, crea un elemento da quella riga e scrive l'elemento nella tabella DynamoDB utilizzando l'SDK boto3.

## Codice della funzione Python

Copia e incolla il codice seguente in un nuovo file `lambda_function.py`.

```
import boto3
from datetime import datetime, timedelta
from boto3.dynamodb.conditions import Key, Attr
import logging

logger = logging.getLogger()
logger.setLevel("INFO")

def lambda_handler(event, context):
    # Initialize the DynamoDB client
    dynamodb = boto3.resource('dynamodb')

    # Specify the table name
    table_name = 'MyOrderTable'
    table = dynamodb.Table(table_name)

    # Get today's date
    today = datetime.now()

    # Calculate the date one year ago
    one_year_ago = (today - timedelta(days=365)).strftime('%Y-%m-%d')

    # Scan the table using a global secondary index
    response = table.scan(
        IndexName='Date-index',
        FilterExpression='#date < :one_year_ago',
        ExpressionAttributeNames={
            '#date': 'Date'
        },
        ExpressionAttributeValues={
            ':one_year_ago': one_year_ago
        }
    )

    # Delete old items
```

```
with table.batch_writer() as batch:
    for item in response['Items']:
        Order_number = item['Order_number']
        batch.delete_item(
            Key={
                'Order_number': Order_number,
                'Date': item['Date']
            }
        )
        logger.info(f'deleted order number {Order_number}')

# Check if there are more items to scan
while 'LastEvaluatedKey' in response:
    response = table.scan(
        IndexName='DateIndex',
        FilterExpression='#date < :one_year_ago',
        ExpressionAttributeNames={
            '#date': 'Date'
        },
        ExpressionAttributeValues={
            ':one_year_ago': one_year_ago
        },
        ExclusiveStartKey=response['LastEvaluatedKey']
    )

# Delete old items
with table.batch_writer() as batch:
    for item in response['Items']:
        batch.delete_item(
            Key={
                'Order_number': item['Order_number'],
                'Date': item['Date']
            }
        )

return {
    'statusCode': 200,
    'body': 'Cleanup completed successfully'
}
```

Il codice della funzione Python contiene la [funzione handler](#) (`lambda_handler`) che Lambda esegue quando viene invocata la funzione.

Quando la funzione viene richiamata da EventBridge Scheduler, utilizza il AWS SDK per Python (Boto3) per creare una connessione alla tabella DynamoDB su cui deve essere eseguita l'attività di manutenzione programmata. Quindi utilizza la libreria `datetime` Python per calcolare la data di un anno fa, prima di scansionare la tabella alla ricerca di elementi più vecchi di questo ed eliminarli.

Tieni presente che le risposte delle operazioni di query e scansione di DynamoDB sono limitate a una dimensione massima di 1 MB. Se la risposta è superiore a 1 MB, DynamoDB impagina i dati e restituisce un elemento `LastEvaluatedKey` nella risposta. Per garantire che la nostra funzione elabori tutti i record della tabella, controlliamo la presenza di questa chiave e continuiamo a eseguire scansioni della tabella dall'ultima posizione valutata fino alla scansione dell'intera tabella.

### **file manifest requirements.txt**

Copia e incolla il codice seguente in un nuovo file `requirements.txt`.

```
boto3
```

Per questo esempio, il codice della funzione ha solo una dipendenza che non fa parte della libreria Python standard: l'SDK for Python (Boto3) che la funzione utilizza per scansionare ed eliminare elementi dalla tabella DynamoDB.

#### Note

Una versione dell'SDK for Python (Boto3) è inclusa come parte del runtime Lambda, quindi il codice può essere eseguito senza aggiungere Boto3 al pacchetto di implementazione della funzione. Tuttavia, per mantenere il pieno controllo delle tue dipendenze ed evitare possibili problemi di disallineamento della versione, la best practice per Python è includere tutte le dipendenze della funzione nel pacchetto di implementazione della tua funzione. Per ulteriori informazioni, consulta [the section called “Dipendenze di runtime in Python”](#).

### AWS SAM modello (app per la manutenzione programmata)

Copia e incolla il codice seguente in un nuovo file `template.yaml`.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: SAM Template for Lambda function and EventBridge Scheduler rule

Resources:
```

```

MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    FunctionName: ScheduledDBMaintenance
    CodeUri: ./
    Handler: lambda_function.lambda_handler
    Runtime: python3.11
    Architectures:
      - x86_64
    Events:
      ScheduleEvent:
        Type: ScheduleV2
        Properties:
          ScheduleExpression: cron(0 3 1 * ? *)
          Description: Run on the first day of every month at 03:00 AM
    Policies:
      - CloudWatchLogsFullAccess
      - Statement:
          - Effect: Allow
            Action:
              - dynamodb:Scan
              - dynamodb:BatchWriteItem
            Resource: !Sub 'arn:aws:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
MyOrderTable'

LambdaLogGroup:
  Type: AWS::Logs::LogGroup
  Properties:
    LogGroupName: !Sub /aws/lambda/${MyLambdaFunction}
    RetentionInDays: 30

Outputs:
  LambdaFunctionName:
    Description: Lambda Function Name
    Value: !Ref MyLambdaFunction
  LambdaFunctionArn:
    Description: Lambda Function ARN
    Value: !GetAtt MyLambdaFunction.Arn

```

### Note

AWS SAM i modelli utilizzano una convenzione di denominazione standard di `template.yaml`. In questo esempio, hai due file modello: uno per creare il database di



esempio e l'altro per creare l'app stessa. Salvali in sottodirectory separate nella cartella di progetto.

Questo AWS SAM modello definisce le risorse per la tua app. Definiamo la funzione Lambda utilizzando la risorsa `AWS::Serverless::Function`. La EventBridge pianificazione dello Scheduler e il trigger per richiamare la funzione Lambda vengono creati utilizzando la `Events` proprietà di questa risorsa utilizzando un tipo di `ScheduleV2`. Per saperne di più sulla definizione delle EventBridge pianificazioni di Scheduler nei AWS SAM modelli, consulta [ScheduleV2 nella Developer Guide](#). AWS Serverless Application Model

Oltre alla funzione Lambda e alla EventBridge pianificazione Scheduler, definiamo anche un gruppo di CloudWatch log a cui la funzione può inviare i record degli elementi eliminati.

### Script di test

Copia e incolla il codice seguente in un nuovo file `test_app.py`.

```
import boto3
from datetime import datetime, timedelta
import json

# Initialize the DynamoDB client
dynamodb = boto3.resource('dynamodb')

# Specify your table name
table_name = 'YourTableName'
table = dynamodb.Table(table_name)

# Get the current date
current_date = datetime.now()

# Calculate the date one year ago
one_year_ago = current_date - timedelta(days=365)

# Convert the date to string format (assuming the date in DynamoDB is stored as a
string)
one_year_ago_str = one_year_ago.strftime('%Y-%m-%d')

# Scan the table
response = table.scan(
    FilterExpression='#date < :one_year_ago',
```

```

    ExpressionAttributeNames={
        '#date': 'Date'
    },
    ExpressionAttributeValues={
        ':one_year_ago': one_year_ago_str
    }
)

# Process the results
old_records = response['Items']

# Continue scanning if we have more items (pagination)
while 'LastEvaluatedKey' in response:
    response = table.scan(
        FilterExpression='#date < :one_year_ago',
        ExpressionAttributeNames={
            '#date': 'Date'
        },
        ExpressionAttributeValues={
            ':one_year_ago': one_year_ago_str
        },
        ExclusiveStartKey=response['LastEvaluatedKey']
    )
    old_records.extend(response['Items'])

for record in old_records:
    print(json.dumps(record))

# The total number of old records should be zero.
print(f"Total number of old records: {len(old_records)}")

```

Questo script di test utilizza il AWS SDK per Python (Boto3) per creare una connessione alla tabella DynamoDB e scansionare gli elementi più vecchi di un anno. Per confermare se la funzione Lambda è stata eseguita correttamente, alla fine del test, la funzione stampa il numero di record più vecchi di un anno ancora nella tabella. Se la funzione Lambda ha avuto successo, il numero di vecchi record nella tabella dovrebbe essere zero.

## Creazione e compilazione della tabella DynamoDB di esempio

Per testare la tua app di manutenzione programmata, devi prima creare una tabella DynamoDB e popolarla con alcuni dati di esempio. È possibile creare la tabella manualmente utilizzando o la

Console di gestione AWS o AWS SAM. Si consiglia di AWS SAM utilizzarlo per creare e configurare rapidamente la tabella utilizzando alcuni AWS CLI comandi.

## Console

### Creare la tabella DynamoDB

1. Aprire la pagina [Tables \(Tabelle\)](#) della console DynamoDB.
2. Scegliere Create table (Crea tabella).
3. Crea la tabella completando le operazioni seguenti:
  - a. In Dettagli tabella, per Nome tabella, immetti **MyOrderTable**.
  - b. Per Partition key (Chiave di partizione), immettere **Order\_number** e mantenere il tipo di dati impostato come String (Stringa).
  - c. Per Chiave di ordinamento, immetti **Date** e lascia il tipo come String.
  - d. Lascia Impostazioni della tabella impostato su Impostazioni predefinite e scegli Crea tabella.
4. Una volta completata la creazione della tabella e lo stato è impostato su Attivo, crea un indice secondario globale (GSI) procedendo come segue. L'app utilizza questo GSI per cercare gli elementi direttamente per data e determinare cosa eliminare.
  - a. Scegli MyOrderTable dall'elenco delle tabelle.
  - b. Scegli la scheda Indici.
  - c. In Indici secondari globali, scegli Crea indice.
  - d. In Dettagli dell'indice, inserisci **Date** per Chiave di partizione e lascia il campo Tipo di dati impostato su Stringa.
  - e. Per Index name (Nome indice), inserisci **Date-index**.
  - f. Lascia tutti gli altri parametri impostati sui valori predefiniti, scorri fino alla parte inferiore della pagina e scegli Crea indice.

## AWS SAM

### Creare la tabella DynamoDB

1. Passa alla cartella in cui è stato salvato il file `template.yaml` per la tabella DynamoDB.  
Nota che questo esempio utilizza due file `template.yaml`. Assicurati che siano salvati in

sottocartelle separate e di trovarti nella cartella corretta contenente il modello per creare la tua tabella DynamoDB.

2. Eseguire il seguente comando seguente.

```
sam build
```

Questo comando raccoglie gli artefatti di compilazione per l'applicazione e li colloca nel formato e nella posizione corretti per l'implementazione.

3. Per creare la risorsa DynamoDB specificata nel file `template.yaml`, eseguire il comando seguente.

```
sam deploy --guided
```

L'uso del `--guided` contrassegno significa che AWS SAM vengono visualizzate istruzioni che guidano l'utente nel processo di distribuzione. Per questa implementazione, inserisci un `Stack name` pari a **cron-app-test-db** e accetta i valori predefiniti per tutte le altre opzioni utilizzando Invio.

Quando hai AWS SAM finito di creare la risorsa DynamoDB, dovresti vedere il seguente messaggio.

```
Successfully created/updated stack - cron-app-test-db in us-west-2
```

4. È inoltre possibile confermare che la tabella DynamoDB è stata creata aprendo la pagina [Tabelle](#) della console DynamoDB. Dovresti vedere una tabella denominata `MyOrderTable`.

Dopo aver creato la tabella, aggiungi alcuni dati di esempio per testare l'app. Il file CSV `sample_data.csv` scaricato in precedenza contiene una serie di voci di esempio tra cui numeri d'ordine, date e informazioni su clienti e ordini. Usa lo script python `load_sample_data.py` fornito per aggiungere questi dati alla tua tabella.

Per aggiungere i dati di esempio alla tabella

1. Passa alla directory contenente i file `sample_data.csv` e `load_sample_data.py`. Se questi file si trovano in directory separate, spostali in modo che vengano salvati nella stessa posizione.

2. Crea un ambiente virtuale Python in cui eseguire lo script mediante il comando seguente. Ti consigliamo di utilizzare un ambiente virtuale perché nel passaggio successivo dovrai installare AWS SDK per Python (Boto3).

```
python -m venv venv
```

3. Nell'ambiente virtuale, esegui il comando seguente.

```
source venv/bin/activate
```

4. Installa l'SDK per Python (Boto3) nell'ambiente virtuale mediante il comando seguente. Lo script utilizza questa libreria per connettersi alla tabella DynamoDB e aggiungere gli elementi.

```
pip install boto3
```

5. Esegui lo script per popolare la tabella mediante il comando seguente.

```
python load_sample_data.py
```

Se lo script viene eseguito correttamente, dovrebbe stampare ogni elemento sulla console man mano che lo carica e riporta `Data loading completed`.

6. Per disattivare l'ambiente virtuale, esegui il comando seguente.

```
deactivate
```

7. È possibile verificare che i dati siano stati caricati nella tabella DynamoDB eseguendo le operazioni seguenti:
  - a. Apri la pagina [Esplora elementi](#) della console DynamoDB e seleziona la tua tabella (`MyOrderTable`).
  - b. Nel riquadro Elementi restituiti, dovresti vedere i 25 elementi del file CSV che lo script ha aggiunto alla tabella.

## Creazione dell'app per la manutenzione programmata

È possibile creare e distribuire le risorse per questa app di esempio passo dopo passo utilizzando Console di gestione AWS o utilizzando AWS SAM. In un ambiente di produzione, ti consigliamo

di utilizzare uno strumento Infrastructure-as-Code (IaC), AWS SAM ad esempio per distribuire ripetutamente applicazioni serverless senza utilizzare processi manuali.

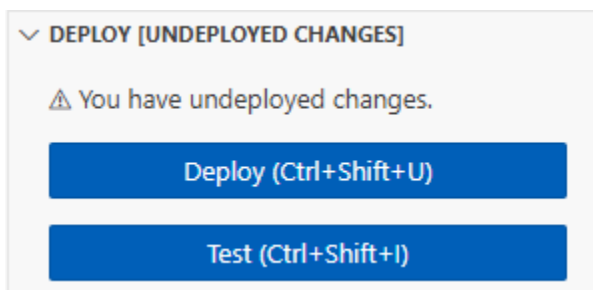
Per questo esempio, segui le istruzioni della console per imparare a configurare ogni AWS risorsa separatamente o segui AWS SAM le istruzioni per distribuire rapidamente l'app utilizzando i comandi AWS CLI

## Console

Per creare la funzione utilizzando Console di gestione AWS

Innanzitutto, crea una funzione contenente il codice di avvio di base. Quindi, sostituisci questo codice con il codice della tua funzione copiando e incollando il codice direttamente nell'editor di codice Lambda o caricandolo come pacchetto .zip. Per questa operazione, consigliamo di copiare e incollare il codice.

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Selezionare Create function (Crea funzione).
3. Scegli Author from scratch (Crea da zero).
4. In Basic information (Informazioni di base) eseguire queste operazioni:
  - a. Nel campo Function name (Nome funzione), immettere **ScheduledDBMaintenance**.
  - b. Per Runtime scegli l'ultima versione di Python.
  - c. In Architecture (Architettura), scegli x86\_64.
5. Scegli Crea funzione.
6. Dopo aver creato la funzione, è possibile configurarla con il codice di funzione fornito.
  - a. Nel riquadro Codice sorgente, sostituisci il codice Hello world creato da Lambda con il codice della funzione Python dal file `lambda_function.py` salvato in precedenza.
  - b. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Per configurare la memoria e il timeout della funzione (console)

1. Seleziona la scheda Configurazione per la tua funzione.
2. Nel riquadro Configurazione generale, scegli Modifica.
3. Imposta Memoria su 256 MB e Timeout su 15 secondi. Se stai elaborando una tabella di grandi dimensioni con molti record, ad esempio nel caso di un ambiente di produzione, potresti prendere in considerazione l'impostazione di Timeout su un numero maggiore. In questo modo la funzione ha più tempo per scansionare e pulire il database.
4. Scegli Save (Salva).

Per configurare il formato di log (console)

Puoi configurare le funzioni Lambda per generare log in formato testo non strutturato o JSON. È consigliabile utilizzare il formato JSON per i log per agevolare la ricerca e il filtraggio dei dati di log. Per ulteriori informazioni sulle opzioni di configurazione del log Lambda, consulta [the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”](#).

1. Seleziona la scheda Configurazione per la tua funzione.
2. Seleziona Strumenti di monitoraggio e operazioni.
3. Nel riquadro Configurazione della registrazione, scegli Modifica.
4. Per Configurazione della registrazione, seleziona JSON.
5. Scegli Save (Salva).

Impostazione delle autorizzazioni IAM

Per concedere alla funzione le autorizzazioni necessarie per leggere ed eliminare gli elementi di DynamoDB, è necessario aggiungere una policy al [ruolo di esecuzione](#) della funzione che definisca le autorizzazioni necessarie.

1. Apri la scheda Configurazione, quindi scegli Autorizzazioni dalla barra di navigazione a sinistra.
2. Scegli il nome del ruolo Ruolo di esecuzione.
3. Nella console IAM, scegli Aggiungi autorizzazioni, quindi Crea policy in linea.
4. Utilizza l'editor JSON e inserisci la seguente policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:Scan",
        "dynamodb>DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/MyOrderTable"
    }
  ]
}
```

5. Assegna un nome alla policy **DynamoDBCleanupPolicy**, quindi creala.

Per configurare EventBridge Scheduler come trigger (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la tua funzione (ScheduledDBMaintenance).
3. Nel riquadro Panoramica della funzione, scegli Aggiungi trigger.
4. Dal menu a discesa della fonte del trigger, scegli Scheduler.
5. Configura le impostazioni del VPC effettuando le seguenti operazioni:
  - a. Per il nome della pianificazione, inserisci. **DynamoDBCleanupSchedule**
  - b. Per il gruppo Schedule, lascia il valore predefinito (default).
  - c. Per il tipo di pianificazione, scegli Pianificazione ricorrente e inserisci la seguente espressione cron: **cron(0 3 1 \* ? \*)**

Una volta valutata, questa espressione cron viene eseguita il primo giorno di ogni mese alle 03:00.

- d. Per la finestra temporale flessibile, scegli Disattivata.
- e. Lascia vuoto il campo Payload.



- f. Per il ruolo di esecuzione, scegli `Crea nuovo ruolo per questa pianificazione` e inserisci un nome per il ruolo. EventBridge Scheduler crea una nuova policy con le autorizzazioni necessarie alla pianificazione per richiamare la tua funzione.
6. Scegliere `Aggiungi`.

## AWS SAM

Per distribuire l'app utilizzando AWS SAM

1. Passa alla cartella in cui è stato salvato il file `template.yaml` per l'app. Nota che questo esempio utilizza due file `template.yaml`. Assicurati che siano salvati in sottocartelle separate e di trovarti nella cartella corretta contenente il modello per creare l'app.
2. Copia i file `lambda_function.py` e `requirements.txt` che hai scaricato in precedenza nella stessa cartella. La posizione del codice specificata nel AWS SAM modello è `./`, ovvero la posizione corrente. AWS SAM cerca in questa cartella il codice della funzione Lambda quando si tenta di distribuire l'app.
3. Eseguire il seguente comando seguente.

```
sam build --use-container
```

Questo comando raccoglie gli artefatti di compilazione per l'applicazione e li colloca nel formato e nella posizione corretti per l'implementazione. Specificando l'opzione `--use-container`, la funzione viene creata all'interno di un contenitore Docker. Lambda-like Lo usiamo qui quindi non è necessario che Python 3.12 sia installato sul computer locale per far funzionare la build.

4. Per creare le risorse Lambda e EventBridge Scheduler specificate nel `template.yaml` file, esegui il seguente comando.

```
sam deploy --guided
```

L'uso del `--guided` flag significa che AWS SAM vengono visualizzate istruzioni che guidano l'utente nel processo di distribuzione. Per questa implementazione, inserisci un Stack name pari a **cron-maintenance-app** e accetta i valori predefiniti per tutte le altre opzioni utilizzando `Invio`.

Al AWS SAM termine della creazione delle risorse Lambda e EventBridge Scheduler, dovresti visualizzare il seguente messaggio.

```
Successfully created/updated stack - cron-maintenance-app in us-west-2
```

5. Puoi inoltre confermare che la funzione Lambda è stata creata aprendo la pagina [Funzioni della console](#) Lambda. Dovresti vedere una funzione denominata `ScheduledDBMaintenance`

## Test dell'app

Per verificare che la pianificazione attivi correttamente la funzione e che la funzione pulisca correttamente i record dal database, è possibile modificare temporaneamente la pianificazione in modo che venga eseguita una volta in un momento specifico. È quindi possibile eseguire `sam deploy` nuovamente l'operazione per reimpostare il programma di ricorrenza in modo che venga eseguito una volta al mese.

Per eseguire l'applicazione utilizzando Console di gestione AWS

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliete la vostra funzione (`ScheduledDBMaintenance`).
3. Nella sezione Trigger, scegli il `DynamoDBCleanupSchedule` trigger.
4. Scegli Modifica.
5. Per Tipo di pianificazione, scegli One-time pianificazione.
6. Imposta il tempo di chiamata tra qualche minuto, quindi scegli Salva.

Dopo l'esecuzione della pianificazione e il richiamo della destinazione, esegui `test_app.py` lo script per verificare che la funzione abbia rimosso correttamente tutti i vecchi record dalla tabella `DynamoDB`.

Per verificare che i vecchi record vengano eliminati utilizzando uno script Python

1. Nella riga di comando, vai alla cartella in cui hai salvato. `test_app.py`
2. Eseguire lo script.

```
python test_app.py
```

In caso di successo, viene visualizzato il seguente risultato.

```
Total number of old records: 0
```

## Fasi successive

È ora possibile modificare la EventBridge pianificazione dello Scheduler per soddisfare i requisiti specifici dell'applicazione. EventBridge Scheduler supporta le seguenti espressioni di pianificazione: cron, rate e pianificazioni una tantum.

Per ulteriori informazioni sulle espressioni di EventBridge pianificazione di Scheduler, vedere Tipi di pianificazione [nella Guida per l'EventBridge utente di](#) Scheduler. [Gestione degli accessi](#) nella IAM User Guide

## Linguaggi di programmazione

Lambda supporta un'ampia gamma di linguaggi di programmazione popolari. Ognuno ha un'esperienza di programmazione idiomática personalizzata per quel linguaggio.

Ogni linguaggio di programmazione è supportato da un [runtime](#) corrispondente, che fornisce l'ambiente di esecuzione specifico del linguaggio per la funzione.

- [the section called “Costruire con Node.js”](#)
- [the section called “Costruire con TypeScript”](#)
- [the section called “Compilazione con Python”](#)
- [the section called “Compilazione con Ruby”](#)
- [the section called “Compilazione con Java”](#)
- [the section called “Compilazione con Go”](#)
- [the section called “Compilazione con C#”](#)
- [the section called “Costruire con PowerShell”](#)
- [the section called “Compilazione con Rust”](#)

## Creazione di funzioni Lambda con Node.js

Puoi eseguire il JavaScript codice con Node.js in. AWS Lambda Lambda fornisce i [runtime](#) per l' Node.js esecuzione del codice per l'elaborazione degli eventi. Il codice viene eseguito in un ambiente che include AWS SDK per JavaScript, con le credenziali di un ruolo AWS Identity and Access Management (IAM) che gestisci. Per saperne di più sulle versioni SDK incluse nei Node.js runtime, consulta. [the section called “Runtime-included Versioni SDK”](#)

Lambda supporta i seguenti runtime. Node.js

| Nome       | Identificatore | Sistema operativo | Data di ritiro  | Blocco creazione funzioni | Blocco aggiornam ento funzioni |
|------------|----------------|-------------------|-----------------|---------------------------|--------------------------------|
| Node.js 26 | nodejs26.x     | Amazon Linux 2023 | Non programmato | Non programmato           | Non programmato                |
| Node.js 24 | nodejs24.x     | Amazon Linux 2023 | 30 aprile 2028  | 1 giugno 2028             | 1 luglio 2028                  |
| Node.js 22 | nodejs22.x     | Amazon Linux 2023 | 30 aprile 2027  | 1 giugno 2027             | 1 luglio 2027                  |

Per creare una funzione Node.js

1. Aprire la [console Lambda](#).
2. Scegli Crea funzione.
3. Configura le impostazioni seguenti:
  - Nome della funzione: inserisci il nome della funzione.
  - Runtime: scegli Node.js 24.x.
4. Scegli Crea funzione.

La console crea una funzione Lambda con un singolo file di origine denominato `index.mjs`. È possibile modificare questo file e aggiungere altri file nell'editor di codice predefinito. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione. Quindi, per eseguire il codice, scegli Crea evento di test nella sezione EVENTI DI TEST.

Il file `index.mjs` esporta una funzione denominata `handler` che richiede un oggetto evento e un oggetto contesto. Questa è la [funzione del gestore](#) chiamata da Lambda quando la funzione viene richiamata. Il runtime della Node.js funzione riceve gli eventi di chiamata da Lambda e li passa al gestore. Nella configurazione della funzione il valore del gestore è `index.handler`.

Quando si salva il codice funzione, la console Lambda crea un pacchetto di implementazione dell'archivio di file `.zip`. Quando sviluppi il codice funzione al di fuori della console (utilizzando un IDE) devi [creare un pacchetto di implementazione](#) per caricare il codice nella funzione Lambda.

Il runtime della funzione passa un oggetto contesto al gestore, oltre all'evento di chiamata. L'[oggetto contesto](#) contiene ulteriori informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. Altre informazioni sono disponibili con le variabili di ambiente.

La funzione Lambda viene fornita con un gruppo di log Logs. CloudWatch Il runtime della funzione invia i dettagli su ogni chiamata a Logs. CloudWatch Si trasmette qualsiasi [log che la tua funzione emette](#) durante la chiamata. Se la funzione restituisce un errore, Lambda formatta l'errore e lo restituisce al chiamante.

## Argomenti

- [Runtime-included Versioni SDK](#)
- [Utilizzo di keep-alive per le connessioni TCP](#)
- [Caricamento di certificati CA](#)
- [Funzionalità sperimentali Node.js](#)
- [Definire il gestore di funzioni Lambda in Node.js](#)
- [Distribuisce funzioni Node.js Lambda con archivi di file `.zip`](#)
- [Implementa funzioni Node.js Lambda con immagini di container](#)
- [Utilizzo dei livelli per le funzioni Lambda Node.js](#)
- [Utilizzo dell'oggetto di contesto Lambda per recuperare le informazioni sulla funzione Node.js](#)
- [Registra e monitora le funzioni Node.js Lambda](#)
- [Codice di strumentazione in Node.js AWS Lambda](#)

## Runtime-included Versioni SDK

Tutti i Node.js runtime Lambda [supportati](#) includono una versione secondaria specifica della AWS SDK per JavaScript v3, non la versione più recente. <https://github.com/aws/aws-sdk-js-v3/releases>  
La versione secondaria specifica inclusa nel runtime dipende dalla versione di runtime e dalla tua.

Regione AWS Per trovare la versione specifica dell'SDK inclusa nel runtime che stai utilizzando, crea una funzione Lambda con il codice seguente.

Example index.mjs

```
import packageJson from '@aws-sdk/client-s3/package.json' with { type: 'json' };

export const handler = async () => ({ version: packageJson.version });
```

Restituisce una risposta nel seguente formato:

```
{
  "version": "3.632.0"
}
```

Per ulteriori informazioni, consulta [Usare l'SDK per la v3 nel tuo gestore JavaScript](#).

## Utilizzo di keep-alive per le connessioni TCP

L' Node.js HTTP/HTTPS agente predefinito crea una nuova connessione TCP per ogni nuova richiesta. Per evitare il costo della creazione di nuove connessioni, keep-alive è abilitato di default in tutti i [runtime supportati. Node.js](#) Keep-alive può ridurre i tempi di richiesta per le funzioni Lambda che effettuano più chiamate API utilizzando l'SDK.

Per disabilitare keep-alive, consulta [Riutilizzare le connessioni con keep-alive in Node.js](#) nella Guida per gli sviluppatori di SDK for 3.x. AWS JavaScript Per ulteriori informazioni sull'uso di keep-alive, vedi [HTTP keep-alive è attivo per impostazione predefinita nell'SDK modulare per nel Developer Tools Blog. AWS JavaScript](#) AWS

## Caricamento di certificati CA

Per le versioni Node.js di runtime fino alla Node.js 18, Lambda carica automaticamente i certificati Amazon-specific CA (autorità di certificazione) per semplificare la creazione di funzioni che interagiscono con altre. Servizi AWS Ad esempio, Lambda include i certificati Amazon RDS necessari per convalidare il [certificato di identità del server](#) installato sul tuo database Amazon RDS. Questo comportamento può avere un impatto sulle prestazioni durante gli avvii a freddo.

A partire da Node.js 20, Lambda non carica più certificati CA aggiuntivi per impostazione predefinita. Il runtime Node.js 20 contiene un file di certificato con tutti i certificati Amazon CA disponibili in `var/runtime/ca-cert.pem`. Per ripristinare lo stesso comportamento a partire da Node.js 18 runtime

e precedenti, imposta la variabile di `NODE_EXTRA_CA_CERTS` [ambiente](#) su `/var/runtime/ca-cert.pem`.

Per prestazioni ottimali, consigliamo di raggruppare solo i certificati necessari con il pacchetto di distribuzione e di caricarli tramite la `NODE_EXTRA_CA_CERTS` variabile di ambiente. Il file dei certificati deve essere composto da uno o più certificati CA root o intermedi affidabili in formato PEM. Ad esempio, per RDS, includi i certificati richiesti insieme al codice come `certificates/rds.pem`. Quindi, carica i certificati impostando `NODE_EXTRA_CA_CERTS` su `/var/task/certificates/rds.pem`.

## Funzionalità sperimentali Node.js

Le versioni Node.js linguistiche upstream abilitano alcune funzionalità sperimentali per impostazione predefinita. Lambda disabilita queste funzionalità per garantire stabilità di runtime e prestazioni costanti. La tabella seguente elenca le funzionalità sperimentali disabilite da Lambda.

| Funzionalità sperimentale                                                             | Node.js Versioni supportate        | Node.js flag applicato da Lambda              | Flag Lambda da riattivare                  |
|---------------------------------------------------------------------------------------|------------------------------------|-----------------------------------------------|--------------------------------------------|
| Supporto per l'importazione di moduli utilizzando i moduli <code>require</code> in ES | Node.js 20, Node.js 22, Node.js 24 | <code>--no-experimental-require-module</code> | <code>--experimental-require-module</code> |
| Supporto per il rilevamento automatico dei moduli ES e CommonJS                       | Node.js 22, 24 Node.js             | <code>--no-experimental-detect-module</code>  | <code>--experimental-detect-module</code>  |

Per abilitare una funzionalità sperimentale disattivata, imposta il flag di riabilitazione nella variabile di `NODE_OPTIONS` ambiente. Ad esempio, per abilitare il modulo ES richiedi supporto, imposta `NODE_OPTIONS` su `--experimental-require-module`. Lambda rileva questo override e rimuove il corrispondente flag di disabilitazione.

### Important

L'utilizzo di funzionalità sperimentali può causare instabilità e problemi di prestazioni. Queste funzionalità potrebbero essere modificate o rimosse nelle Node.js versioni future. Le

funzioni che utilizzano funzionalità sperimentali non sono idonee per il Lambda Service Level Agreement (SLA) o. Supporto AWS



## Definire il gestore di funzioni Lambda in Node.js

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come lavorare con i gestori di funzioni Lambda in Node.js, comprese le opzioni per la configurazione del progetto, le convenzioni di denominazione e le best practice. Questa pagina include anche un esempio di funzione Node.js Lambda che acquisisce informazioni su un ordine, produce una ricevuta in un file di testo e lo inserisce in un bucket Amazon Simple Storage Service (Amazon S3). Per informazioni su come distribuire una funzione dopo averla scritta, consulta o [the section called “Implementazione di archivi di file .zip”](#) [the section called “Implementazione di immagini di container”](#)

### Argomenti

- [Configurazione del progetto handler Node.js](#)
- [Esempio di codice della Node.js funzione Lambda](#)
- [Moduli CommonJS ed ES](#)
- [Node.js inizializzazione](#)
- [Convenzioni di denominazione dei gestori](#)
- [Definizione e accesso all'oggetto evento di input](#)
- [Schemi di gestione validi per le funzioni Node.js](#)
- [Usare l'SDK per la v3 nel tuo gestore JavaScript](#)
- [Accesso alle variabili d'ambiente](#)
- [Utilizzo dello stato globale](#)
- [Procedure consigliate per il codice per le funzioni Lambda Node.js](#)

### Configurazione del progetto handler Node.js

Esistono diversi modi per inizializzare un progetto Node.js Lambda. Ad esempio, è possibile creare un Node.js progetto standard utilizzando npm, creare un'[AWS SAM applicazione](#) o creare un'[AWS CDK applicazione](#).

Per creare il progetto utilizzandonpm:

```
npm init
```

Questo comando inizializza il progetto e genera un `package.json` file che gestisce i metadati e le dipendenze del progetto.

Il codice della funzione si trova in un file `or.js` o `or.mjs` JavaScript. Nel seguente esempio, chiamiamo questo file `index.mjs` perché utilizza un gestore di moduli ES. Lambda supporta sia il modulo ES che i gestori CommonJS. Per ulteriori informazioni, consulta [Moduli CommonJS ed ES](#).

Un tipico progetto di funzione Node.js Lambda segue questa struttura generale:

```
/project-root
  ### index.mjs - Contains main handler
  ### package.json - Project metadata and dependencies
  ### package-lock.json - Dependency lock file
  ### node_modules/ - Installed dependencies
```

### Esempio di codice della Node.js funzione Lambda

Il seguente codice di funzione Lambda di esempio acquisisce informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3.

### Example `index.mjs`, funzione Lambda

```
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';

// Initialize the S3 client outside the handler for reuse
const s3Client = new S3Client();

/**
 * Lambda handler for processing orders and storing receipts in S3.
 * @param {Object} event - Input event containing order details
 * @param {string} event.order_id - The unique identifier for the order
 * @param {number} event.amount - The order amount
 * @param {string} event.item - The item purchased
 * @returns {Promise<string>} Success message
 */
export const handler = async(event) => {
  try {
    // Access environment variables
    const bucketName = process.env.RECEIPT_BUCKET;
    if (!bucketName) {
      throw new Error('RECEIPT_BUCKET environment variable is not set');
    }
  }
}
```

```

    // Create the receipt content and key destination
    const receiptContent = `OrderID: ${event.order_id}\nAmount: $
    ${event.amount.toFixed(2)}\nItem: ${event.item}`;
    const key = `receipts/${event.order_id}.txt`;

    // Upload the receipt to S3
    await uploadReceiptToS3(bucketName, key, receiptContent);

    console.log(`Successfully processed order ${event.order_id} and stored receipt
    in S3 bucket ${bucketName}`);
    return 'Success';
  } catch (error) {
    console.error(`Failed to process order: ${error.message}`);
    throw error;
  }
};

/**
 * Helper function to upload receipt to S3
 * @param {string} bucketName - The S3 bucket name
 * @param {string} key - The S3 object key
 * @param {string} receiptContent - The content to upload
 * @returns {Promise<void>}
 */
async function uploadReceiptToS3(bucketName, key, receiptContent) {
  try {
    const command = new PutObjectCommand({
      Bucket: bucketName,
      Key: key,
      Body: receiptContent
    });

    await s3Client.send(command);
  } catch (error) {
    throw new Error(`Failed to upload receipt to S3: ${error.message}`);
  }
}

```

Questo file `index.mjs` contiene le sezioni seguenti:

- **Blocco import:** utilizza questo blocco per includere le librerie richieste dalla funzione Lambda, come i [client SDK AWS](#).

- `const s3Client` dichiarazione: inizializza un client [Amazon S3 al di fuori della funzione handler](#). Ciò fa sì che Lambda esegua questo codice durante la fase di inizializzazione e il client viene preservato per essere riutilizzato in più invocazioni. [???](#)
- Blocco di commenti JSDoc: definisci i tipi di input e output per il tuo gestore utilizzando le annotazioni JSDoc. <https://jsdoc.app/about-getting-started>
- `export const handler`: Questa è la funzione principale del gestore richiamata da Lambda. Quando distribuisce la tua funzione, specifica `index.handler` la proprietà Handler. [https://docs.aws.amazon.com/lambda/latest/api/API\\_CreateFunction.html#lambda-CreateFunction-request-Handler](https://docs.aws.amazon.com/lambda/latest/api/API_CreateFunction.html#lambda-CreateFunction-request-Handler) Il valore della Handler proprietà è il nome del file e il nome del metodo del gestore esportato, separati da un punto.
- `uploadReceiptToS3function`: Questa è una funzione di supporto a cui fa riferimento la funzione principale del gestore.

Affinché questa funzione funzioni correttamente, il suo ruolo di [esecuzione](#) deve consentire l'azione. `s3:PutObject` Inoltre, assicuratevi di definire la variabile di ambiente `RECEIPT_BUCKET`. Dopo una chiamata riuscita, il bucket Amazon S3 dovrebbe contenere un file di ricevuta.

## Moduli CommonJS ed ES

Node.js supporta due sistemi di moduli: moduli CommonJS ed ECMAScript (moduli ES). Lambda consiglia di utilizzare i moduli ES in quanto supporta l'attesa di primo livello, che consente il completamento di attività asincrone durante l'inizializzazione dell'ambiente di esecuzione. [???](#)

Node.js tratta i file con estensione `.mjs` come moduli CommonJS mentre un' `.cjs` estensione indica i moduli ES. Per impostazione predefinita, Node.js tratta i file con estensione `.js` come moduli CommonJS. È possibile configurare il trattamento dei file `.js` come moduli ES specificando `type="module"` nel file della funzione. `package.json` È possibile configurare Node.js in Lambda per rilevare automaticamente se un `.js` file deve essere trattato come CommonJS o come modulo ES aggiungendo il `--experimental-detect-module` flag alla variabile di ambiente. `NODE_OPTIONS` Per ulteriori informazioni, consulta Funzionalità [sperimentali Node.js](#).

Gli esempi seguenti mostrano i gestori di funzioni scritti utilizzando sia i moduli ES che i moduli CommonJS. Gli esempi rimanenti in questa pagina utilizzano tutti moduli ES.

## ES module example

### Example— Gestore di moduli ES

```
const url = "https://aws.amazon.com/";

export const handler = async(event) => {
  try {
    const res = await fetch(url);
    console.info("status", res.status);
    return res.status;
  }
  catch (e) {
    console.error(e);
    return 500;
  }
};
```

## CommonJS module example

### Example— gestore di moduli CommonJS

```
const https = require("https");
let url = "https://aws.amazon.com/";

exports.handler = async function (event) {
  let statusCode;
  await new Promise(function (resolve, reject) {
    https.get(url, (res) => {
      statusCode = res.statusCode;
      resolve(statusCode);
    }).on("error", (e) => {
      reject(Error(e));
    });
  });
  console.log(statusCode);
  return statusCode;
};
```

## Node.js inizializzazione

Node.js utilizza un I/O modello non bloccante che supporta operazioni asincrone efficienti utilizzando un ciclo di eventi. Ad esempio, se Node.js effettua una chiamata di rete, la funzione continua a elaborare altre operazioni senza bloccare una risposta di rete. Quando la risposta di rete viene ricevuta, viene inserita nella coda di callback. Le attività della coda vengono elaborate al termine dell'attività corrente.

Lambda consiglia di utilizzare `await` di primo livello in modo che le attività asincrone avviate durante l'inizializzazione dell'ambiente di esecuzione vengano completate durante l'inizializzazione. Le attività asincrone che non vengono completate durante l'inizializzazione vengono in genere eseguite durante la prima chiamata di funzione. Ciò può causare comportamenti o errori imprevisti.

Ad esempio, l'inizializzazione della funzione potrebbe effettuare una chiamata di rete per recuperare un AWS parametro da Parameter Store. Se questa attività non viene completata durante l'inizializzazione, il valore potrebbe essere nullo durante una chiamata. Può inoltre verificarsi un ritardo tra l'inizializzazione e l'invoke, che può causare errori nelle operazioni sensibili al fattore tempo. In particolare, le chiamate AWS di assistenza possono basarsi su firme di richieste sensibili al fattore tempo, con conseguenti errori delle chiamate di servizio se la chiamata non viene completata durante la fase di inizializzazione.

Il completamento delle attività durante l'inizializzazione in genere migliora le prestazioni di avvio a freddo e le richiama innanzitutto quando si utilizza Provisioned Concurrency. Per ulteriori informazioni, consulta il nostro post sul blog [Using Node.js ES modules and top-level await in. AWS Lambda](#)

## Convenzioni di denominazione dei gestori

Quando si configura una funzione, il valore dell'[impostazione](#) Handler è il nome del file e il nome del metodo handler esportato, separati da un punto. L'impostazione predefinita per le funzioni create nella console e negli esempi in questa guida è `index.handler`. Questo indica il metodo handler che viene esportato dal file `index.js` o `index.mjs`.

Se si crea una funzione nella console utilizzando un nome di file o un nome del gestore di funzione diverso, è necessario modificare il nome del gestore predefinito.

## Modifica del nome del gestore funzioni (console)

1. Apri la pagina [Funzioni](#) della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.

3. Scorri verso il basso fino al riquadro Impostazioni di runtime e scegli Modifica.
4. In Gestore, inserisci il nuovo nome per il tuo gestore di funzioni.
5. Scegli Save (Salva).

### Definizione e accesso all'oggetto evento di input

JSON è il formato di input più comune e standard per le funzioni Lambda. In questo esempio, la funzione prevede un input simile a quanto segue:

```
{
  "order_id": "12345",
  "amount": 199.99,
  "item": "Wireless Headphones"
}
```

Quando si lavora con le funzioni Lambda in Node.js, è possibile definire la forma prevista dell'evento di input utilizzando le annotazioni JSDoc. In questo esempio, definiamo la struttura di input nel commento JSDoc del gestore:

```
/**
 * Lambda handler for processing orders and storing receipts in S3.
 * @param {Object} event - Input event containing order details
 * @param {string} event.order_id - The unique identifier for the order
 * @param {number} event.amount - The order amount
 * @param {string} event.item - The item purchased
 * @returns {Promise<string>} Success message
 */
```

Dopo aver definito questi tipi nel commento JSDoc, puoi accedere ai campi dell'oggetto evento direttamente nel codice. Ad esempio, `event.order_id` recupera il valore di `order_id` dall'input originale.

### Schemi di gestione validi per le funzioni Node.js

Si consiglia di utilizzare [async/await](https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/using-a-callback-function.html) per dichiarare il gestore della funzione invece di utilizzare i callback. <https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/using-a-callback-function.html> Async/await è un modo conciso e leggibile per scrivere codice asincrono, senza la necessità di callback annidati o di concatenare promesse. Con `async/await`, puoi scrivere codice che si legge come codice sincrono, pur essendo asincrono e non bloccante.

## gestori di funzioni asincrone (consigliati)

La parola chiave `async` contrassegna una funzione come asincrona, mentre la parola chiave `await` mette in pausa l'esecuzione della funzione fino alla risoluzione di `Promise`. Il gestore accetta i seguenti argomenti:

- `event`: contiene i dati di input passati alla funzione.
- `context`: contiene informazioni sull'invocazione, sulla funzione e sull'ambiente di esecuzione.

Per ulteriori informazioni, consulta [Utilizzo dell'oggetto di contesto Lambda per recuperare le informazioni sulla funzione Node.js](#).

Ecco le firme valide per il `async/await` pattern:

```
export const handler = async (event) => { };
```

```
export const handler = async (event, context) => { };
```

## Gestori di funzioni sincroni

Se la funzione non esegue alcuna attività asincrona, è possibile utilizzare un gestore di funzioni sincrone, utilizzando una delle seguenti firme di funzione:

```
export const handler = (event) => { };
```

```
export const handler = (event, context) => { };
```

## gestori di funzioni di streaming delle risposte

Lambda supporta lo streaming di risposte con Node.js. I gestori delle funzioni di streaming delle risposte utilizzano il `awslambda.streamifyResponse()` decoratore e accettano 3 parametri: `event`, `e. responseStream` e `context`. La firma della funzione è:

```
export const handler = awslambda.streamifyResponse(async (event, responseStream, context) => { });
```

Per ulteriori informazioni, consulta [Streaming di risposte per le funzioni Lambda](#).



## Callback-based gestori di funzioni

### Note

Callback-based i gestori di funzioni sono supportati solo fino a 22. Node.js A partire da Node.js 24, le attività asincrone dovrebbero essere implementate utilizzando gestori di funzioni asincroni.

Callback-based i gestori di funzioni devono utilizzare gli argomenti event, context e callback. Esempio:

```
export const handler = (event, context, callback) => { };
```

La funzione di callback prevede una `Error` e una risposta, che deve essere. JSON-serializable La funzione continua a essere eseguita fino a quando il ciclo di [eventi non](#) è vuoto o la funzione scade. La risposta non viene inviata all'invoker finché tutte le attività del ciclo di eventi non giungono a termine. Se la funzione va in timeout, viene invece restituito un errore. È possibile configurare il runtime per inviare immediatamente la risposta impostando [context.callbackWaitsForEmptyEventLoop](#) su false.

### Example– richiesta HTTP con callback

La seguente funzione di esempio controlla un URL e restituisce il codice di stato all'invoker.

```
import https from "https";
let url = "https://aws.amazon.com/";

export const handler = (event, context, callback) => {
  https.get(url, (res) => {
    callback(null, res.statusCode);
  }).on("error", (e) => {
    callback(Error(e));
  });
};
```

### Usare l'SDK per la v3 nel tuo gestore JavaScript

Spesso, utilizzerai le funzioni Lambda per interagire o apportare aggiornamenti ad altre AWS risorse. Il modo più semplice per interfacciarsi con queste risorse è usare la AWS SDK per JavaScript. Tutti i Node.js runtime Lambda [supportati](#) includono l'[SDK per la versione 3. JavaScript](#)

Tuttavia, consigliamo vivamente di includere i client AWS SDK necessari nel pacchetto di distribuzione. Ciò massimizza la [compatibilità con le versioni precedenti](#) durante i futuri aggiornamenti del runtime di Lambda. Affidati all'SDK fornito in fase di esecuzione solo quando non puoi includere pacchetti aggiuntivi (ad esempio, quando usi l'editor di codice della console Lambda o il codice in linea in un modello). AWS CloudFormation

Per aggiungere dipendenze SDK alla tua funzione, usa il `npm install` comando per i client SDK specifici di cui hai bisogno. Nel codice di esempio, abbiamo utilizzato il client Amazon S3. <https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/client/s3/> Aggiungi queste dipendenze eseguendo i seguenti comandi nella directory che contiene il file `package.json`:

```
npm install @aws-sdk/client-s3
```

Nel codice della funzione, importa il client e i comandi necessari, come dimostra la funzione di esempio:

```
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';
```

Quindi, inizializza un client [Amazon S3](#):

```
const s3Client = new S3Client();
```

In questo esempio, abbiamo inizializzato il nostro client Amazon S3 all'esterno della funzione del gestore principale per evitare di doverlo inizializzare ogni volta che invochiamo la nostra funzione. Dopo aver inizializzato il client SDK, puoi utilizzarlo per effettuare chiamate API per quel servizio. AWS Il codice di esempio richiama l'azione [PutObject](#) API Amazon S3 nel modo seguente:

```
const command = new PutObjectCommand({
  Bucket: bucketName,
  Key: key,
  Body: receiptContent
});
```

## Accesso alle variabili d'ambiente

Nel codice del gestore, puoi fare riferimento a qualsiasi variabile di [ambiente](#) utilizzando `process.env`. In questo esempio, facciamo riferimento alla variabile di ambiente `RECEIPT_BUCKET` definita utilizzando le seguenti righe di codice:

```
// Access environment variables
const bucketName = process.env.RECEIPT_BUCKET;
if (!bucketName) {
    throw new Error('RECEIPT_BUCKET environment variable is not set');
}
```

## Utilizzo dello stato globale

Lambda esegue il codice statico durante la fase di [inizializzazione](#) prima di richiamare la funzione per la prima volta. Le risorse create durante l'inizializzazione restano in memoria tra le invocazioni, in modo da evitare di doverle creare ogni volta che si invoca la funzione.

Nel codice di esempio, il codice di inizializzazione del client S3 è esterno al gestore. Il runtime inietta il client prima che la funzione gestisca il primo evento e il client rimane disponibile per il riutilizzo in tutte le chiamate.

Procedure consigliate per il codice per le funzioni Lambda Node.js

Segui queste linee guida quando crei funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. In questo modo è possibile creare una funzione di cui è più semplice eseguire l'unit test.
- Controllare le dipendenze nel pacchetto di distribuzione della funzione. L'ambiente di AWS Lambda esecuzione contiene una serie di librerie. Per i runtime di Python Node.js e Python, questi includono gli AWS SDK. Per abilitare il set di caratteristiche e aggiornamenti della sicurezza più recenti, Lambda aggiorna periodicamente tali librerie. Tali aggiornamenti possono introdurre lievi modifiche al comportamento della funzione Lambda. Per mantenere il controllo completo delle dipendenze utilizzate dalla funzione, inserire tutte le dipendenze nel pacchetto di implementazione.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del [contesto di esecuzione](#).
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. In questo modo viene ridotta la quantità di tempo necessaria per il download del pacchetto e per la relativa decompressione prima dell'invocazione.

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione.

Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive

elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedi [Riutilizzo delle connessioni con in. Keep-Alive Node.js](#)

Utilizzare [le variabili di ambiente](#) per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su 0 per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionali alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha una dipendenza su queste API non pubbliche. Consulta il [riferimento all'API](#) per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta [Come posso rendere idempotente la mia funzione Lambda?](#).

## Distribuisci funzioni Node.js Lambda con archivi di file .zip

Il codice della AWS Lambda funzione è costituito da un file .js o .mjs contenente il codice del gestore della funzione, insieme a eventuali pacchetti e moduli aggiuntivi da cui dipende il codice. Per implementare questo codice della funzione in Lambda, utilizza un pacchetto di implementazione. Questo pacchetto può essere un archivio di file .zip o un'immagine di container. Per ulteriori informazioni sull'utilizzo di immagini container con Node.js, consulta [Deploy Node.js Lambda functions with container images](#).

Per creare un pacchetto di implementazione come archivio di file .zip, puoi utilizzare l'utilità di archiviazione di file .zip incorporata del tuo strumento della linea di comando o qualsiasi altra utilità file .zip, come ad esempio [7zip](#). Gli esempi mostrati nelle sezioni seguenti presuppongono che tu stia utilizzando uno strumento della linea di comando zip in un ambiente Linux o MacOS. Per utilizzare gli stessi comandi in Windows, puoi [installare il sottosistema Windows per Linux per](#) ottenere una Windows-integrated versione di Ubuntu e Bash.

Nota che Lambda utilizza le autorizzazioni dei file POSIX, quindi potresti aver bisogno di [impostare le autorizzazioni per la cartella del pacchetto di implementazione](#) prima di creare l'archivio di file .zip.

### Argomenti

- [Dipendenze di runtime in Node.js](#)
- [Creazione di un pacchetto di implementazione .zip senza dipendenze](#)
- [Creazione di un pacchetto di implementazione .zip con dipendenze](#)
- [Creare un livello per le tue Node.js dipendenze](#)
- [Percorso di ricerca delle dipendenze e librerie incluse nel runtime](#)
- [Creazione e aggiornamento di funzioni Node.js Lambda utilizzando file .zip](#)

### Dipendenze di runtime in Node.js

Per le funzioni Lambda che utilizzano il Node.js runtime, una dipendenza può essere qualsiasi modulo. Node.js Il Node.js runtime include una serie di librerie comuni, oltre a una versione di. AWS SDK per JavaScript Tutti i Node.js runtime [supportati](#) includono la versione 3 dell'SDK. Per utilizzare la versione 2 dell'SDK, devi aggiungere l'SDK al pacchetto di distribuzione del file .zip. Per trovare la versione specifica dell'SDK inclusa nel runtime che stai utilizzando, consulta. [the section called "Runtime-included Versions SDK"](#)

Lambda aggiorna periodicamente le librerie SDK nel Node.js runtime per includere le funzionalità e gli aggiornamenti di sicurezza più recenti. Lambda applica anche patch di sicurezza e aggiornamenti alle altre librerie incluse nel runtime. Per avere il pieno controllo delle dipendenze del pacchetto, puoi aggiungere la tua versione preferita di qualsiasi dipendenza inclusa nel runtime al tuo pacchetto di implementazione. Ad esempio, se desideri utilizzare una particolare versione dell'SDK per JavaScript, puoi includerla nel tuo file .zip come dipendenza. Per ulteriori informazioni sull'aggiunta di dipendenze incluse nel runtime al tuo file .zip, consulta la pagina [Percorso di ricerca delle dipendenze e librerie incluse nel runtime](#).

In base al [modello di responsabilità condivisa di AWS](#), è tua responsabilità gestire eventuali dipendenze nei pacchetti di implementazione delle tue funzioni. Ciò include l'applicazione di aggiornamenti e patch di sicurezza. Per aggiornare le dipendenze nel pacchetto di implementazione della funzione, crea prima un nuovo file .zip e poi caricalo su Lambda. Per ulteriori informazioni, consulta [Creazione di un pacchetto di implementazione .zip con dipendenze](#) e [Creazione e aggiornamento di funzioni Node.js Lambda utilizzando file .zip](#).

#### Creazione di un pacchetto di implementazione .zip senza dipendenze

Se il codice della funzione non ha dipendenze ad eccezione delle librerie incluse nel runtime Lambda, il file .zip contiene solo il `index.mjs` file `index.js` o con il codice del gestore della funzione. Utilizza il tuo strumento di compressione preferito per creare un file .zip con il file `index.js` o `index.mjs` nella directory principale. Se il file contenente il codice del gestore non è alla radice del file .zip, Lambda non sarà in grado di eseguire il codice.

Per informazioni su come implementare il file .zip per creare una nuova funzione Lambda o aggiornarne una esistente, consulta la sezione [Creazione e aggiornamento di funzioni Node.js Lambda utilizzando file .zip](#).

#### Creazione di un pacchetto di implementazione .zip con dipendenze

Se il codice della funzione dipende da pacchetti o moduli non inclusi nel Node.js runtime Lambda, puoi aggiungere queste dipendenze al tuo file .zip con il tuo codice funzione o utilizzare un layer Lambda. [???](#) Le istruzioni in questa sezione mostrano come includere le dipendenze nel pacchetto di implementazione .zip. Per istruzioni sull'inclusione di dipendenze in un livello, consulta [the section called “Creare un livello per le tue Node.js dipendenze”](#).

I seguenti comandi della CLI di esempio creano un file .zip denominato `my_deployment_package.zip` contenente il file `index.js` o `index.mjs` con il codice del gestore della funzione e le relative dipendenze. Nell'esempio, installi le dipendenze utilizzando il gestore di pacchetti npm.

## Per creare il pacchetto di implementazione

1. Passa alla directory del progetto contenente il file del codice sorgente `index.js` o `index.mjs`. In questo esempio, la directory è denominata `my_function`.

```
cd my_function
```

2. Installa le librerie richieste dalla tua funzione nella directory `node_modules` utilizzando il comando `npm install`. In questo esempio installi l' SDK AWS X-Ray per Node.js.

```
npm install aws-xray-sdk
```

Questo crea una struttura di cartelle simile alla seguente:

```
~/my_function
### index.mjs
### node_modules
    ### async
    ### async-listener
    ### atomic-batcher
    ### aws-sdk
    ### aws-xray-sdk
    ### aws-xray-sdk-core
```

Al tuo pacchetto di implementazione puoi anche aggiungere moduli personalizzati che crei in autonomia. Crea una directory in `node_modules` con il nome del tuo modulo e salva lì i tuoi pacchetti personalizzati.

3. Crea un file `.zip` dei contenuti della cartella di progetto della directory principale. Utilizza l'opzione (ricorsiva) `-r` per garantire che lo zip comprima le sottocartelle.

```
zip -r my_deployment_package.zip .
```

## Creare un livello per le tue Node.js dipendenze

Le istruzioni in questa sezione spiegano come includere dipendenze in un livello. Per istruzioni sull'inclusione di dipendenze in un pacchetto di implementazione, consulta [the section called “Creazione di un pacchetto di implementazione .zip con dipendenze”](#). Per ulteriori informazioni sulla creazione e la gestione dei Node.js livelli, vedere [the section called “Livelli”](#).

Quando si aggiunge un livello a una funzione, Lambda carica il contenuto del livello nella directory `/opt` di quell'ambiente di esecuzione. Per ogni runtime Lambda, la variabile `PATH` include percorsi di cartelle specifici nella directory `/opt`. Per garantire che Lambda raccolga il contenuto del livello, il file `.zip` del livello dovrebbe avere le sue dipendenze in uno dei seguenti percorsi di cartella:

- `nodejs/node_modules`
- `nodejs/node18/node_modules` (`NODE_PATH`)
- `nodejs/node20/node_modules` (`NODE_PATH`)
- `nodejs/node22/node_modules` (`NODE_PATH`)

Ad esempio, la struttura del file `.zip` del livello potrebbe essere simile alla seguente:

```
xray-sdk.zip
# nodejs/node_modules/aws-xray-sdk
```

Lambda rileva automaticamente tutte le librerie nella directory `/opt/lib` e tutti i file binari nella directory `/opt/bin`. Per accertarti che Lambda trovi correttamente il contenuto del tuo livello, crea un livello con la seguente struttura:

```
custom-layer.zip
# lib
  | lib_1
  | lib_2
# bin
  | bin_1
  | bin_2
```

Dopo aver creato un pacchetto del livello, consulta [the section called “Creazione ed eliminazione di livelli”](#) e [the section called “Aggiunta di livelli”](#) per completare l'impostazione del livello.

## Percorso di ricerca delle dipendenze e librerie incluse nel runtime

Il Node.js runtime include una serie di librerie comuni, oltre a una versione di AWS SDK per JavaScript. Se desideri utilizzare una versione diversa di una libreria inclusa nel runtime, puoi farlo raggruppandola con la tua funzione o aggiungendola come dipendenza nel tuo pacchetto di implementazione. Ad esempio, puoi utilizzare una versione diversa dell'SDK aggiungendola al tuo pacchetto di implementazione `.zip`. Puoi anche includerlo in un [livello Lambda](#) per la tua funzione.



Quando si utilizza un'istruzione `import` o nel codice, il Node.js runtime cerca nelle directory del `NODE_PATH` percorso finché non trova il modulo. Per impostazione predefinita, la prima posizione cercata dal runtime è la directory in cui il pacchetto di implementazione `.zip` viene decompresso e montato (`/var/task`). Se includi una versione di una libreria inclusa nel runtime nel pacchetto di distribuzione, questa versione ha la precedenza sulla versione inclusa nel runtime. Le dipendenze nel pacchetto di implementazione hanno la precedenza anche sulle dipendenze nei livelli.

Quando aggiungi una dipendenza a un livello, Lambda la estrae in `/opt/nodejs/nodexx/node_modules`, dove `nodexx` rappresenta la versione del runtime che stai utilizzando. Nel percorso di ricerca, questa directory ha la precedenza sulla directory contenente le librerie incluse nel runtime (`/var/lang/lib/node_modules`). Le librerie nei livelli di funzione hanno quindi la precedenza sulle versioni incluse nel runtime.

Puoi visualizzare il percorso di ricerca completo per la tua funzione Lambda aggiungendo la seguente riga di codice.

```
console.log(process.env.NODE_PATH)
```

Puoi anche aggiungere dipendenze in una cartella separata all'interno del tuo pacchetto `.zip`. Ad esempio, potresti aggiungere un modulo personalizzato a una cartella del tuo pacchetto `.zip` denominata `common`. Quando il pacchetto `.zip` viene decompresso e montato, questa cartella viene inserita nella directory `/var/task`. Per utilizzare una dipendenza da una cartella del pacchetto di implementazione `.zip` nel codice, utilizza un'istruzione `import { } from` o `const { } = require()`, a seconda che tu stia utilizzando la risoluzione del modulo CJS o ESM. Ad esempio:

```
import { myModule } from './common'
```

Se raggruppi il codice con `esbuild`, `rollup` o qualcosa di simile, le dipendenze utilizzate dalla funzione vengono raggruppate in uno o più file. Si consiglia di utilizzare questo metodo per eliminare le dipendenze ogni volta che è possibile. Rispetto all'aggiunta di dipendenze al pacchetto di distribuzione, il raggruppamento del codice comporta un miglioramento delle prestazioni grazie alla riduzione delle operazioni. I/O

## Creazione e aggiornamento di funzioni Node.js Lambda utilizzando file `.zip`

Dopo aver creato il pacchetto di implementazione `.zip`, puoi utilizzarlo per creare una nuova funzione Lambda o aggiornarne una esistente. Puoi distribuire il tuo pacchetto `.zip` utilizzando la console Lambda, l'API Lambda e l'AWS Command Line Interface API Lambda. Puoi anche

creare e aggiornare le funzioni Lambda usando AWS Serverless Application Model (AWS SAM) e CloudFormation.

La dimensione massima per un pacchetto di implementazione .zip per Lambda è di 250 MB (dopo l'estrazione). Nota che questo limite si applica alla dimensione combinata di tutti i file caricati, inclusi eventuali livelli Lambda.

Il runtime Lambda necessita dell'autorizzazione per leggere i file nel pacchetto di distribuzione. Nella notazione ottale delle autorizzazioni di Linux, Lambda richiede 644 autorizzazioni per i file non eseguibili (rw-r--r--) e 755 autorizzazioni (rwxr-xr-x) per le directory e i file eseguibili.

In Linux e macOS, utilizza il comando `chmod` per modificare le autorizzazioni file su file e directory nel pacchetto di implementazione. Ad esempio, per assegnare a un file non eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 644 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta [Set, View, Change, or Remove Permissions on an Object](#) nella documentazione di Microsoft Windows.

#### Note

Se non concedi a Lambda le autorizzazioni necessarie per accedere alle directory del pacchetto di implementazione, Lambda imposta le autorizzazioni per tali directory su 755 (rwxr-xr-x).

## Creazione e aggiornamento delle funzioni con file .zip utilizzando la console

Per creare una nuova funzione, devi prima creare la funzione nella console, quindi devi caricare il tuo archivio .zip. Per aggiornare una funzione esistente, apri la pagina relativa alla funzione, quindi segui la stessa procedura per aggiungere il file .zip aggiornato.

Se il file .zip ha dimensioni inferiori a 50 MB, è possibile creare o aggiornare una funzione caricando il file direttamente dal computer locale. Per i file .zip di dimensioni superiori a 50 MB, prima è necessario caricare il pacchetto in un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il Console di gestione AWS, consulta [Guida introduttiva ad Amazon S3](#). Per caricare file utilizzando il AWS CLI, consulta [Sposta oggetti](#) nella Guida per l'AWS CLI utente.

 Note

Non è possibile modificare il [tipo di pacchetto di implementazione](#) (.zip o immagine di container) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio di file .zip. È necessario creare una nuova funzione.

### Creazione di una nuova funzione (console)

1. Apri la [pagina Funzioni](#) della console Lambda e scegli Crea funzione.
2. Scegli Author from scratch (Crea da zero).
3. In Basic information (Informazioni di base) eseguire queste operazioni:
  - a. In Nome funzione, inserisci il nome della funzione.
  - b. Per Runtime, seleziona il runtime che desideri utilizzare.
  - c. (Facoltativo) Per Architettura, scegli l'architettura del set di istruzioni per la funzione. L'architettura predefinita è x86\_64. Assicurati che il pacchetto di implementazione per la tua funzione sia compatibile con l'architettura del set di istruzioni scelta.
4. (Opzionale) In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito. Puoi creare un nuovo ruolo di esecuzione o utilizzare un ruolo esistente.
5. Scegli Crea funzione. Lambda crea una funzione di base "Hello world" utilizzando il runtime scelto.

### Caricamento di un archivio .zip dal computer locale (console)

1. Nella [pagina Funzioni](#) della console Lambda, scegli la funzione per cui vuoi caricare il file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli File .zip.
5. Per caricare il file .zip, procedi come segue:
  - a. Seleziona Carica, quindi seleziona il tuo file .zip nel selettore di file.
  - b. Seleziona Apri.
  - c. Scegli Save (Salva).

## Caricamento di un archivio .zip da un bucket Amazon S3 (console)

1. Nella [pagina Funzioni](#) della console Lambda, scegli la funzione per cui vuoi caricare un nuovo file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli Posizione Amazon S3.
5. Incolla l'URL del link Amazon S3 del tuo file .zip e scegli Salva.

## Aggiornamento delle funzioni dei file .zip tramite l'editor di codice della console

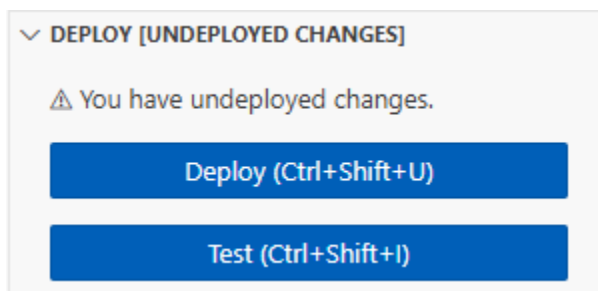
Per alcune funzioni con pacchetti di implementazione .zip, puoi utilizzare l'editor di codice integrato nella console Lambda per aggiornare direttamente il codice della funzione. Per utilizzare questa funzione, la funzione deve soddisfare i seguenti criteri:

- La funzione deve utilizzare uno dei runtime del linguaggio interpretato (Python o Node.js Ruby)
- Il pacchetto di implementazione della funzione deve avere dimensioni inferiori a 50 MB (non compresso).

Il codice della funzione per le funzioni con pacchetti di implementazione di immagini di container non può essere modificato direttamente nella console.

## Aggiornamento del codice della funzione utilizzando l'editor di codice della console

1. Apri la [pagina Funzioni](#) della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, seleziona il tuo file di codice sorgente e modificalo nell'editor di codice integrato.
4. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



## Creazione e aggiornamento di funzioni con file .zip utilizzando AWS CLI

È possibile utilizzare la [AWS CLI](#) per creare una nuova funzione o aggiornare una funzione esistente mediante un file .zip. Utilizza i comandi [create-function](#) e [update-function-code](#) per implementare il tuo pacchetto .zip. Se il file .zip ha dimensioni inferiori a 50 MB, è possibile caricare il pacchetto .zip da una posizione di file nella macchina di compilazione locale. Per i file di dimensioni maggiori, è necessario caricare il pacchetto .zip da un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il AWS CLI, consulta [Move objects](#) nella Guida per l'AWS CLI utente.

### Note

Se carichi il tuo file .zip da un bucket Amazon S3 utilizzando il AWS CLI, il bucket deve trovarsi nella stessa posizione della tua funzione. Regione AWS

Per creare una nuova funzione utilizzando un file .zip con AWS CLI, devi specificare quanto segue:

- Il nome della funzione (`--function-name`)
- Il runtime della tua funzione (`--runtime`)
- Il nome della risorsa Amazon (ARN) del [ruolo di esecuzione](#) della funzione (`--role`)
- Il nome del metodo del gestore nel codice della funzione (`--handler`)

È inoltre necessario specificare la posizione del file .zip. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda create-function --function-name myFunction \  
--runtime nodejs24.x --handler index.handler \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza l'opzione `--code` illustrata nel seguente comando di esempio. È necessario utilizzare il parametro `S3ObjectVersion` solo per gli oggetti con controllo delle versioni.

```
aws lambda create-function --function-name myFunction \  
--runtime nodejs24.x --handler index.handler \  
--code s3://my-bucket/my-function.zip
```

```
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--code S3Bucket=amzn-s3-demo-  
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Per aggiornare una funzione esistente mediante la CLI, specifica il nome della funzione utilizzando il parametro `--function-name`. È inoltre necessario specificare la posizione del file `.zip` che desideri utilizzare per aggiornare il codice della funzione. Se il file `.zip` si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda update-function-code --function-name myFunction \  
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file `.zip` in un bucket Amazon S3, utilizza le opzioni `--s3-bucket` e `--s3-key` come illustrato nel seguente comando di esempio. È necessario utilizzare il parametro `--s3-object-version` solo per gli oggetti con controllo delle versioni.

```
aws lambda update-function-code --function-name myFunction \  
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject  
Version
```

## Creazione e aggiornamento delle funzioni con file `.zip` utilizzando l'API Lambda

Per creare e aggiornare le funzioni mediante un archivio di file `.zip`, utilizza le seguenti operazioni API:

- [CreateFunction](#)
- [UpdateFunctionCode](#)

## Creazione e aggiornamento di funzioni con file `.zip` utilizzando AWS SAM

AWS Serverless Application Model (AWS SAM) è un toolkit che aiuta a semplificare il processo di creazione ed esecuzione di applicazioni serverless su AWS. Definisci le risorse per la tua applicazione in un modello YAML o JSON e usi l'interfaccia a riga di AWS SAM comando (AWS SAM CLI) per creare, impacchettare e distribuire le tue applicazioni. Quando crei una funzione Lambda da un AWS SAM modello, crea AWS SAM automaticamente un pacchetto di distribuzione `.zip` o un'immagine del contenitore con il codice della funzione e le dipendenze specificate. Per saperne di più sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda, consulta la [Guida introduttiva nella Developer Guide. AWS SAM](#) AWS Serverless Application Model

Puoi anche usare AWS SAM per creare una funzione Lambda utilizzando un archivio di file .zip esistente. Per creare una funzione Lambda utilizzando AWS SAM, puoi salvare il tuo file .zip in un bucket Amazon S3 o in una cartella locale sulla macchina di compilazione. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il AWS CLI, consulta [Move objects nella Guida per l'utente](#). AWS CLI

Nel AWS SAM modello, la `AWS::Serverless::Function` risorsa specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `CodeUri`- impostato sull'URI Amazon S3, sul percorso della cartella locale o sull'oggetto del codice funzione [FunctionCode](#)
- `Runtime`: imposta il runtime prescelto

Con AWS SAM, se il tuo file .zip è più grande di 50 MB, non devi prima caricarlo su un bucket Amazon S3. AWS SAM puoi caricare pacchetti .zip fino alla dimensione massima consentita di 250 MB (decompressi) da una posizione sul computer di compilazione locale.

Per saperne di più sulla distribuzione delle funzioni utilizzando il file .zip in AWS SAM, consulta [AWS::Serverless::Function nella Developer Guide](#). AWS SAM

Creazione e aggiornamento di funzioni con file .zip utilizzando CloudFormation

È possibile utilizzare CloudFormation per creare una funzione Lambda utilizzando un archivio di file .zip. Per creare una funzione Lambda da un file .zip, devi prima caricare il file su un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando AWS CLI, consulta [Move objects](#) nella Guida per l'utente. AWS CLI

Nel CloudFormation modello, la `AWS::Lambda::Function` risorsa specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `Code`: inserisci il nome del bucket Amazon S3 e il nome del file .zip nei campi `S3Bucket` e `S3Key`
- `Runtime`: imposta il runtime prescelto

Il file .zip CloudFormation generato non può superare i 4 MB. Per saperne di più sulla distribuzione delle funzioni utilizzando il file .zip in CloudFormation, vedi [AWS::Lambda: :Function](#) nella Guida per l'utente. CloudFormation



## Implementa funzioni Node.js Lambda con immagini di container

Esistono tre modi per creare un'immagine contenitore per una funzione Node.js Lambda:

- [Usando un AWS immagine base per Node.js](#)

[Le immagini di base AWS](#) sono precaricate con un runtime in linguaggio, un client di interfaccia di runtime per gestire l'interazione tra Lambda e il codice della funzione e un emulatore di interfaccia di runtime per i test locali.

- [Usando un AWS OS-only immagine di base](#)

[AWS OS-only le immagini di base](#) contengono una distribuzione Amazon Linux e l'emulatore [dell'interfaccia](#) di runtime. Queste immagini vengono comunemente utilizzate per creare immagini contenitore per linguaggi compilati, come [Go](#) e [Rust](#), e per una lingua o una versione per la quale Lambda non fornisce un'immagine di base, come 19. Node.js Puoi anche utilizzare immagini di OS-only base per implementare un runtime [personalizzato](#). Per rendere l'immagine compatibile con Lambda, è necessario includere Node.js [nell'immagine il client dell'interfaccia di](#) runtime for.

- [Usare un'immagine non di AWS base](#)

È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, è necessario includere Node.js [nell'immagine il client dell'interfaccia di](#) runtime per.

### Tip

Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta [Utilizzo di compilazioni a più fasi](#) nella documentazione Docker. Per creare immagini di container efficienti, segui le [best practice per scrivere file Docker](#).

Questa pagina spiega come creare, testare e implementare le immagini di container per Lambda.

### Argomenti

- [AWS immagini di base per Node.js](#)
- [Usando un AWS immagine base per Node.js](#)
- [Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime](#)

## AWS immagini di base per Node.js

AWS fornisce le seguenti immagini di base per Node.js:

| Tag | Runtime    | Sistema operativo | Dockerfile                                                               | Raggiunta obsolescenza |
|-----|------------|-------------------|--------------------------------------------------------------------------|------------------------|
| 26  | Node.js 26 | Amazon Linux 2023 | <a href="#">Dockerfile per 26 anni</a><br><a href="#">Node.js GitHub</a> | Non programmato        |
| 24  | Node.js 24 | Amazon Linux 2023 | <a href="#">Dockerfile per 24 anni</a><br><a href="#">Node.js GitHub</a> | 30 aprile 2028         |
| 22  | Node.js 22 | Amazon Linux 2023 | <a href="#">Dockerfile per 22 anni</a><br><a href="#">Node.js GitHub</a> | 30 aprile 2027         |

Repository Amazon ECR: [gallery.ecr. aws/lambda/node.js](https://gallery.ecr.aws/lambda/node.js)

Le Node.js 20 immagini di base e successive sono basate sull'immagine minima del contenitore di [Amazon Linux 2023](#). Le immagini di base precedenti utilizzano Amazon Linux 2. AL2023 offre diversi vantaggi rispetto ad Amazon Linux 2, tra cui un'impronta di implementazione ridotta e versioni aggiornate di librerie come `glibc`.

AL2023-based le immagini utilizzano `microdnf` (con collegamenti simbolici `dnf`) come gestore di pacchetti invece di `yum`, che è il gestore di pacchetti predefinito in Amazon Linux 2. `microdnf` è un'implementazione autonoma di `dnf`. Per un elenco dei pacchetti inclusi nelle AL2023-based immagini, consulta le colonne Minimal Container in [Confronto dei pacchetti installati su Amazon Linux 2023 Container Images](#). Per ulteriori informazioni sulle differenze tra AL2023 e Amazon Linux 2, consulta [Introduzione del runtime di Amazon Linux 2023 per AWS Lambda](#) nel blog AWS Compute.

### Note

Per eseguire AL2023-based le immagini localmente, incluso con AWS Serverless Application Model (AWS SAM), devi utilizzare la versione Docker 20.10.10 o successiva.

## Usando un AWS immagine base per Node.js

### Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [AWS CLI versione 2](#)
- [Docker](#) (versione minima 25.0.0)
- Il [plugin buildx](#) di Docker.
- Node.js

### Creazione di un'immagine da un'immagine di base

Per creare un'immagine contenitore da un AWS immagine base per Node.js

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
cd example
```

2. Crea un nuovo Node.js progetto con npm. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi Enter.

```
npm init
```

3. Crea un nuovo file denominato `index.js`. A fini di test, puoi utilizzare il codice della funzione di esempio seguente o sostituirlo con il tuo codice personalizzato.

#### Example Gestore CommonJS

```
exports.handler = async (event) => {
  const response = {
    statusCode: 200,
    body: JSON.stringify('Hello from Lambda!'),
  };
  return response;
};
```

4. Se la tua funzione dipende da librerie diverse da AWS SDK per JavaScript, usa [npm](#) per aggiungerle al tuo pacchetto.

## 5. Crea un nuovo Dockerfile con la seguente configurazione:

- Imposta la proprietà FROM sull'[URI dell'immagine di base](#).
- Usa il comando COPY per copiare il codice della funzione e le dipendenze di runtime in una `{LAMBDA_TASK_ROOT}` variabile di [Lambda-defined ambiente](#).
- Imposta l'argomento CMD specificando il gestore della funzione Lambda.

Nota che l'esempio Dockerfile non include un'[istruzione USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

### Example Dockerfile

```
FROM public.ecr.aws/lambda/nodejs:22

# Copy function code
COPY index.js ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler (could also be done as a parameter override outside
  of the Dockerfile)
CMD [ "index.handler" ]
```

6. Crea l'immagine Docker con il comando [docker build](#). L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il [tag](#) `test`. Per rendere l'immagine compatibile con Lambda, devi usare l'opzione `--provenance=false`.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
.
```

#### Note

Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Per creare una funzione Lambda utilizzando l'architettura del set di istruzioni ARM64, accertati di modificare il comando per utilizzare l'opzione `--platform linux/arm64`.

## (Facoltativo) Test dell'immagine in locale

1. Avvia l'immagine Docker con il comando `docker run`. In questo esempio, `docker-image` è il nome dell'immagine e `test` è il tag.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

### Note

Per creare l'immagine Docker per l'architettura del set di istruzioni ARM64, assicurati di utilizzare l'opzione `--platform linux/arm64` anziché `--platform linux/amd64`.

2. Da una nuova finestra di terminale, invia un evento all'endpoint locale.

### Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

### PowerShell

In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType "application/json"
```

3. Ottieni l'ID del container.

```
docker ps
```

4. Utilizza il comando [docker kill](#) per arrestare il container. In questo comando, sostituisci 3766c4ab331c con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

## Implementazione dell'immagine

### Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Per autenticare la CLI Docker nel registro Amazon ECR, esegui il comando [get-login-password](#).

- Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
- Sostituiscilo 111122223333 con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

 Note

Il repository Amazon ECR deve essere nello stesso della funzione Regione AWS Lambda.

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando [docker tag](#) per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:
  - `docker-image:test` è il nome e [tag](#) dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
  - Sostituisci l'<ECRrepositoryUri> con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando [docker push](#) per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Crea un ruolo di esecuzione](#) per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puoi creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta [Autorizzazioni multiaccount Amazon ECR](#).

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.



Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nel repository Amazon ECR e quindi utilizzare il comando [update-function-code](#) per implementare l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per implementare la nuova immagine nella stessa funzione Lambda, devi utilizzare il comando [update-function-code](#) anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \  
  --function-name hello-world \  
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --publish
```

Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

Se si utilizza un'immagine di [OS-only base](#) o un'immagine di base alternativa, è necessario includere il client dell'interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime estende l'[API Runtime](#), che gestisce l'interazione tra Lambda e il codice della funzione.

Installa il client dell'interfaccia [Node.js di runtime](#) utilizzando il gestore di pacchetti npm:

```
npm install aws-lambda-ric
```

Puoi anche scaricare il client dell'interfaccia [Node.js di runtime](#) da GitHub.

L'esempio seguente dimostra come creare un'immagine contenitore per l' Node.js utilizzo di un'immagine non di AWS base. Il Dockerfile di esempio utilizza l'immagine di base `bookworm`. Il Dockerfile include il client di interfaccia di runtime.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [AWS CLI versione 2](#)
- [Docker](#) (versione minima 25.0.0)

- Il [plugin buildx](#) di Docker.
- Node.js

## Creazione di un'immagine da un'immagine di base alternativa

Per creare un'immagine contenitore da un file non-AWS immagine di base

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
cd example
```

2. Crea un nuovo Node.js progetto con npm. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi Enter.

```
npm init
```

3. Crea un nuovo file denominato `index.js`. A fini di test, puoi utilizzare il codice della funzione di esempio seguente o sostituirlo con il tuo codice personalizzato.

### Example Gestore CommonJS

```
exports.handler = async (event) => {
  const response = {
    statusCode: 200,
    body: JSON.stringify('Hello from Lambda!'),
  };
  return response;
};
```

4. Crea un nuovo Dockerfile. Il seguente Dockerfile utilizza un'immagine di base bookworm anziché un'[immagine di base AWS](#). Il Dockerfile include il [client di interfaccia di runtime](#), che rende l'immagine compatibile con Lambda. Il seguente Dockerfile utilizza una [build multi-fase](#).

La prima fase crea un'immagine di build, che è un Node.js ambiente standard in cui sono installate le dipendenze della funzione. Nella seconda fase viene creata un'immagine più leggera con il codice della funzione e le relative dipendenze. Ciò riduce la dimensione finale dell'immagine.

- Imposta la proprietà FROM sull'identificativo dell'immagine di base.

- Utilizza il comando COPY per copiare il codice della funzione e le dipendenze di runtime.
- Imposta l'ENTRYPOINT sul modulo su cui desideri che il container Docker venga eseguito all'avvio. In questo caso, il modulo è il client di interfaccia di runtime.
- Imposta l'argomento CMD specificando il gestore della funzione Lambda.

Nota che l'esempio Dockerfile non include un'istruzione [USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente root quando non viene fornita alcuna istruzione USER.

### Example Dockerfile

```
# Define custom function directory
ARG FUNCTION_DIR="/function"

FROM node:20-bookworm as build-image

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Install build dependencies
RUN apt-get update && \
    apt-get install -y \
    g++ \
    make \
    cmake \
    unzip \
    libcurl4-openssl-dev

# Copy function code
RUN mkdir -p ${FUNCTION_DIR}
COPY . ${FUNCTION_DIR}

WORKDIR ${FUNCTION_DIR}

# Install Node.js dependencies
RUN npm install

# Install the runtime interface client
RUN npm install aws-lambda-ric
```

```
# Grab a fresh slim copy of the image to reduce the final size
FROM node:20-bookworm-slim

# Required for Node runtimes which use npm@8.6.0+ because
# by default npm writes logs under /home/.npm and Lambda fs is read-only
ENV NPM_CONFIG_CACHE=/tmp/.npm

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Set working directory to function root directory
WORKDIR ${FUNCTION_DIR}

# Copy in the built dependencies
COPY --from=build-image ${FUNCTION_DIR} ${FUNCTION_DIR}

# Set runtime interface client as default command for the container runtime
ENTRYPOINT ["/usr/local/bin/npx", "aws-lambda-ric"]
# Pass the name of the function handler as an argument to the runtime
CMD ["index.handler"]
```

5. Crea l'immagine Docker con il comando [docker build](#). L'esempio seguente assegna un nome all'immagine docker-image e le assegna il [tag](#) test. Per rendere l'immagine compatibile con Lambda, devi usare l'opzione `--provenance=false`.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test .
```

#### Note

Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Per creare una funzione Lambda utilizzando l'architettura del set di istruzioni ARM64, accertati di modificare il comando per utilizzare l'opzione `--platform linux/arm64`.

### (Facoltativo) Test dell'immagine in locale

Usa il [simulatore dell'interfaccia di runtime](#) per testare localmente l'immagine. Puoi [creare l'emulatore nella tua immagine](#) o seguire la procedura riportata e installarlo sul tuo computer locale.

## Installazione ed esecuzione dell'emulatore di interfaccia di runtime sul computer locale

1. Dalla directory del progetto, esegui il comando seguente per scaricare l'emulatore di interfaccia di runtime (architettura x86-64) GitHub e installarlo sul tuo computer locale.

### Linux/macOS

```
mkdir -p ~/.aws-lambda-rie && \
  curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-
runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
  chmod +x ~/.aws-lambda-rie/aws-lambda-rie
```

Per installare l'emulatore arm64, sostituite l'URL del GitHub repository nel comando precedente con il seguente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

### PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"
if (-not (Test-Path $dirPath)) {
    New-Item -Path $dirPath -ItemType Directory
}

$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/
releases/latest/download/aws-lambda-rie"
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Per installare l'emulatore arm64, sostituisci \$downloadLink con quanto segue:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Avvia l'immagine Docker con il comando `docker run`. Tenere presente quanto segue:
  - `docker-image` è il nome dell'immagine e `test` è il tag.
  - `/usr/local/bin/npx aws-lambda-ric index.handler` è l'ENTRYPOINT seguito dal CMD del Dockerfile.

## Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
9000:8080 \
  --entrypoint /aws-lambda/aws-lambda-rie \
  docker-image:test \
  /usr/local/bin/npx aws-lambda-ric index.handler
```

## PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
9000:8080 `
  --entrypoint /aws-lambda/aws-lambda-rie `
  docker-image:test `
  /usr/local/bin/npx aws-lambda-ric index.handler
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

### Note

Per creare l'immagine Docker per l'architettura del set di istruzioni ARM64, assicurati di utilizzare l'opzione `--platform linux/arm64` anziché `--platform linux/amd64`.

3. Pubblica un evento nell'endpoint locale.

## Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

## PowerShell

In PowerShell, esegui il seguente comando: Invoke-WebRequest

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{} ' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType  
"application/json"
```

### 4. Ottieni l'ID del container.

```
docker ps
```

### 5. Utilizza il comando [docker kill](#) per arrestare il container. In questo comando, sostituisci 3766c4ab331c con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

## Implementazione dell'immagine

### Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

#### 1. Per autenticare la CLI Docker nel registro Amazon ECR, esegui il comando [get-login-password](#).

- Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
- Sostituiscilo 111122223333 con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

#### Note

Il repository Amazon ECR deve essere nello stesso della funzione Regione AWS Lambda.

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando [docker tag](#) per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:



- `docker-image:test` è il nome e [tag](#) dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
- Sostituisci l'<ECRrepositoryUri> con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando [docker push](#) per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Crea un ruolo di esecuzione](#) per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puoi creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta [Autorizzazioni multiaccount Amazon ECR](#).

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nel repository Amazon ECR e quindi utilizzare il comando [update-function-code](#) per implementare l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per implementare la nuova immagine nella stessa funzione Lambda, devi utilizzare il comando [update-function-code](#) anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \  
  --function-name hello-world \  
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --publish
```

## Utilizzo dei livelli per le funzioni Lambda Node.js

Usa i [livelli Lambda](#) per impacchettare codice e dipendenze che desideri riutilizzare in più funzioni. I livelli di solito contengono dipendenze dalla libreria, un [runtime personalizzato](#) o file di configurazione. La creazione di un livello prevede tre passaggi generali:

1. Crea un pacchetto per il contenuto del livello. Ciò significa creare un archivio di file con estensione .zip che contiene le dipendenze che desideri utilizzare nelle funzioni.
2. Crea il livello in Lambda.
3. Aggiungi il livello alle tue funzioni.

### Argomenti

- [Crea un pacchetto per il contenuto del livello](#)
- [Crea il layer in Lambda](#)
- [Aggiungete il layer alla vostra funzione](#)
- [App di esempio](#)

### Crea un pacchetto per il contenuto del livello

Per creare un layer, raggruppa i pacchetti in un archivio di file.zip che soddisfi i seguenti requisiti:

- Create il layer utilizzando la stessa versione di Node.js che intendete utilizzare per la funzione Lambda. Ad esempio, se create il layer utilizzando Node.js 24, utilizzate il runtime Node.js 24 per la funzione.
- Il file.zip del livello deve utilizzare una di queste strutture di directory:
  - nodejs/node\_modules
  - nodejs/nodeX/node\_modules(Xdov'è la versione di Node.js, ad esempio node22)

Per ulteriori informazioni, consulta [Percorsi dei livelli per ciascun runtime Lambda](#).

- I pacchetti del livello devono essere compatibili con Linux. Le funzioni Lambda vengono eseguite su Amazon Linux.

Puoi creare livelli che contengono librerie Node.js di terze parti installate con npm (come `axios` o `lodash`) o JavaScript moduli personalizzati.

## Dipendenze da terze parti

Per creare un livello utilizzando i pacchetti npm

1. Crea la struttura di directory richiesta e installa i pacchetti direttamente al suo interno:

```
mkdir -p nodejs
npm install --prefix nodejs lodash axios
```

Questo comando installa i pacchetti direttamente nella `nodejs/node_modules` directory, che è la struttura richiesta da Lambda.

### Note

Per i pacchetti con dipendenze native o componenti binari (come sharp o bcrypt), assicurati che siano compatibili con l'ambiente Lambda Linux e l'architettura della tua funzione. Potrebbe essere necessario utilizzare il flag: `--platform`

```
npm install --prefix nodejs --platform=linux --arch=x64 sharp
```

Per dipendenze native più complesse, potrebbe essere necessario compilarle in un ambiente Linux che corrisponda al runtime Lambda. Puoi usare Docker per questo scopo.

2. Comprimi il contenuto del livello:

Linux/macOS

```
zip -r layer.zip nodejs/
```

PowerShell

```
Compress-Archive -Path .\nodejs -DestinationPath .\layer.zip
```

La struttura delle cartelle del file.zip dovrebbe essere simile alla seguente:

```
nodejs/
### package.json
```

```
### package-lock.json
### node_modules/
### lodash/
### axios/
### (dependencies of the other packages)
```

### Note

- Assicurati che il tuo file.zip includa la nodejs directory a livello principale al node\_modules suo interno. Questa struttura assicura che Lambda possa localizzare e importare i tuoi pacchetti.
- I file package.json e package-lock.json nella directory nodejs/ vengono utilizzati da npm per la gestione delle dipendenze ma non sono richiesti da Lambda per la funzionalità dei livelli. Ogni pacchetto installato contiene già il proprio package.json file che definisce il modo in cui Lambda importa il pacchetto.

## Moduli personalizzati JavaScript

Per creare un livello utilizzando il proprio codice

1. Crea la struttura di directory richiesta per il tuo livello:

```
mkdir -p nodejs/node_modules/validator
cd nodejs/node_modules/validator
```

2. Crea un package.json file per il tuo modulo personalizzato per definire come deve essere importato:

Example nodejs/node\_modules/validator/package.json

```
{
  "name": "validator",
  "version": "1.0.0",
  "type": "module",
  "main": "index.mjs"
}
```

3. Crea il tuo file del JavaScript modulo:

## Example nodejs/node\_modules/validator/index.mjs

```
export function validateOrder(orderData) {
  // Validates an order and returns formatted data
  const requiredFields = ['productId', 'quantity'];

  // Check required fields
  const missingFields = requiredFields.filter(field => !(field in orderData));
  if (missingFields.length > 0) {
    throw new Error(`Missing required fields: ${missingFields.join(', ')}`);
  }

  // Validate quantity
  const quantity = orderData.quantity;
  if (!Number.isInteger(quantity) || quantity < 1) {
    throw new Error('Quantity must be a positive integer');
  }

  // Format and return the validated data
  return {
    productId: String(orderData.productId),
    quantity: quantity,
    shippingPriority: orderData.priority || 'standard'
  };
}

export function formatResponse(statusCode, body) {
  // Formats the API response
  return {
    statusCode: statusCode,
    body: JSON.stringify(body)
  };
}
```

### 4. Comprimi il contenuto del livello:

#### Linux/macOS

```
zip -r layer.zip nodejs/
```

## PowerShell

```
Compress-Archive -Path .\nodejs -DestinationPath .\layer.zip
```

La struttura delle cartelle del file.zip dovrebbe essere simile alla seguente:

```
nodejs/  
### node_modules/  
    ### validator/  
        ### package.json  
        ### index.mjs
```

5. Nella tua funzione, importa e usa i moduli. Esempio:

```
import { validateOrder, formatResponse } from 'validator';  
  
export const handler = async (event) => {  
  try {  
    // Parse the order data from the event body  
    const orderData = JSON.parse(event.body || '{}');  
  
    // Validate and format the order  
    const validatedOrder = validateOrder(orderData);  
  
    return formatResponse(200, {  
      message: 'Order validated successfully',  
      order: validatedOrder  
    });  
  } catch (error) {  
    if (error instanceof Error && error.message.includes('Missing required  
fields')) {  
      return formatResponse(400, {  
        error: error.message  
      });  
    }  
  
    return formatResponse(500, {  
      error: 'Internal server error'  
    });  
  }  
}
```

```
};
```

È possibile utilizzare il seguente [evento di test](#) per richiamare la funzione:

```
{
  "body": "{\"productId\": \"ABC123\", \"quantity\": 2, \"priority\": \"express\"}"
}
```

Risposta prevista:

```
{
  "statusCode": 200,
  "body": "{\"message\": \"Order validated successfully\", \"order\": {\"productId\": \"ABC123\", \"quantity\": 2, \"shippingPriority\": \"express\"}}"
```

## Crea il layer in Lambda

È possibile pubblicare il layer utilizzando la console AWS CLI o la console Lambda.

### AWS CLI

Esegui il [publish-layer-version](#) AWS CLI comando per creare il livello Lambda:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip
--compatible-runtimes nodejs24.x
```

Il parametro di [runtime compatibile](#) è facoltativo. Quando specificato, Lambda utilizza questo parametro per filtrare i livelli nella console Lambda.

### Console

#### Creazione di un livello (console)

1. Apri la [pagina Layers](#) (Livelli) nella console Lambda.
2. Scegli Create layer (Crea livello).
3. Scegli Carica un file.zip, quindi carica l'archivio.zip che hai creato in precedenza.
4. (Facoltativo) Per i runtime compatibili, scegliete il runtime Node.js che corrisponde alla versione Node.js utilizzata per creare il layer.



## 5. Scegli Create (Crea).

Aggiungete il layer alla vostra funzione

### AWS CLI

Per collegare il layer alla tua funzione, esegui il [update-function-configuration](#) AWS CLI comando. Per il `--layers` parametro, utilizzate il livello ARN. L'ARN deve specificare la versione (ad esempio, `arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`). Per ulteriori informazioni, consulta [Livelli e versioni di livelli](#).

```
aws lambda update-function-configuration --function-name my-function --cli-binary-format raw-in-base64-out --layers "arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1"
```

L'`cli-binary-format` opzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

### Console

Aggiunta di un livello a una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione.
3. Scorri verso il basso fino alla sezione Livelli, quindi scegli Aggiungi un livello.
4. In Scegli un livello, seleziona Livelli personalizzati, quindi scegli il tuo livello.

#### Note

Se non hai aggiunto un [runtime compatibile](#) quando hai creato il layer, il tuo layer non verrà elencato qui. È possibile specificare invece il livello ARN.

5. Scegliere Aggiungi.

## App di esempio

Per altri esempi di utilizzo dei layer Lambda, consulta l'applicazione di esempio [layer-nodejs](#) nel repository Developer Guide. AWS Lambda GitHub [Questa applicazione include un livello che contiene la libreria lodash](#). Dopo aver creato il layer, è possibile implementare e richiamare la funzione corrispondente per confermare che il layer funzioni come previsto.

## Utilizzo dell'oggetto di contesto Lambda per recuperare le informazioni sulla funzione Node.js

Quando Lambda esegue la funzione, passa un oggetto Context al [gestore](#). Questo oggetto fornisce i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

### Metodi del contesto

- `getRemainingTimeInMillis()`: restituisce il numero di millisecondi rimasti prima del timeout dell'esecuzione.

### Proprietà del contesto

- `functionName`: il nome della funzione Lambda.
- `functionVersion`: la [versione](#) della funzione.
- `invokedFunctionArn`: l'Amazon Resource Name (ARN) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `memoryLimitInMB`: la quantità di memoria allocata per la funzione.
- `awsRequestId`: l'identificatore della richiesta di invocazione.
- `logGroupName`: il gruppo di log per la funzione.
- `logStreamName`: il flusso di log per l'istanza della funzione.
- `identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
  - `cognitoIdentityId`: l'identità autenticata di Amazon Cognito.
  - `cognitoIdentityPoolId`: il pool di identità Amazon Cognito che ha autorizzato l'invocazione.
- `clientContext`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`
  - `client.app_package_name`
  - `env.platform_version`

- `env.platform`
- `env.make`
- `env.model`
- `env.locale`
- `custom`: valori personalizzati impostati dall'applicazione mobile.
- `callbackWaitsForEmptyEventLoop`— Per impostazione predefinita (`true`), quando si utilizza un gestore di funzioni basato sul callback, Lambda attende che il ciclo di eventi sia vuoto dopo l'esecuzione del callback prima di terminare l'invoca della funzione. Imposta su `false` per inviare la risposta e terminare la chiamata immediatamente dopo l'esecuzione del callback anziché attendere che il ciclo di eventi sia vuoto. Gli eventi eccezionali continuano a verificarsi durante la chiamata successiva. Nota che Lambda supporta gestori di funzioni basati su callback solo per Node.js 22 e runtime precedenti.

La seguente funzione di esempio registra le informazioni di contesto e restituisce la posizione dei log.

Example File `index.js`

```
exports.handler = async function(event, context) {
  console.log('Remaining time: ', context.getRemainingTimeInMillis())
  console.log('Function name: ', context.functionName)
  return context.logStreamName
}
```

## Registra e monitora le funzioni Node.js Lambda

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon CloudWatch. La tua funzione Lambda include un gruppo di CloudWatch log e un flusso di log per ogni istanza della tua funzione. L'ambiente del runtime Lambda invia i dettagli su ogni richiamo al flusso di log e inoltra i log e l'output del codice della funzione. Per ulteriori informazioni, consulta [Invio dei log delle funzioni Lambda ai CloudWatch log](#).

Questa pagina descrive come produrre un output di log dal codice della funzione Lambda e accedere ai log utilizzando la AWS Command Line Interface console Lambda o la console CloudWatch.

### Sections

- [Creazione di una funzione che restituisce i registri](#)
- [Utilizzo dei controlli di registrazione avanzati Lambda con Node.js](#)
- [Visualizzazione dei log nella console Lambda](#)
- [Visualizzazione dei log nella console CloudWatch](#)
- [Visualizzazione dei log utilizzando AWS Command Line Interface \(AWS CLI\)](#)
- [Eliminazione dei log](#)

### Creazione di una funzione che restituisce i registri

Per generare i log dal codice della funzione, è possibile utilizzare i metodi dell'[oggetto console](#) o di qualsiasi libreria di registrazione che scriva su `stdout` o `stderr`. L'esempio seguente registra i valori delle variabili di ambiente e l'oggetto evento.

#### Note

Si consiglia di utilizzare tecniche come la convalida dell'input e la codifica dell'output durante la registrazione degli input. Se registri direttamente i dati di input, un utente malintenzionato potrebbe essere in grado di utilizzare il codice per rendere difficile l'individuazione delle manomissioni, falsificare le voci di log o aggirare i monitor dei log. Per ulteriori informazioni, consulta [Neutralizzazione di output non corretta per i log](#) in Enumerazione delle debolezze comuni.

## Example File index.js – Registrazione dei log

```
exports.handler = async function(event, context) {
  console.log("ENVIRONMENT VARIABLES\n" + JSON.stringify(process.env, null, 2))
  console.info("EVENT\n" + JSON.stringify(event, null, 2))
  console.warn("Event not processed.")
  return context.logStreamName
}
```

## Example Formato dei log

```
START RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac Version: $LATEST
2019-06-07T19:11:20.562Z c793869b-ee49-115b-a5b6-4fd21e8dedac INFO ENVIRONMENT
VARIABLES
{
  "AWS_LAMBDA_FUNCTION_VERSION": "$LATEST",
  "AWS_LAMBDA_LOG_GROUP_NAME": "/aws/lambda/my-function",
  "AWS_LAMBDA_LOG_STREAM_NAME": "2019/06/07/[$LATEST]e6f4a0c4241adcd70c262d34c0bbc85c",
  "AWS_EXECUTION_ENV": "AWS_Lambda_nodejs12.x",
  "AWS_LAMBDA_FUNCTION_NAME": "my-function",
  "PATH": "/var/lang/bin:/usr/local/bin:/usr/bin:/bin:/opt/bin",
  "NODE_PATH": "/opt/nodejs/node10/node_modules:/opt/nodejs/node_modules:/var/runtime/
node_modules",
  ...
}
2019-06-07T19:11:20.563Z c793869b-ee49-115b-a5b6-4fd21e8dedac INFO EVENT
{
  "key": "value"
}
2019-06-07T19:11:20.564Z c793869b-ee49-115b-a5b6-4fd21e8dedac WARN Event not processed.
END RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac
REPORT RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac Duration: 128.83 ms Billed
Duration: 296 ms Memory Size: 128 MB Max Memory Used: 74 MB Init Duration: 166.62 ms
XRAY TraceId: 1-5d9d007f-0a8c7fd02xmpl480aed55ef0 SegmentId: 3d752xmpl1bbe37e Sampled:
true
```

Il Node.js runtime registra le REPORT righe STARTEND, e per ogni chiamata. A ogni voce registrata dalla funzione aggiunge un timestamp, l'ID della richiesta e il livello di log. La riga del report fornisce i seguenti dettagli.

## Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per la chiamata.
- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.
- Quantità max utilizzata – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- Durata Init – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- XRAY TraceId: per le richieste tracciate, l'ID di traccia. [AWS X-Ray](#)
- SegmentId— Per le richieste tracciate, l'ID del X-Ray segmento.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

È possibile visualizzare i log nella console Lambda, nella console CloudWatch Logs o dalla riga di comando.

### Utilizzo dei controlli di registrazione avanzati Lambda con Node.js

Per darti un maggiore controllo su come i log delle tue funzioni vengono acquisiti, elaborati e consumati, puoi configurare le seguenti opzioni di registrazione per i runtime supportati: Node.js

- Formato di registro: seleziona tra testo normale e formato JSON strutturato per i log della tua funzione
- Livello di registro: per i log in formato JSON, scegli il livello di dettaglio dei log che Lambda invia ad Amazon CloudWatch, ad esempio ERROR, DEBUG o INFO
- Gruppo di log: scegli il gruppo di CloudWatch log a cui la tua funzione invia i log

Per ulteriori informazioni su queste opzioni di registrazione e istruzioni su come configurare la funzione per utilizzarle, consulta la pagina [the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”](#).

Per utilizzare le opzioni relative al formato e al livello di registro con le tue funzioni Node.js Lambda, consulta le linee guida nelle sezioni seguenti.

## Utilizzo di log JSON strutturati con Node.js

Se selezioni JSON come formato di registro della tua funzione, Lambda invia l'output dei log utilizzando i metodi della console `console.trace`, `console.debug`, `console.log`, `console.info`, `console.error`, e `console.warn` a CloudWatch come JSON strutturato. Ogni oggetto di log JSON contiene almeno quattro coppie chiave-valore con le seguenti chiavi:

- `"timestamp"`: l'ora in cui è stato generato il messaggio di log
- `"level"`: il livello di log assegnato al messaggio
- `"message"`: il contenuto del messaggio di log
- `"requestId"`: l'ID di richiesta univoco dell'invocazione alla funzione

A seconda del metodo di registrazione di log utilizzato dalla funzione, questo oggetto JSON può anche contenere coppie di chiavi aggiuntive. Ad esempio, se la funzione utilizza metodi della console per registrare i log degli oggetti di errore con più argomenti, l'oggetto JSON conterrà coppie chiave-valore aggiuntive con le chiavi `errorMessage`, `errorType` e `stackTrace`.

Se il codice utilizza già un'altra libreria di registrazione, come Powertools for AWS Lambda, per produrre log strutturati JSON, non è necessario apportare alcuna modifica. Lambda non codifica due volte i log già codificati in JSON, quindi i log delle applicazioni della tua funzione continuano a essere acquisiti come prima.

Per ulteriori informazioni sull'utilizzo del pacchetto Powertools for AWS Lambda logging per creare log strutturati JSON in fase di esecuzione, consulta Node.js [the section called “Registrazione dei log”](#)

## Esempi di output di log in formato JSON

Gli esempi seguenti mostrano come i vari output di log generati utilizzando i `console` metodi con argomenti singoli e multipli vengono acquisiti in CloudWatch Logs quando si imposta il formato di log della funzione su JSON.

Il primo esempio utilizza il metodo `console.error` per generare una stringa semplice.

### Example Node.js codice di registrazione

```
export const handler = async (event) => {
  console.error("This is a warning message");
  ...
}
```



## Example Record di log JSON

```
{
  "timestamp": "2025-11-01T00:21:51.358Z",
  "level": "ERROR",
  "message": "This is a warning message",
  "requestId": "93f25699-2cbf-4976-8f94-336a0aa98c6f"
}
```

Con i metodi della `console`, puoi anche generare messaggi di log strutturati più complessi utilizzando argomenti singoli o multipli. Nel prossimo esempio, viene utilizzato `console.log` per generare due coppie chiave-valore con un singolo argomento. Nota che il `"message"` campo nell'oggetto JSON che Lambda invia a CloudWatch Logs non è stringato.

## Example Node.js codice di registrazione

```
export const handler = async (event) => {
  console.log({data: 12.3, flag: false});
  ...
}
```

## Example Record di log JSON

```
{
  "timestamp": "2025-12-08T23:21:04.664Z",
  "level": "INFO",
  "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
  "message": {
    "data": 12.3,
    "flag": false
  }
}
```

Nel prossimo esempio, viene utilizzato ancora una volta il metodo `console.log` per creare un output di log. Nella fattispecie, il metodo richiede due argomenti, una mappa contenente due coppie chiave-valore e una stringa identificativa. Tieni presente che in questo caso, poiché hai fornito due argomenti, Lambda rende il campo `"message"` in formato stringa.

## Example Node.js codice di registrazione

```
export const handler = async (event) => {
```

```
console.log('Some object - ', {data: 12.3, flag: false});
...
}
```

### Example Record di log JSON

```
{
  "timestamp": "2025-12-08T23:21:04.664Z",
  "level": "INFO",
  "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
  "message": "Some object -  { data: 12.3, flag: false }"
}
```

Lambda assegna gli output generati utilizzando il livello di log INFO della console.log.

L'ultimo esempio mostra come gli oggetti di errore possono essere inviati a CloudWatch Logs utilizzando i console metodi. Tieni presente che quando esegui il log di oggetti di errore utilizzando più argomenti, Lambda aggiunge i campi `errorMessage`, `errorType` e `stackTrace` all'output del log.

### Example Node.js codice di registrazione

```
export const handler = async (event) => {
  let e1 = new ReferenceError("some reference error");
  let e2 = new SyntaxError("some syntax error");
  console.log(e1);
  console.log("errors logged - ", e1, e2);
};
```

### Example Record di log JSON

```
{
  "timestamp": "2025-12-08T23:21:04.632Z",
  "level": "INFO",
  "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
  "message": {
    "errorType": "ReferenceError",
    "errorMessage": "some reference error",
    "stackTrace": [
      "ReferenceError: some reference error",
      "    at Runtime.handler (file:///var/task/index.mjs:3:12)",
    ]
  }
}
```

```

        "    at Runtime.handleOnceNonStreaming (file:///var/runtime/
index.mjs:1173:29)"
      ]
    }
  }
}

{
  "timestamp": "2025-12-08T23:21:04.646Z",
  "level": "INFO",
  "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
  "message": "errors logged - ReferenceError: some reference error
\n    at Runtime.handler (file:///var/task/index.mjs:3:12)\n    at
Runtime.handleOnceNonStreaming
(file:///var/runtime/index.mjs:1173:29) SyntaxError: some syntax
error\n    at Runtime.handler (file:///var/task/index.mjs:4:12)\n    at
Runtime.handleOnceNonStreaming
(file:///var/runtime/index.mjs:1173:29)",
  "errorType": "ReferenceError",
  "errorMessage": "some reference error",
  "stackTrace": [
    "ReferenceError: some reference error",
    "    at Runtime.handler (file:///var/task/index.mjs:3:12)",
    "    at Runtime.handleOnceNonStreaming (file:///var/runtime/index.mjs:1173:29)"
  ]
}

```

Quando si registrano i log di più tipi di errore, i campi aggiuntivi `errorMessage`, `errorType` e `stackTrace` vengono estratti dal primo tipo di errore fornito al metodo della console.

Utilizzo di librerie client formato del parametro incorporato (EMF) con log JSON strutturati

AWS fornisce librerie client open source Node.js che è possibile utilizzare per creare log EMF ([Embedded Metric Format](#)). Se disponi di funzioni esistenti che utilizzano queste librerie e modifichi il formato di registro della funzione in JSON, CloudWatch potresti non riconoscere più le metriche emesse dal codice.

Se il codice attualmente emette log EMF direttamente utilizzando `console.log` o utilizzando Powertools for AWS Lambda (TypeScript), non CloudWatch è inoltre in grado di analizzarli se si modifica il formato di registro della funzione in JSON.

### ⚠ Important

Per garantire che i log EMF delle tue funzioni continuino a essere analizzati correttamente CloudWatch, aggiorna le librerie <https://www.npmjs.com/package/aws-embedded-metrics> EMF e Powertools for alle versioni più recenti. [AWS Lambda](#) Se passi al formato di log JSON, ti consigliamo di eseguire dei test anche per garantire la compatibilità con i parametri incorporati della tua funzione. Se il tuo codice emette i log EMF direttamente utilizzando `console.log`, modifica il codice in modo che emetta tali parametri direttamente a `stdout`, come mostrato nel seguente esempio di codice.

Example codice che emette metriche incorporate su stdout

```
process.stdout.write(JSON.stringify(
  {
    "_aws": {
      "Timestamp": Date.now(),
      "CloudWatchMetrics": [{
        "Namespace": "lambda-function-metrics",
        "Dimensions": [["functionVersion"]],
        "Metrics": [{
          "Name": "time",
          "Unit": "Milliseconds",
          "StorageResolution": 60
        }]
      }]
    },
    "functionVersion": "$LATEST",
    "time": 100,
    "requestId": context.awsRequestId
  }
) + "\n")
```

Utilizzo del filtro a livello di registro con Node.js

Per AWS Lambda filtrare i log delle applicazioni in base al relativo livello di registro, la funzione deve utilizzare registri in formato JSON. Puoi farlo in due modi:

- Crea output di log utilizzando i metodi standard della console e configura la tua funzione per utilizzare la formattazione dei log JSON. AWS Lambda quindi filtra gli output del log utilizzando la coppia chiave-valore «level» nell'oggetto JSON descritto in. [the section called “Utilizzo di log JSON](#)

[strutturati con Node.js](#)” Per informazioni su come configurare il formato di registro della funzione, consulta. [the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”](#)

- Utilizzate un'altra libreria o metodo di registrazione per creare log strutturati JSON nel codice che includono una coppia chiave-valore «level» che definisce il livello dell'output del log. Ad esempio, puoi utilizzare Powertools per generare output AWS Lambda di log strutturati JSON dal tuo codice. [the section called “Registrazione dei log”](#) Per saperne di più sull'uso di Powertools con il runtime, consulta. Node.js

Per consentire a Lambda di filtrare i log della funzione, è necessario includere anche una coppia chiave-valore "timestamp" nell'output log JSON. L'ora deve essere specificata in un formato di timestamp [RFC 3339](#) valido. Se non fornisci un timestamp valido, Lambda assegna al log il livello INFO e aggiunge un timestamp per te.

Quando configuri la tua funzione per utilizzare il filtro a livello di log, selezioni il livello di log che desideri AWS Lambda inviare a Logs dalle seguenti opzioni: CloudWatch

| Livello di log            | Utilizzo standard                                                                               |
|---------------------------|-------------------------------------------------------------------------------------------------|
| TRACE (dettaglio massimo) | Le informazioni più dettagliate utilizzate per tracciare il percorso di esecuzione del codice   |
| DEBUG                     | Informazioni dettagliate per il debug del sistema                                               |
| INFO                      | Messaggi che registrano il normale funzionamento della funzione                                 |
| WARN                      | Messaggi su potenziali errori che potrebbero portare a comportamenti imprevisti se non risolti  |
| ERRORE                    | Messaggi relativi a problemi che impediscono al codice di funzionare come previsto              |
| FATAL (dettaglio minimo)  | Messaggi relativi a errori gravi che causano l'interruzione del funzionamento dell'applicazione |

Lambda invia i log del livello selezionato e inferiore a. CloudWatch Ad esempio, se si configura un livello di registro WARN, Lambda invia i log corrispondenti ai livelli WARN, ERROR e FATAL.

### Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor di codice incorporato, i log vengono visualizzati nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, trovi l'output del registro nella sezione Dettagli.

### Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Aprire la pagina dei gruppi di [log](#) sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (/aws/lambda/***your-function-name***).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'[istanza della funzione](#). Un flusso di log viene visualizzato quando si aggiorna la funzione Lambda e quando vengono create istanze aggiuntive per gestire chiamate simultanee. Per trovare i log per una chiamata specifica, ti consigliamo di strumentare la tua funzione con. AWS X-Ray X-Ray registra i dettagli sulla richiesta e sul flusso di log nella traccia.

### Visualizzazione dei log utilizzando AWS Command Line Interface (AWS CLI)

AWS CLI È uno strumento open source che puoi utilizzare per interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della [AWS CLI versione 2](#).

È possibile utilizzare [AWS CLI](#) per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

## Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "LogResult":
    "U1RBULQgUmVxdWVzdElk0iA4N2QwNDRI0C1mMTU0LTExZTgt0GNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
  "ExecutedVersion": "$LATEST"
}
```

## Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'base64 utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'`cli-binary-format` opzione è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ22luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
Duration: 80 ms      Memory Size: 128 MB      Max Memory Used: 73 MB
```

L'utilità `base64` è disponibile su Linux, macOS e [Ubuntu su Windows](#). Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

## Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza sed per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'opzione `cli-binary-format` è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"/'/g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

## Example(solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il comando seguente per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

## Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
```



```

    "ExecutedVersion": "$LATEST"
  }
  {
    "events": [
      {
        "timestamp": 1559763003171,
        "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
        "ingestionTime": 1559763003309
      },
      {
        "timestamp": 1559763003173,
        "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\n$LATEST\n",
\r ...",
        "ingestionTime": 1559763018353
      },
      {
        "timestamp": 1559763003173,
        "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
        "ingestionTime": 1559763018353
      },
      {
        "timestamp": 1559763003218,
        "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
        "ingestionTime": 1559763018353
      },
      {
        "timestamp": 1559763003218,
        "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
MB\t\n",
        "ingestionTime": 1559763018353
      }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
  }
}

```

## Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o [configurare un periodo di conservazione](#) trascorso il quale i log vengono eliminati automaticamente.

## Codice di strumentazione in Node.js AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzarlo X-Ray per tracciare una richiesta mentre attraversa le risorse dell'applicazione, che possono includere funzioni Lambda e altri servizi. AWS

Per inviare dati di tracciamento a X-Ray, puoi utilizzare una delle due librerie SDK:

- [AWS Distro for OpenTelemetry \(ADOT\)](#): una distribuzione sicura, pronta per la produzione e AWS supportata dell'SDK (OtEL). OpenTelemetry
- [SDK AWS X-Ray per Node.js](#)— Un SDK per la generazione e l'invio di dati di traccia a. X-Ray

Ciascuno degli SDK offre modi per inviare i dati di telemetria al servizio. X-Ray Puoi quindi utilizzarlo X-Ray per visualizzare, filtrare e ottenere informazioni dettagliate sulle metriche delle prestazioni dell'applicazione per identificare problemi e opportunità di ottimizzazione.

### Important

The X-Ray e Powertools for AWS Lambda SDK fanno parte di una soluzione di strumentazione strettamente integrata offerta da. AWS I livelli Lambda ADOT fanno parte di uno standard di settore per la strumentazione di tracciamento che in generale raccoglie più dati, ma potrebbero non essere adatti a tutti i casi d'uso. È possibile implementare il tracciamento end-to-end utilizzando entrambe le soluzioni. X-Ray Per saperne di più sulla scelta tra di esse, consulta [Scelta tra AWS Distro for Open Telemetry e SDK. X-Ray](#)

## Sections

- [Usare ADOT per strumentare le tue funzioni Node.js](#)
- [Usare l' X-Ray SDK per strumentare le tue funzioni Node.js](#)
- [Attivazione del tracciamento con la console Lambda](#)
- [Attivazione del tracciamento con l'API Lambda](#)
- [Attivazione del tracciamento con CloudFormation](#)
- [Interpretazione e tracciamento X-Ray](#)
- [Archiviazione delle dipendenze di runtime in un livello \(SDK\) X-Ray](#)

## Usare ADOT per strumentare le tue funzioni Node.js

ADOT fornisce [livelli](#) Lambda completamente gestiti che mettono insieme tutto il necessario per raccogliere i dati di telemetria utilizzando l'SDK OTel. Usando questo livello, è possibile strumentare le funzioni Lambda senza dover modificare alcun codice funzione. È inoltre possibile configurare il livello per eseguire l'inizializzazione personalizzata di OTel. Per ulteriori informazioni, consulta la sezione relativa alla [configurazione personalizzata per ADOT Collector su Lambda](#) nella documentazione di ADOT.

Per i Node.js runtime, puoi aggiungere il layer Lambda AWS gestito per ADOT Javascript per strumentare automaticamente le tue funzioni. Per istruzioni dettagliate su come aggiungere questo layer, consulta [AWS Distro for OpenTelemetry Lambda Support for JavaScript](#) nella documentazione ADOT.

## Usare l' X-Ray SDK per strumentare le tue funzioni Node.js

Per registrare i dettagli sulle chiamate effettuate dalla funzione Lambda ad altre risorse nell'applicazione, è anche possibile utilizzare il SDK AWS X-Ray per Node.js. Per ottenere l'SDK, aggiungere il pacchetto `aws-xray-sdk-core` alle dipendenze dell'applicazione.

Example [blank- .json nodejs/package](#)

```
{
  "name": "blank-nodejs",
  "version": "1.0.0",
  "private": true,
  "devDependencies": {
    "jest": "29.7.0"
  },
  "dependencies": {
    "@aws-sdk/client-lambda": "3.345.0",
    "aws-xray-sdk-core": "3.5.3"
  },
  "scripts": {
    "test": "jest"
  }
}
```

Per configurare i client AWS SDK nella [AWS SDK per JavaScript v3](#), racchiudi l'istanza del client con il metodo. `captureAWSv3Client`

## Example [vuoto- /index.js nodejs/function](#) — Tracciamento di un AWS Client SDK

```
const AWSXRay = require('aws-xray-sdk-core');
const { LambdaClient, GetAccountSettingsCommand } = require('@aws-sdk/client-lambda');

// Create client outside of handler to reuse
const lambda = AWSXRay.captureAWSv3Client(new LambdaClient());

// Handler
exports.handler = async function(event, context) {
  event.Records.forEach(record => {
    ...
```

Il runtime Lambda imposta alcune variabili di ambiente per configurare l' X-Ray SDK. Ad esempio, Lambda imposta `AWS_XRAY_CONTEXT_MISSING` su `LOG_ERROR` per evitare di generare errori di runtime dall'SDK. X-Ray Per impostare una strategia mancante di contesto personalizzato, sovrascrivi la variabile di ambiente nella configurazione della funzione in modo da non avere alcun valore, quindi puoi impostare la strategia mancante di contesto a livello di programmazione.

### Example Esempio di codice di inizializzazione

```
const AWSXRay = require('aws-xray-sdk-core');

// Configure the context missing strategy to do nothing
AWSXRay.setContextMissingStrategy(() => {});
```

Per ulteriori informazioni, consulta [the section called “Variabili di ambiente”](#).

Dopo aver aggiunto le dipendenze corrette e apportato le modifiche al codice necessarie, attiva il tracciamento nella configurazione della funzione tramite la console Lambda o l'API.

### Attivazione del tracciamento con la console Lambda

Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.

3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. In Strumenti di monitoraggio aggiuntivi scegli Modifica.
5. In CloudWatch Application Signals and AWS X-Ray, scegli Abilita per le tracce del servizio Lambda.
6. Scegli Save (Salva).

### Attivazione del tracciamento con l'API Lambda

Configura il tracciamento sulla tua funzione Lambda con AWS CLI or AWS SDK, utilizza le seguenti operazioni API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

Il seguente AWS CLI comando di esempio abilita la traccia attiva su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

### Attivazione del tracciamento con CloudFormation

Per attivare la traccia su una `AWS::Lambda::Function` risorsa in un CloudFormation modello, utilizzate la proprietà `TracingConfig`.

Example [function-inline.yml](#) — Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
```

...

Per una `AWS::Serverless::Function` risorsa AWS Serverless Application Model (AWS SAM), usa la `Tracing` proprietà.

Example [template.yml](#) — Configurazione di tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

## Interpretazione e tracciamento X-Ray

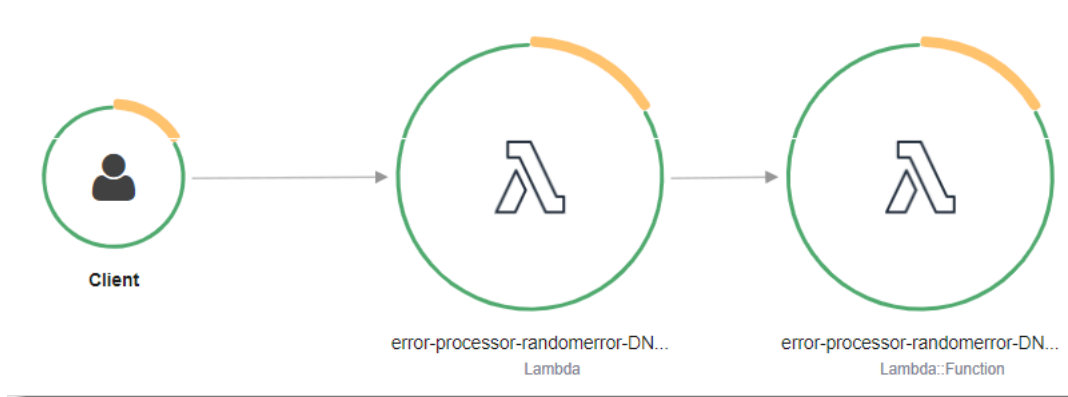
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al [ruolo di esecuzione](#) della funzione. Altrimenti, aggiungi la [AWSXRayDaemonWriteAccess](#) policy al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico del [X-Ray servizio](#) mostra le informazioni sull'applicazione e su tutti i suoi componenti. Il seguente esempio mostra un'applicazione con due funzioni. La funzione principale elabora gli eventi e talvolta restituisce errori. La seconda funzione in alto elabora gli errori che appaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon CloudWatch Logs.

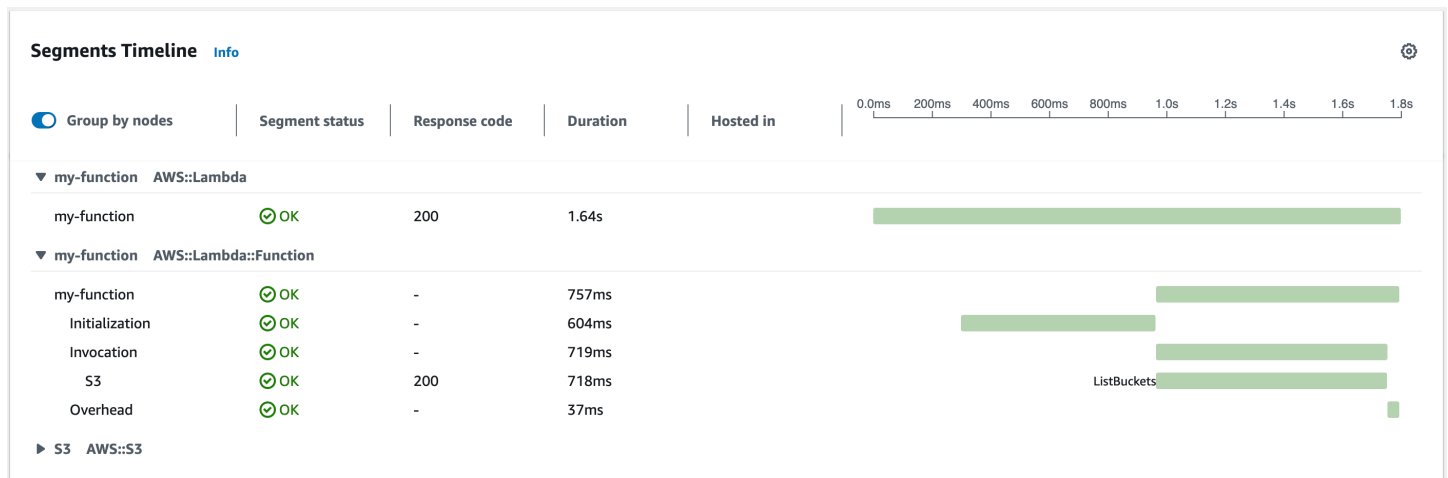


X-Ray non traccia tutte le richieste all'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento sia efficiente, fornendo comunque un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non puoi configurare la frequenza di X-Ray campionamento per le tue funzioni.

In X-Ray, una traccia registra le informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.



### Note

AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti. Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regioni, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia nel nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- Inizializzazione – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il [codice di inizializzazione](#). Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.
- Chiamata: rappresenta il tempo impiegato per eseguire il codice del gestore.
- Overhead: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla struttura dei segmenti di funzioni vecchio e nuovo stile, consulta [the section called “Comprensione delle tracce X-Ray”](#).

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta [SDK AWS X-Ray per Node.js](#) nella Guida per gli sviluppatori di AWS X-Ray .

### Prezzi

Puoi utilizzare il X-Ray tracciamento gratuitamente ogni mese fino a un determinato limite come parte del piano gratuito. AWS Oltre tale soglia, vengono X-Ray addebitati i costi per l'archiviazione e il recupero delle tracce. Per ulteriori informazioni, consultare [Prezzi di AWS X-Ray](#).

## Archiviazione delle dipendenze di runtime in un livello (SDK) X-Ray

Se si utilizza l' X-Ray SDK per configurare il codice funzione dei client AWS SDK, il pacchetto di distribuzione può diventare piuttosto ingombrante. Per evitare di caricare le dipendenze di runtime ogni volta che aggiorni il codice funzione, impacchetta l' X-Ray SDK in un layer Lambda. ???

L'esempio seguente mostra una risorsa `AWS::Serverless::LayerVersion` che memorizza SDK AWS X-Ray per Node.js.

Example [template.yml](#) — Livello delle dipendenze

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/.
      Tracing: Active
      Layers:
        - !Ref libs
        ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-nodejs-lib
      Description: Dependencies for the blank sample app.
      ContentUri: lib/.
      CompatibleRuntimes:
        - nodejs24.x
```

Con questa configurazione, si aggiorna il livello della libreria solo se si modificano le dipendenze di runtime. Poiché il pacchetto di implementazione della funzione contiene solo il codice, questo può contribuire a ridurre i tempi di caricamento.

La creazione di un livello per le dipendenze richiede modifiche alla compilazione per generare l'archivio dei livelli prima della distribuzione. Per un esempio funzionante, vedere l'applicazione di esempio [blank-nodejs](#) .

## Creazione di funzioni Lambda con TypeScript

È possibile utilizzare il Node.js runtime per eseguire il TypeScript codice in AWS Lambda. Poiché Node.js non esegue il TypeScript codice in modo nativo, devi prima trasporre il TypeScript codice in JavaScript. Quindi, usa JavaScript i file per distribuire il codice della funzione su Lambda. Il codice

viene eseguito in un ambiente che include l' AWS SDK per JavaScript, con le credenziali di un ruolo AWS Identity and Access Management (IAM) che gestisci. Per saperne di più sulle versioni SDK incluse nei Node.js runtime, consulta. [the section called “Runtime-included Versions SDK”](#)

Per informazioni sulle TypeScript funzioni di tracciamento con, vedere. [the section called “Tracciamento”](#)

Lambda supporta i seguenti Node.js runtime.

| Nome       | Identificatore | Sistema operativo | Data di ritiro  | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|------------|----------------|-------------------|-----------------|---------------------------|-------------------------------|
| Node.js 26 | nodejs26.x     | Amazon Linux 2023 | Non programmato | Non programmato           | Non programmato               |
| Node.js 24 | nodejs24.x     | Amazon Linux 2023 | 30 aprile 2028  | 1 giugno 2028             | 1 luglio 2028                 |
| Node.js 22 | nodejs22.x     | Amazon Linux 2023 | 30 aprile 2027  | 1 giugno 2027             | 1 luglio 2027                 |

## Argomenti

- [Impostazione di un TypeScript ambiente di sviluppo](#)
- [Definizioni dei tipi per Lambda](#)
- [Definire il gestore di funzioni Lambda in TypeScript](#)
- [Distribuisci TypeScript codice transpilato in Lambda con archivi di file .zip](#)
- [Implementa codice trascritto TypeScript in Lambda con immagini di container](#)
- [Utilizzo dell'oggetto contestuale Lambda per recuperare informazioni TypeScript sulla funzione](#)
- [Registra e monitora le funzioni TypeScript Lambda](#)
- [TypeScript Codice di tracciamento in AWS Lambda](#)

## Impostazione di un TypeScript ambiente di sviluppo

Utilizzate un ambiente di sviluppo integrato locale (IDE) o un editor di testo per scrivere il codice TypeScript della funzione. Non puoi creare TypeScript codice sulla console Lambda.

Puoi usare [esbuild](#) o il TypeScript compiler (`tsc`) di Microsoft per trascrivere il tuo codice in. TypeScript JavaScript [AWS Serverless Application Model \(AWS SAM\)](#) e [Kit di sviluppo per il cloud AWS \(AWS CDK\)](#) usano entrambi esbuild.

Durante l'utilizzo di esbuild, considera quanto segue:

- Ci sono diversi avvertimenti. [TypeScript](#)
- È necessario configurare le impostazioni di TypeScript traspilazione in modo che corrispondano al Node.js runtime che si intende utilizzare. Per ulteriori informazioni, consulta [Target](#) nella documentazione esbuild. Per un esempio di file `tsconfig.json` che dimostra come scegliere come target una Node.js versione specifica supportata da Lambda, consulta il repository. [TypeScript GitHub](#)
- esbuild non esegue controlli di tipo. Per controllare i tipi, utilizza il compilatore `tsc`. Esegui `tsc -noEmit` oppure aggiungi un parametro `"noEmit"` al file `tsconfig.json` come mostrato nell'esempio seguente. Questo configura `tsc` per non emettere JavaScript file. Dopo aver controllato i tipi, usa esbuild per convertire i TypeScript file in. JavaScript

Example `tsconfig.json`

```
{
  "compilerOptions": {
    "target": "es2020",
    "strict": true,
    "preserveConstEnums": true,
    "noEmit": true,
    "sourceMap": false,
    "module": "commonjs",
    "moduleResolution": "node",
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "isolatedModules": true,
  },
  "exclude": ["node_modules", "**/*.test.ts"]
}
```

## Definizioni dei tipi per Lambda

Il [pacchetto](#) `@types/aws-lambda` fornisce le definizioni dei tipi per le funzioni Lambda. Installa questo pacchetto quando la tua funzione utilizza uno dei seguenti elementi:

- Fonti di AWS eventi comuni, come:
  - APIGatewayProxyEvent: per [integrazioni proxy Amazon API Gateway](#)
  - SNSEvent: per [notifiche Amazon Simple Notification Service](#)
  - SQSEvent: per [messaggi Amazon Simple Queue Service](#)
  - S3Event: per [eventi di trigger S3](#)
  - DynamoDBStreamEvent: per [flussi Amazon DynamoDB](#)
- L'oggetto [contestuale](#) Lambda
- Il modello di gestore di [callback](#)

Per aggiungere le definizioni dei tipi Lambda alla tua funzione, installa `@types/aws-lambda` come dipendenza di sviluppo:

```
npm install -D @types/aws-lambda
```

Quindi, importa i tipi da `aws-lambda`:

```
import { Context, S3Event, APIGatewayProxyEvent } from 'aws-lambda';

export const handler = async (event: S3Event, context: Context) => {
  // Function code
};
```

L'istruzione `import ... from 'aws-lambda'` importa le definizioni dei tipi. Non importa il pacchetto npm `aws-lambda`, che è uno strumento di terzi non correlato. Per ulteriori informazioni, consulta [aws-lambda](#) nel repository. DefinitelyTyped GitHub

#### Note

Non hai bisogno di [@types/aws-lambda](#) quando usi le tue definizioni di tipo personalizzate. Per una funzione di esempio che definisce il proprio tipo per un oggetto evento, vedi [Esempio di codice della TypeScript funzione Lambda](#).

## Definire il gestore di funzioni Lambda in TypeScript

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come lavorare con i gestori di funzioni Lambda in TypeScript, incluse le opzioni per la configurazione del progetto, le convenzioni di denominazione e le best practice. Questa pagina include anche un esempio di funzione TypeScript Lambda che acquisisce informazioni su un ordine, produce una ricevuta in un file di testo e lo inserisce in un bucket Amazon Simple Storage Service (Amazon S3). Per informazioni su come distribuire una funzione dopo averla scritta, consulta o. [the section called “Implementare archivi di file .zip”](#) [the section called “Implementazione di immagini di container”](#)

### Argomenti

- [Configurazione del progetto TypeScript](#)
- [Esempio di codice della TypeScript funzione Lambda](#)
- [Moduli CommonJS ed ES](#)
- [Node.js inizializzazione](#)
- [Convenzioni di denominazione dei gestori](#)
- [Definizione e accesso all'oggetto evento di input](#)
- [Schemi di gestione validi per le funzioni TypeScript](#)
- [Utilizzo dell'SDK per la JavaScript v3 nel gestore](#)
- [Accesso alle variabili d'ambiente](#)
- [Utilizzo dello stato globale](#)
- [Procedure consigliate per il codice per le funzioni Lambda TypeScript](#)

### Configurazione del progetto TypeScript

Utilizzate un ambiente di sviluppo integrato locale (IDE) o un editor di testo per scrivere il codice TypeScript della funzione. Non puoi creare TypeScript codice sulla console Lambda.

Esistono diversi modi per inizializzare un progetto TypeScript Lambda. Ad esempio, puoi creare un progetto utilizzandonpm, creare un'[AWS SAM applicazione](#) o creare un'[AWS CDK applicazione](#). Per creare il progetto utilizzandonpm:

```
npm init
```

Il codice della funzione risiede in un `.ts` file, che viene convertito in un JavaScript file in fase di compilazione. Puoi usare [esbuild](#) o il TypeScript compiler (`tsc`) di Microsoft per trasporre il codice in TypeScript JavaScript. Per usare esbuild, aggiungilo come dipendenza di sviluppo:

```
npm install -D esbuild
```

Un tipico progetto di funzione TypeScript Lambda segue questa struttura generale:

```
/project-root
  ### index.ts - Contains main handler
  ### dist/ - Contains compiled JavaScript
  ### package.json - Project metadata and dependencies
  ### package-lock.json - Dependency lock file
  ### tsconfig.json - TypeScript configuration
  ### node_modules/ - Installed dependencies
```

## Esempio di codice della TypeScript funzione Lambda

Il seguente codice di funzione Lambda di esempio acquisisce informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3. Questo esempio definisce un tipo di evento personalizzato (`OrderEvent`). Per informazioni su come importare le definizioni dei tipi per le sorgenti di AWS eventi, consulta Definizioni dei [tipi per Lambda](#).

### Note

Questo esempio utilizza un gestore di moduli ES. Lambda supporta sia il modulo ES che i gestori CommonJS. Per ulteriori informazioni, consulta [Moduli CommonJS ed ES](#).

## Example index.ts, funzione Lambda

```
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';

// Initialize the S3 client outside the handler for reuse
const s3Client = new S3Client();

// Define the shape of the input event
type OrderEvent = {
```

```

    order_id: string;
    amount: number;
    item: string;
  }

/**
 * Lambda handler for processing orders and storing receipts in S3.
 */
export const handler = async (event: OrderEvent): Promise<string> => {
  try {
    // Access environment variables
    const bucketName = process.env.RECEIPT_BUCKET;
    if (!bucketName) {
      throw new Error('RECEIPT_BUCKET environment variable is not set');
    }

    // Create the receipt content and key destination
    const receiptContent = `OrderID: ${event.order_id}\nAmount: $
${event.amount.toFixed(2)}\nItem: ${event.item}`;
    const key = `receipts/${event.order_id}.txt`;

    // Upload the receipt to S3
    await uploadReceiptToS3(bucketName, key, receiptContent);

    console.log(`Successfully processed order ${event.order_id} and stored receipt
in S3 bucket ${bucketName}`);
    return 'Success';
  } catch (error) {
    console.error(`Failed to process order: ${error instanceof Error ?
error.message : 'Unknown error'}`);
    throw error;
  }
};

/**
 * Helper function to upload receipt to S3
 */
async function uploadReceiptToS3(bucketName: string, key: string, receiptContent:
string): Promise<void> {
  try {
    const command = new PutObjectCommand({
      Bucket: bucketName,
      Key: key,
      Body: receiptContent

```



```

    });

    await s3Client.send(command);
  } catch (error) {
    throw new Error(`Failed to upload receipt to S3: ${error instanceof Error ?
error.message : 'Unknown error'}`);
  }
}

```

Questo file `index.ts` contiene le sezioni seguenti:

- **Blocco import:** utilizza questo blocco per includere le librerie richieste dalla funzione Lambda, come i [client SDK AWS](#).
- **const s3Client** dichiarazione: inizializza un client <https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/client/s3/> Amazon S3 al di fuori della funzione handler. Ciò fa sì che Lambda esegua questo codice durante la fase di inizializzazione e il client viene preservato per essere riutilizzato in più invocazioni. [???](#)
- **type OrderEvent:** definisce la struttura dell'evento di input previsto.
- **export const handler:** Questa è la principale funzione di gestione richiamata da Lambda. Quando distribuisce la tua funzione, specifica `index.handler` la proprietà Handler. [https://docs.aws.amazon.com/lambda/latest/api/API\\_CreateFunction.html#lambda-CreateFunction-request-Handler](https://docs.aws.amazon.com/lambda/latest/api/API_CreateFunction.html#lambda-CreateFunction-request-Handler) Il valore della Handler proprietà è il nome del file e il nome del metodo del gestore esportato, separati da un punto.
- **uploadReceiptToS3function:** Questa è una funzione di supporto a cui fa riferimento la funzione principale del gestore.

Affinché questa funzione funzioni correttamente, il suo ruolo di [esecuzione](#) deve consentire l'azione. `s3:PutObject` Inoltre, assicuratevi di definire la variabile di `RECEIPT_BUCKET` ambiente. Dopo una chiamata riuscita, il bucket Amazon S3 dovrebbe contenere un file di ricevuta.

## Moduli CommonJS ed ES

Node.js supporta due sistemi di moduli: moduli CommonJS ed ECMAScript (moduli ES). Lambda consiglia di utilizzare i moduli ES in quanto supporta l'attesa di primo livello, che consente il completamento di attività asincrone durante l'inizializzazione dell'ambiente di esecuzione. [???](#)

Node.js tratta i file con estensione `.js` come moduli CommonJS mentre un' `.mjs` estensione indica i moduli ES. Per impostazione predefinita, Node.js tratta i file con estensione `.mjs` come moduli

.js CommonJS. È possibile Node.js configurare il trattamento .js dei file come moduli ES specificando l'typeas module nel file della funzione. package.json È possibile configurare Node.js in Lambda per rilevare automaticamente se un .js file deve essere trattato come CommonJS o come modulo ES aggiungendo il `--experimental-detect-module` flag alla variabile di ambiente. `NODE_OPTIONS` Per ulteriori informazioni, consulta Funzionalità [sperimentali Node.js](#).

Gli esempi seguenti mostrano i gestori di funzioni scritti utilizzando sia i moduli ES che i moduli CommonJS. Gli esempi rimanenti in questa pagina utilizzano tutti moduli ES.

## Node.js inizializzazione

Node.js utilizza un I/O modello non bloccante che supporta operazioni asincrone efficienti utilizzando un ciclo di eventi. Ad esempio, se Node.js effettua una chiamata di rete, la funzione continua a elaborare altre operazioni senza bloccare una risposta di rete. Quando la risposta di rete viene ricevuta, viene inserita nella coda di callback. Le attività della coda vengono elaborate al termine dell'attività corrente.

Lambda consiglia di utilizzare `await` di primo livello in modo che le attività asincrone avviate durante l'inizializzazione dell'ambiente di esecuzione vengano completate durante l'inizializzazione. Le attività asincrone che non vengono completate durante l'inizializzazione vengono in genere eseguite durante la prima chiamata di funzione. Ciò può causare comportamenti o errori imprevisti.

Ad esempio, l'inizializzazione della funzione potrebbe effettuare una chiamata di rete per recuperare un AWS parametro da Parameter Store. Se questa attività non viene completata durante l'inizializzazione, il valore potrebbe essere nullo durante una chiamata. Può inoltre verificarsi un ritardo tra l'inizializzazione e l'invoke, che può causare errori nelle operazioni sensibili al fattore tempo. In particolare, le chiamate AWS di assistenza possono basarsi su firme di richieste sensibili al fattore tempo, con conseguenti errori delle chiamate di servizio se la chiamata non viene completata durante la fase di inizializzazione.

Il completamento delle attività durante l'inizializzazione in genere migliora le prestazioni di avvio a freddo e le richiama innanzitutto quando si utilizza Provisioned Concurrency. Per ulteriori informazioni, consulta il nostro post sul blog [Using Node.js ES modules and top-level await in. AWS Lambda](#)

## Convenzioni di denominazione dei gestori

Quando si configura una funzione, il valore dell'[impostazione](#) `Handler` è il nome del file e il nome del metodo handler esportato, separati da un punto. L'impostazione predefinita per le funzioni create

nella console e negli esempi in questa guida è `index.handler`. Questo indica il metodo `handler` che viene esportato dal file `index.js` o `index.mjs`.

Se si crea una funzione nella console utilizzando un nome di file o un nome del gestore di funzione diverso, è necessario modificare il nome del gestore predefinito.

#### Modifica del nome del gestore funzioni (console)

1. Apri la pagina [Funzioni](#) della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Scorri verso il basso fino al riquadro Impostazioni di runtime e scegli Modifica.
4. In Gestore, inserisci il nuovo nome per il tuo gestore di funzioni.
5. Scegli Save (Salva).

#### Definizione e accesso all'oggetto evento di input

JSON è il formato di input più comune e standard per le funzioni Lambda. In questo esempio, la funzione prevede un input simile a quanto segue:

```
{
  "order_id": "12345",
  "amount": 199.99,
  "item": "Wireless Headphones"
}
```

Quando si lavora con le funzioni Lambda in TypeScript, è possibile definire la forma dell'evento di input utilizzando un tipo o un'interfaccia. In questo esempio, definiamo la struttura dell'evento utilizzando un tipo:

```
type OrderEvent = {
  order_id: string;
  amount: number;
  item: string;
}
```

Dopo aver definito il tipo o l'interfaccia, utilizzateli nella firma del gestore per garantire la sicurezza del tipo:

```
export const handler = async (event: OrderEvent): Promise<string> => {
```

Durante la compilazione, TypeScript verifica che l'oggetto evento contenga i campi obbligatori con i tipi corretti. Ad esempio, il TypeScript compilatore segnala un errore se si tenta di utilizzarlo `event.order_id` come numero o `event.amount` come stringa.

Schemi di gestione validi per le funzioni TypeScript

Si consiglia di utilizzare [async/await](https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/using-a-callback-function.html) per dichiarare il gestore della funzione invece di utilizzare i callback. <https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/using-a-callback-function.html> Async/await è un modo conciso e leggibile per scrivere codice asincrono, senza la necessità di callback annidati o di concatenare promesse. Con async/await, puoi scrivere codice che si legge come codice sincrono, pur essendo asincrono e non bloccante.

Gli esempi in questa sezione utilizzano il tipo `S3Event`. Tuttavia, puoi utilizzare qualsiasi altro tipo di AWS evento nel [pacchetto](#) `@types/aws-lambda` o definire il tuo tipo di evento. Per utilizzare i tipi di `@types/aws-lambda`

1. Aggiungi il `@types/aws-lambda` pacchetto come dipendenza di sviluppo:

```
npm install -D @types/aws-lambda
```

2. Importa i tipi di cui hai bisogno, ad esempio `ContextS3Event`, o `Callback`.

gestori di funzioni asincrone (consigliati)

La parola chiave `async` contrassegna una funzione come asincrona, mentre la parola chiave `await` mette in pausa l'esecuzione della funzione fino alla risoluzione di `Promise`. Il gestore accetta i seguenti argomenti:

- `event`: contiene i dati di input passati alla funzione.
- `context`: contiene informazioni sull'invocazione, sulla funzione e sull'ambiente di esecuzione. Per ulteriori informazioni, consulta [Utilizzo dell'oggetto contestuale Lambda per recuperare informazioni TypeScript sulla funzione](#).

Ecco le firme valide per il `async/await` pattern:

```
export const handler = async (event: S3Event): Promise<void> => { };
```

```
export const handler = async (event: S3Event, context: Context): Promise<void> => { };
```

 Note

Quando elabori matrici di elementi in modo asincrono, assicurati di utilizzare `await with Promise.all` per garantire il completamento di tutte le operazioni. Metodi come `forEach` non aspettano il completamento dei callback asincroni. Per ulteriori informazioni, consulta [Array.prototype.forEach \(\)](#) nella documentazione di Mozilla.

## Gestori di funzioni sincroni

Se la funzione non esegue alcuna attività asincrona, è possibile utilizzare un gestore di funzioni sincrone, utilizzando una delle seguenti firme di funzione:

```
export const handler = (event: S3Event): void => { };
```

```
export const handler = (event: S3Event, context: Context): void => { };
```

## gestori di funzioni di streaming delle risposte

Lambda supporta lo streaming di risposte con Node.js. I gestori delle funzioni di streaming delle risposte utilizzano il decoratore `AWSLambda.streamifyResponse()` e accettano 3 parametri: `event`, `responseStream` e `context`. La firma della funzione è:

```
export const handler = awslambda.streamifyResponse(async (event: APIGatewayProxyEvent, responseStream: NodeJS.WritableStream, context: Context) => { });
```

Per ulteriori informazioni, consulta [Response streaming for Lambda functions](#).

## Callback-based gestori di funzioni

 Note

Callback-based i gestori di funzioni sono supportati solo fino a Node.js 22. A partire da Node.js 24, le attività asincrone dovrebbero essere implementate utilizzando gestori di funzioni asincroni.

Callback-based i gestori di funzioni possono utilizzare gli argomenti `event`, `context` e `callback`. L'argomento `callback` prevede una `Error` e una risposta, che deve essere JSON-serializable.

Ecco la firma valida per il pattern del callback handler:

```
export const handler = (event: S3Event, context: Context, callback: Callback<void>):  
  void => { };
```

La funzione continua a essere eseguita fino a quando il ciclo di [eventi non](#) è vuoto o la funzione scade. La risposta non viene inviata all'invoker finché tutte le attività del ciclo di eventi non giungono a termine. Se la funzione va in timeout, viene invece restituito un errore. È possibile configurare il runtime per inviare immediatamente la risposta impostando [context.callback WaitsForEmptyEventLoop](#) su false.

Example TypeScript funzione con callback

L'esempio seguente utilizza `APIGatewayProxyCallback`, che è un tipo di callback specializzato specifico per le integrazioni di API Gateway. La maggior parte delle fonti di AWS eventi utilizza il Callback tipo generico mostrato nelle firme precedenti.

```
import { Context, APIGatewayProxyCallback, APIGatewayEvent } from 'aws-lambda';  
  
export const lambdaHandler = (event: APIGatewayEvent, context: Context, callback:  
  APIGatewayProxyCallback): void => {  
  console.log(`Event: ${JSON.stringify(event, null, 2)}`);  
  console.log(`Context: ${JSON.stringify(context, null, 2)}`);  
  callback(null, {  
    statusCode: 200,  
    body: JSON.stringify({  
      message: 'hello world',  
    }),  
  });  
};
```

Utilizzo dell'SDK per la JavaScript v3 nel gestore

Spesso, utilizzerai le funzioni Lambda per interagire o apportare aggiornamenti ad altre AWS risorse. Il modo più semplice per interfacciarsi con queste risorse è usare la AWS SDK per JavaScript. Tutti i Node.js runtime Lambda supportati includono l'[SDK per la versione 3. JavaScript](#). Tuttavia, ti consigliamo vivamente di includere i client AWS SDK di cui hai bisogno nel tuo pacchetto di distribuzione. Ciò massimizza la [compatibilità con le versioni precedenti](#) durante i futuri aggiornamenti del runtime di Lambda.

Per aggiungere dipendenze SDK alla tua funzione, usa il `npm install` comando per i client SDK specifici di cui hai bisogno. Nel codice di esempio, abbiamo utilizzato il client [Amazon S3](#). Aggiungi queste dipendenze eseguendo i seguenti comandi nella directory che contiene il file `package.json`:

```
npm install @aws-sdk/client-s3
```

Nel codice della funzione, importate il client e i comandi necessari, come dimostra la funzione di esempio:

```
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';
```

Quindi, inizializza un client [Amazon S3](#):

```
const s3Client = new S3Client();
```

In questo esempio, abbiamo inizializzato il nostro client Amazon S3 all'esterno della funzione del gestore principale per evitare di doverlo inizializzare ogni volta che invochiamo la nostra funzione. Dopo aver inizializzato il client SDK, puoi utilizzarlo per effettuare chiamate API per quel servizio. AWS Il codice di esempio richiama l'azione [PutObject](#) API Amazon S3 nel modo seguente:

```
const command = new PutObjectCommand({
  Bucket: bucketName,
  Key: key,
  Body: receiptContent
});
```

## Accesso alle variabili d'ambiente

Nel codice del gestore, puoi fare riferimento a qualsiasi variabile di [ambiente](#) utilizzando `process.env`. In questo esempio, facciamo riferimento alla variabile di ambiente `RECEIPT_BUCKET` definita utilizzando le seguenti righe di codice:

```
// Access environment variables
const bucketName = process.env.RECEIPT_BUCKET;
if (!bucketName) {
  throw new Error('RECEIPT_BUCKET environment variable is not set');
}
```

## Utilizzo dello stato globale

Lambda esegue il codice statico durante la fase di [inizializzazione](#) prima di richiamare la funzione per la prima volta. Le risorse create durante l'inizializzazione restano in memoria tra le invocazioni, in modo da evitare di doverle creare ogni volta che si invoca la funzione.

Nel codice di esempio, il codice di inizializzazione del client S3 è esterno al gestore. Il runtime inizializza il client prima che la funzione gestisca il primo evento e il client rimane disponibile per il riutilizzo in tutte le chiamate.

## Procedure consigliate per il codice per le funzioni Lambda TypeScript

Segui queste linee guida quando crei funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. In questo modo è possibile creare una funzione di cui è più semplice eseguire l'unit test.
- Controllare le dipendenze nel pacchetto di distribuzione della funzione. L'ambiente di AWS Lambda esecuzione contiene una serie di librerie. Per i runtime di Python Node.js e Python, questi includono gli AWS SDK. Per abilitare il set di caratteristiche e aggiornamenti della sicurezza più recenti, Lambda aggiorna periodicamente tali librerie. Tali aggiornamenti possono introdurre lievi modifiche al comportamento della funzione Lambda. Per mantenere il controllo completo delle dipendenze utilizzate dalla funzione, inserire tutte le dipendenze nel pacchetto di implementazione.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del [contesto di esecuzione](#).
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. In questo modo viene ridotta la quantità di tempo necessaria per il download del pacchetto e per la relativa decompressione prima dell'invocazione.

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione.

Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.



Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedi [Riutilizzo delle connessioni con in. Keep-Alive Node.js](#)

Utilizzare [le variabili di ambiente](#) per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su 0 per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionali alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha una dipendenza su queste API non pubbliche. Consulta il [riferimento all'API](#) per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta [Come posso rendere idempotente la mia funzione Lambda?](#).

## Distribuisci TypeScript codice transpilato in Lambda con archivi di file .zip

Prima di poter distribuire il TypeScript codice in AWS Lambda, devi traspirarlo in JavaScript. Questa pagina spiega tre modi per creare e distribuire TypeScript codice in Lambda con archivi di file .zip:

- [Usando \(\) AWS Serverless Application Model AWS SAM](#)
- [Usando il Kit di sviluppo per il cloud AWS \(AWS CDK\)](#)
- [Usando AWS Command Line Interface \(AWS CLI\) ed esbuild](#)

AWS SAM e AWS CDK semplifica la creazione e l'implementazione delle funzioni. TypeScript La [specifica del modello AWS SAM](#) fornisce una sintassi semplice e chiara per descrivere le funzioni, le API, le autorizzazioni, le configurazioni e gli eventi Lambda che compongono l'applicazione serverless. Con [AWS CDK](#), puoi creare applicazioni affidabili, scalabili ed economiche nel cloud con la notevole potenza espressiva di un linguaggio di programmazione. AWS CDK È destinato a utenti con esperienza da moderata a elevata. AWS AWS CDK Sia il che AWS SAM usano esbuild per TypeScript trasporre il codice in JavaScript.

Utilizzo AWS SAM per distribuire TypeScript codice su Lambda

Segui i passaggi seguenti per scaricare, creare e distribuire un' TypeScript applicazione Hello World di esempio utilizzando AWS SAM. Questa applicazione implementa un backend API di base. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando si invia una richiesta GET all'endpoint API Gateway, viene richiamata la funzione Lambda. La funzione restituisce un messaggio hello world.

### Note

AWS SAM utilizza esbuild per creare funzioni Node.js Lambda a partire dal TypeScript codice. Il supporto per esbuild è attualmente disponibile in anteprima pubblica. Durante l'anteprima pubblica, il supporto per esbuild potrebbe essere soggetto a modifiche incompatibili con le versioni precedenti.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [AWS CLI versione 2](#)

- [AWS SAM CLI versione 1.75 o successiva](#)
- Node.js

Distribuisci un campione AWS SAM applicazione

1. Inizializza l'applicazione utilizzando il modello Hello World. TypeScript

```
sam init --app-template hello-world-typescript --name sam-app --package-type Zip --runtime nodejs24.x
```

2. (Facoltativo) L'applicazione di esempio include configurazioni per strumenti di uso comune, ad esempio [ESLint](#) per analisi del codice e [Jest](#) per test unitario. Per eseguire i comandi lint e test:

```
cd sam-app/hello-world
npm install
npm run lint
npm run test
```

3. Costruisci l'app.

```
cd sam-app
sam build
```

4. Distribuire l'app.

```
sam deploy --guided
```

5. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, rispondi con Enter.
6. L'output mostra l'endpoint per la REST API. Apri l'endpoint in un browser per testare la funzione. Dovresti vedere questa risposta:

```
{"message":"hello world"}
```

7. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

## Utilizzo di AWS CDK per distribuire TypeScript codice su Lambda

Segui i passaggi seguenti per creare e distribuire un' TypeScript applicazione di esempio utilizzando AWS CDK. Questa applicazione implementa un backend API di base. È costituito da un endpoint API Gateway e da una funzione Lambda. Quando si invia una richiesta GET all'endpoint API Gateway, viene richiamata la funzione Lambda. La funzione restituisce un messaggio `hello world`.

### Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [AWS CLI versione 2](#)
- [AWS CDK versione 2](#)
- Node.js
- [Docker](#) o [esbuild](#)

### Distribuisci un campione AWS CDK applicazione

1. Crea una directory di progetto per la nuova applicazione.

```
mkdir hello-world  
cd hello-world
```

2. Inizializza l'app.

```
cdk init app --language typescript
```

3. Aggiungi il [pacchetto](#) `@types/aws-lambda` come dipendenza di sviluppo. Questo pacchetto contiene le definizioni dei tipi per Lambda.

```
npm install -D @types/aws-lambda
```

4. Apri la directory `lib`. Dovrebbe essere visualizzato un file chiamato `hello-world-stack.ts`. Crea due nuovi file in questa directory: `hello-world.function.ts` e `hello-world.ts`.
5. Apri `hello-world.function.ts` e aggiungi il seguente codice al file. Questo è il codice per la funzione Lambda.

**Note**

L'importazione importa le definizioni dei tipi da `@-lambda`. [types/aws](#) Non importa il pacchetto NPM `aws-lambda`, che è uno strumento di terzi non correlato. Per ulteriori informazioni, vedi <https://github.com/DefinitelyTyped/DefinitelyTyped/tree/master/types/aws-lambda> `aws-lambda` nel repository. DefinitelyTyped GitHub

```
import { Context, APIGatewayProxyResult, APIGatewayEvent } from 'aws-lambda';

export const handler = async (event: APIGatewayEvent, context: Context):
  Promise<APIGatewayProxyResult> => {
  console.log(`Event: ${JSON.stringify(event, null, 2)}`);
  console.log(`Context: ${JSON.stringify(context, null, 2)}`);
  return {
    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
    }),
  };
};
```

6. Apri `hello-world.ts` e aggiungi il seguente codice al file. Contiene il [NodejsFunction costruito](#), che crea la funzione Lambda, e il [LambdaRestApi costruito, che crea l'API REST](#).

```
import { Construct } from 'constructs';
import { NodejsFunction } from 'aws-cdk-lib/aws-lambda-nodejs';
import { LambdaRestApi } from 'aws-cdk-lib/aws-apigateway';

export class HelloWorld extends Construct {
  constructor(scope: Construct, id: string) {
    super(scope, id);
    const helloFunction = new NodejsFunction(this, 'function');
    new LambdaRestApi(this, 'apigw', {
      handler: helloFunction,
    });
  }
}
```

Il costruito `NodejsFunction` presuppone quanto segue per impostazione predefinita:

- Viene richiamato il gestore di funzioni `handler`.
- Il file `.ts` che contiene il codice funzione (`hello-world.function.ts`) si trova nella stessa directory del file `.ts` che contiene il costrutto (`hello-world.ts`). Il costrutto utilizza l'ID del costrutto (`"hello-world"`) e il nome del file del gestore Lambda (`"funzione"`) per trovare il codice funzione. Ad esempio, se il codice funzione si trova in un file chiamato `hello-world.my-function.ts`, il file `hello-world.ts` deve fare riferimento al codice funzione in questo modo:

```
const helloFunction = new NodejsFunction(this, 'my-function');
```

È possibile modificare questo comportamento e configurare altri parametri `esbuild`. Per ulteriori informazioni, consulta [Configurazione di esbuild](#) nel riferimento API AWS CDK

7. Apri `hello-world-stack.ts`. Questo è il codice che definisce lo [stack AWS CDK](#). Sostituisci il codice con il seguente:

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { HelloWorld } from './hello-world';

export class HelloWorldStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new HelloWorld(this, 'hello-world');
  }
}
```

8. dalla directory `hello-world` contenente il file `cdk.json`, implementa l'applicazione.

```
cdk deploy
```

9. Compila e AWS CDK impacchetta la funzione Lambda utilizzando `esbuild`, quindi distribuisce la funzione nel runtime Lambda. L'output mostra l'endpoint per la REST API. Apri l'endpoint in un browser per testare la funzione. Dovresti vedere questa risposta:

```
{"message":"hello world"}
```

Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

## Utilizzo di AWS CLI ed esbuild per distribuire il codice su Lambda TypeScript

L'esempio seguente dimostra come transpilare e distribuire TypeScript codice in Lambda utilizzando esbuild e. esbuild produce un file con tutte le AWS CLI dipendenze. JavaScript Questo è l'unico file che devi aggiungere all'archivio .zip.

### Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [AWS CLI versione 2](#)
- Node.js
- Un [ruolo di esecuzione](#) per la funzione Lambda
- Per gli utenti Windows, un'utilità di file zip come [7zip](#).

### Implementare una funzione di esempio

1. Sul tuo computer locale, crea una directory di progetto per la nuova funzione.
2. Crea un nuovo Node.js progetto con npm o un gestore di pacchetti a tua scelta.

```
npm init
```

3. Aggiungi i [pacchetti @types/aws-lambda ed esbuild](#) come dipendenze di sviluppo. Il pacchetto @types/aws-lambda contiene le definizioni dei tipi per Lambda.

```
npm install -D @types/aws-lambda esbuild
```

4. Crea un nuovo file denominato index.ts. Aggiungi il seguente codice al nuovo file. Questo è il codice per la funzione Lambda. La funzione restituisce un messaggio hello world. La funzione non crea alcuna risorsa API Gateway.

#### Note

L'importistruzione importa le definizioni dei tipi da [@types/aws-lambda](#). Non importa il pacchetto NPM aws-lambda, che è uno strumento di terzi non correlato. Per ulteriori informazioni, vedi <https://github.com/DefinitelyTyped/DefinitelyTyped/tree/master/types/aws-lambda> aws-lambda nel repository. DefinitelyTyped GitHub

```
import { Context, APIGatewayProxyResult, APIGatewayEvent } from 'aws-lambda';

export const handler = async (event: APIGatewayEvent, context: Context):
Promise<APIGatewayProxyResult> => {
  console.log(`Event: ${JSON.stringify(event, null, 2)}`);
  console.log(`Context: ${JSON.stringify(context, null, 2)}`);
  return {
    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
    }),
  };
};
```

5. Aggiungi uno script di compilazione al file package.json. esbuild viene configurato per creare automaticamente il pacchetto di implementazione .zip. Per ulteriori informazioni, consulta la sezione relativa alla [compilazione degli script](#) nella documentazione di esbuild.

## Linux and MacOS

```
"scripts": {
  "prebuild": "rm -rf dist",
  "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --
target=es2020 --outfile=dist/index.js",
  "postbuild": "cd dist && zip -r index.zip index.js*"
},
```

## Windows

In questo esempio, il comando "postbuild" utilizza l'utilità [7zip](#) per creare il file con estensione .zip. Utilizza l'utilità zip di Windows preferita e modifica il comando secondo necessità.

```
"scripts": {
  "prebuild": "del /q dist",
  "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --
target=es2020 --outfile=dist/index.js",
  "postbuild": "cd dist && 7z a -tzip index.zip index.js*"
},
```



## 6. Creare il pacchetto.

```
npm run build
```

## 7. Crea una funzione Lambda utilizzando il pacchetto di implementazione .zip. Sostituisci il testo evidenziato con l'Amazon Resource Name (ARN) del tuo [ruolo di esecuzione](#).

```
aws lambda create-function --function-name hello-world --runtime "nodejs24.x" --  
role arn:aws:iam::123456789012:role/lambda-ex --zip-file "fileb://dist/index.zip"  
--handler index.handler
```

## 8. [Esegui un evento di test](#) per confermare che la funzione restituisca la risposta seguente. Se desideri richiamare questa funzione utilizzando API Gateway, [crea e configura una REST API](#).

```
{  
  "statusCode": 200,  
  "body": "{\"message\":\"hello world\"}"  
}
```

## Implementa codice trascritto TypeScript in Lambda con immagini di container

[Puoi distribuire il TypeScript codice in una AWS Lambda funzione come immagine del contenitore](#)

[Node.js](#). AWS fornisce [immagini di base](#) per Node.js per aiutarti a creare l'immagine del contenitore. Queste immagini di base sono precaricate con un runtime del linguaggio e altri componenti necessari per eseguire l'immagine su Lambda. AWS fornisce un Dockerfile per ciascuna delle immagini di base per facilitare la creazione dell'immagine del contenitore.

Se si utilizza un'immagine di base di community o aziendale privata, è necessario [aggiungere il client di interfaccia di runtime \(RIC\) Node.js](#) all'immagine di base per renderla compatibile con Lambda.

Lambda fornisce un emulatore di interfaccia di runtime per i test in locale. Le immagini di AWS base per Node.js includono l'emulatore di interfaccia di runtime. Se utilizzi un'immagine di base alternativa, come Alpine Linux o Debian, puoi [creare il simulatore nella tua immagine](#) o [installarlo sul tuo computer locale](#).

Utilizzo di un'immagine di base Node.js per creare e impacchettare il codice TypeScript funzionale

### Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [AWS CLI versione 2](#)
- [Docker](#) (versione minima 25.0.0)
- Il [plugin buildx](#) di Docker.
- Node.js 22.x

### Creazione di un'immagine da un'immagine di base

Per creare un'immagine da un'immagine di AWS base per Lambda

1. Sul tuo computer locale, crea una directory di progetto per la nuova funzione.
2. Crea un nuovo progetto Node.js con npm o un gestore di pacchetti a tua scelta.

```
npm init
```

3. Aggiungi i pacchetti [@types/aws-lambda](#) e [esbuild](#) come dipendenze di sviluppo. Il pacchetto `@types/aws-lambda` contiene le definizioni dei tipi per Lambda.

```
npm install -D @types/aws-lambda esbuild
```

4. Aggiungi uno [script di compilazione](#) al file `package.json`.

```
"scripts": {  
  "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --  
target=es2020 --outfile=dist/index.js"  
}
```

5. Crea un nuovo file denominato `index.ts`. Aggiungi il codice di esempio seguente al nuovo file. Questo è il codice per la funzione Lambda. La funzione restituisce un messaggio `hello world`.

 Note

L'istruzione `import` importa le definizioni dei tipi da [@types/aws-lambda](#). Non importa il pacchetto NPM `aws-lambda`, che è uno strumento di terzi non correlato. Per ulteriori informazioni, consulta [aws-lambda](#) nel repository. DefinitelyTyped GitHub

```
import { Context, APIGatewayProxyResult, APIGatewayEvent } from 'aws-lambda';  
  
export const handler = async (event: APIGatewayEvent, context: Context):  
  Promise<APIGatewayProxyResult> => {  
  console.log(`Event: ${JSON.stringify(event, null, 2)}`);  
  console.log(`Context: ${JSON.stringify(context, null, 2)}`);  
  return {  
    statusCode: 200,  
    body: JSON.stringify({  
      message: 'hello world',  
    }),  
  };  
};
```

6. Crea un nuovo Dockerfile con la seguente configurazione:
- Imposta la proprietà `FROM` sull'URI dell'immagine di base.
  - Imposta l'argomento `CMD` per specificare il gestore della funzione Lambda.

Il seguente Dockerfile di esempio utilizza una build multi-fase. Il primo passaggio trascrive il codice in TypeScript JavaScript. Il secondo passaggio produce un'immagine del contenitore che contiene solo JavaScript file e dipendenze di produzione.

Nota che l'esempio Dockerfile non include un'[istruzione USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

### Example Dockerfile

```
FROM public.ecr.aws/lambda/nodejs:22 as builder
WORKDIR /usr/app
COPY package.json index.ts ./
RUN npm install
RUN npm run build

FROM public.ecr.aws/lambda/nodejs:22
WORKDIR ${LAMBDA_TASK_ROOT}
COPY --from=builder /usr/app/dist/* ./
CMD ["index.handler"]
```

7. Crea l'immagine Docker con il comando [docker build](#). L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il [tag](#) `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'opzione `--provenance=false`.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test .
```

#### Note

Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Se intendi creare una funzione Lambda utilizzando l'architettura del set di ARM64 istruzioni, assicurati di modificare il comando per utilizzare invece l'opzione `--platform linux/arm64`.

## (Facoltativo) Test dell'immagine in locale

1. Avvia l'immagine Docker con il comando `docker run`. In questo esempio, `docker-image` è il nome dell'immagine e `test` è il tag.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

### Note

Se hai creato l'immagine Docker per l'architettura del set di ARM64 istruzioni, assicurati di utilizzare l'opzione `--platform linux/arm64` invece di `--platform linux/amd64`.

2. Da una nuova finestra di terminale, invia un evento all'endpoint locale.

### Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload": "hello world!"}'
```

### PowerShell

In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType "application/json"
```

3. Ottieni l'ID del container.

```
docker ps
```

4. Utilizza il comando [docker kill](#) per arrestare il container. In questo comando, sostituisci 3766c4ab331c con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

## Implementazione dell'immagine

### Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Esegui il [get-login-password](#) comando per autenticare la CLI Docker nel tuo registro Amazon ECR.
  - Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
  - Sostituiscilo 111122223333 con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

 Note

Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando [docker tag](#) per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:
  - `docker-image:test` è il nome e [tag](#) dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
  - Sostituisci l'<ECRrepositoryUri> con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando [docker push](#) per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Crea un ruolo di esecuzione](#) per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per ImageUri, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta [Autorizzazioni multiaccount Amazon ECR](#).

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.



Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nell'archivio Amazon ECR e quindi utilizzare il [update-function-code](#) comando per distribuire l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per distribuire la nuova immagine nella stessa funzione Lambda, è necessario utilizzare [update-function-code](#) il comando, anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \  
  --function-name hello-world \  
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --publish
```

## Utilizzo dell'oggetto contestuale Lambda per recuperare informazioni TypeScript sulla funzione

Quando Lambda esegue la funzione, passa un oggetto Context al [gestore](#). Questo oggetto fornisce i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

Per abilitare il controllo del tipo per l'oggetto di contesto, è necessario aggiungere il pacchetto [@types /aws-lambda](#) come dipendenza di sviluppo e importare il tipo. Context Per ulteriori informazioni, consulta [Definizioni dei tipi per Lambda](#).

### Metodi del contesto

- `getRemainingTimeInMillis()`: restituisce il numero di millisecondi rimasti prima del timeout dell'esecuzione.

### Proprietà del contesto

- `functionName`: il nome della funzione Lambda.
- `functionVersion`: la [versione](#) della funzione.
- `invokedFunctionArn`: l'Amazon Resource Name (ARN) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `memoryLimitInMB`: la quantità di memoria allocata per la funzione.
- `awsRequestId`: l'identificatore della richiesta di invocazione.
- `logGroupName`: il gruppo di log per la funzione.
- `logStreamName`: il flusso di log per l'istanza della funzione.
- `identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
  - `cognitoIdentityId`: l'identità autenticata di Amazon Cognito.
  - `cognitoIdentityPoolId`: il pool di identità Amazon Cognito che ha autorizzato l'invocazione.
- `clientContext`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`

- `client.app_version_code`
  - `client.app_package_name`
  - `env.platform_version`
  - `env.platform`
  - `env.make`
  - `env.model`
  - `env.locale`
  - Custom: valori personalizzati impostati dall'applicazione mobile.
- `callbackWaitsForEmptyEventLoop`— Per impostazione predefinita (`true`), quando si utilizza un gestore di funzioni basato sul callback, Lambda attende che il ciclo di eventi sia vuoto dopo l'esecuzione del callback prima di terminare l'invoca della funzione. Imposta su `false` per inviare la risposta e terminare la chiamata immediatamente dopo l'esecuzione del callback anziché attendere che il ciclo di eventi sia vuoto. Gli eventi eccezionali continuano a verificarsi durante la chiamata successiva. Nota che Lambda supporta gestori di funzioni basati su callback solo per Node.js 22 e runtime precedenti.

### Example File `index.js`

La seguente funzione di esempio registra le informazioni di contesto e restituisce la posizione dei log.

#### Note

Prima di utilizzare questo codice in una funzione Lambda, devi aggiungere il pacchetto [@types/aws-lambda](https://www.npmjs.com/package/@types/aws-lambda) come dipendenza di sviluppo. Questo pacchetto contiene le definizioni dei tipi per Lambda. Per ulteriori informazioni, consulta [Definizioni dei tipi per Lambda](#).

```
import { Context } from 'aws-lambda';
export const lambdaHandler = async (event: string, context: Context): Promise<string>
=> {
  console.log('Remaining time: ', context.getRemainingTimeInMillis());
  console.log('Function name: ', context.functionName);
  return context.logStreamName;
};
```

## Registra e monitora le funzioni TypeScript Lambda

AWS Lambda monitora automaticamente le funzioni Lambda e invia le voci di registro ad Amazon CloudWatch. La tua funzione Lambda include un gruppo di CloudWatch log Logs e un flusso di log per ogni istanza della tua funzione. L'ambiente di runtime di Lambda invia al flusso di log i dettagli su ogni invocazione e altri output dal codice della funzione. Per ulteriori informazioni sui CloudWatch log, consulta [Invio dei log delle funzioni Lambda ai CloudWatch log](#).

Per i log di output dal codice della funzione, puoi utilizzare i metodi nell'[oggetto console](#). Per una registrazione più dettagliata, puoi utilizzare qualunque libreria di registrazione che scrive su `stdout` o `stderr`.

### Sections

- [Utilizzo di strumenti di registrazione e librerie](#)
- [Utilizzo di Powertools per AWS Lambda \(TypeScript\) e AWS SAM per la registrazione strutturata](#)
- [Utilizzo di Powertools per AWS Lambda \(TypeScript\) e il AWS CDK per la registrazione strutturata](#)
- [Visualizzazione dei log nella console Lambda](#)
- [Visualizzazione dei log nella console CloudWatch](#)

### Utilizzo di strumenti di registrazione e librerie

[Powertools for AWS Lambda \(TypeScript\)](#) è un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori. L'[utilità di registrazione](#) fornisce un logger ottimizzato per Lambda che include informazioni aggiuntive sul contesto delle funzioni in tutte le funzioni con output strutturato come JSON. Utilizza l'utility per eseguire le seguenti operazioni:

- Acquisizione di campi essenziali dal contesto Lambda, avvio a freddo e output di registrazione della struttura come JSON
- Registrazione degli eventi di chiamata Lambda quando richiesto (disabilitata per impostazione predefinita)
- Stampa di tutti i log solo per una percentuale di chiamate tramite campionamento dei log (disabilitata per impostazione predefinita)
- Aggiunta di chiavi supplementari al log strutturato in qualsiasi momento
- Utilizzo di un formattatore di log personalizzato (Bring Your Own Formatter) per generare i log in una struttura compatibile con Logging RFC dell'organizzazione

## Utilizzo di Powertools per AWS Lambda (TypeScript) e AWS SAM per la registrazione strutturata

Segui i passaggi seguenti per scaricare, creare e distribuire un' TypeScript applicazione Hello World di esempio con [moduli](#) Powertools for AWS Lambda (TypeScript) integrati utilizzando il. AWS SAM Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format a e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio hello world.

### Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Node.js 20 o versione successiva
- [AWS CLI versione 2](#)
- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI. AWS SAM](#)

### Distribuisci un esempio AWS SAM applicazione

1. Inizializza l'applicazione utilizzando il modello Hello World. TypeScript

```
sam init --app-template hello-world-powertools-typescript --name sam-app --package-type Zip --runtime nodejs24.x
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi Enter.

 Note

Perché HelloWorldFunction potrebbe non avere l'autorizzazione definita, va bene? , assicurati di entrarey.

## 5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query
'Stacks[0].Outputs[?OutputKey=='HelloWorldApi'].OutputValue' --output text
```

## 6. Richiama l'endpoint dell'API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere i log per la funzione, esegui [sam logs](#). Per ulteriori informazioni, consulta l'argomento relativo all'[utilizzo dei log](#) nella Guida per sviluppatori AWS Serverless Application Model .

```
sam logs --stack-name sam-app
```

L'output del log ha la struttura seguente:

```
2025/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2025-08-31T09:33:10.552000
START RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb Version: $LATEST
2025/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2025-08-31T09:33:10.594000
2025-08-31T09:33:10.557Z 70693159-7e94-4102-a2af-98a6343fb8fb
INFO {"_aws":{"Timestamp":1661938390556,"CloudWatchMetrics":
[{"Namespace":"sam-app","Dimensions":[["service"]],"Metrics":
[{"Name":"ColdStart","Unit":"Count"}]}]}, "service":"helloWorld","ColdStart":1}
2025/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2025-08-31T09:33:10.595000
2025-08-31T09:33:10.595Z 70693159-7e94-4102-a2af-98a6343fb8fb INFO
{"level":"INFO","message":"This is an INFO log - sending HTTP 200 - hello world
response","service":"helloWorld","timestamp":"2025-08-31T09:33:10.594Z"}
2025/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2025-08-31T09:33:10.655000
2025-08-31T09:33:10.655Z 70693159-7e94-4102-a2af-98a6343fb8fb INFO
```

```
{ "_aws": { "Timestamp": 1661938390655, "CloudWatchMetrics": [ { "Namespace": "sam-app", "Dimensions": [ [ "service" ] ], "Metrics": [ ] } ], "service": "helloWorld" }
2025/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2025-08-31T09:33:10.754000 END
RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb
2025/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2025-08-31T09:33:10.754000
REPORT RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb Duration: 201.55 ms Billed
Duration: 202 ms Memory Size: 128 MB Max Memory Used: 66 MB Init Duration: 252.42
ms
XRAY TraceId: 1-630f2ad5-1de22b6d29a658a466e7ecf5 SegmentId: 567c116658fbf11a
Sampled: true
```

8. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

## Gestione della conservazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, elimina il gruppo di log o configura un periodo di conservazione dopo il quale i log CloudWatch vengono eliminati automaticamente. Per impostare la conservazione dei log, aggiungi quanto segue al tuo AWS SAM modello:

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

Utilizzo di Powertools per AWS Lambda (TypeScript) e il AWS CDK per la registrazione strutturata

Segui i passaggi seguenti per scaricare, creare e distribuire un' TypeScript applicazione Hello World di esempio con [moduli](#) Powertools for AWS Lambda (TypeScript) integrati utilizzando il. AWS CDK Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda.

Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format a e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio `hello world`.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Node.js 20 o versione successiva
- [AWS CLI versione 2](#)
- [AWS CDK versione 2](#)
- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI. AWS SAM](#)

## Distribuisci un esempio AWS CDK applicazione

1. Crea una directory di progetto per la nuova applicazione.

```
mkdir hello-world
cd hello-world
```

2. Inizializza l'app.

```
cdk init app --language typescript
```

3. Aggiungi il [pacchetto](#) `@types/aws-lambda` come dipendenza di sviluppo.

```
npm install -D @types/aws-lambda
```

4. Installa l'[utilità Logger](#) di Powertools.

```
npm install @aws-lambda-powertools/logger
```

5. Apri la directory `lib`. Dovrebbe essere visualizzato un file chiamato `hello-world-stack.ts`. Crea due nuovi file in questa directory: `hello-world.function.ts` e `hello-world.ts`.
6. Apri `hello-world.function.ts` e aggiungi il seguente codice al file. Questo è il codice per la funzione Lambda.

```
import { APIGatewayEvent, APIGatewayProxyResult, Context } from 'aws-lambda';
```



```
import { Logger } from '@aws-lambda-powertools/logger';
const logger = new Logger();

export const handler = async (event: APIGatewayEvent, context: Context):
  Promise<APIGatewayProxyResult> => {
  logger.info('This is an INFO log - sending HTTP 200 - hello world response');
  return {
    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
    }),
  };
};
```

7. Apri `hello-world.ts` e aggiungi il seguente codice al file. Contiene il [NodejsFunction costruito](#), che crea la funzione Lambda, configura le variabili di ambiente per Powertools e imposta la conservazione dei log su una settimana. Include anche il [LambdaRestApi costruito](#), che crea l'API REST.

```
import { Construct } from 'constructs';
import { NodejsFunction } from 'aws-cdk-lib/aws-lambda-nodejs';
import { LambdaRestApi } from 'aws-cdk-lib/aws-apigateway';
import { RetentionDays } from 'aws-cdk-lib/aws-logs';
import { CfnOutput } from 'aws-cdk-lib';

export class HelloWorld extends Construct {
  constructor(scope: Construct, id: string) {
    super(scope, id);
    const helloFunction = new NodejsFunction(this, 'function', {
      environment: {
        Powertools_SERVICE_NAME: 'helloWorld',
        LOG_LEVEL: 'INFO',
      },
      logRetention: RetentionDays.ONE_WEEK,
    });
    const api = new LambdaRestApi(this, 'apigw', {
      handler: helloFunction,
    });
    new CfnOutput(this, 'apiUrl', {
      exportName: 'apiUrl',
      value: api.url,
    });
  }
}
```

```
}
```

8. Apri `hello-world-stack.ts`. Questo è il codice che definisce lo [stack AWS CDK](#). Sostituisci il codice con il seguente:

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { HelloWorld } from './hello-world';

export class HelloWorldStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new HelloWorld(this, 'hello-world');
  }
}
```

9. Passa alla directory del progetto.

```
cd hello-world
```

10. Distribuisci l'applicazione.

```
cdk deploy
```

11. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
'Stacks[0].Outputs[?ExportName==`apiUrl`].OutputValue' --output text
```

12. Richiama l'endpoint dell'API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

13. Per ottenere i log per la funzione, esegui [sam logs](#). Per ulteriori informazioni, consulta l'argomento relativo all'[utilizzo dei log](#) nella Guida per sviluppatori AWS Serverless Application Model .

```
sam logs --stack-name HelloWorldStack
```

L'output del log ha la struttura seguente:

```
2025/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2025-08-31T14:48:37.047000
  START RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c Version: $LATEST
2025/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2025-08-31T14:48:37.050000 {
  "level": "INFO",
  "message": "This is an INFO log - sending HTTP 200 - hello world response",
  "service": "helloWorld",
  "timestamp": "2025-08-31T14:48:37.048Z",
  "xray_trace_id": "1-630f74c4-2b080cf77680a04f2362bcf2"
}
2025/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2025-08-31T14:48:37.082000 END
  RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c
2025/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2025-08-31T14:48:37.082000
  REPORT RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c Duration: 34.60 ms Billed
  Duration: 35 ms Memory Size: 128 MB Max Memory Used: 57 MB Init Duration: 173.48
  ms
```

14. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
cdk destroy
```

## Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor di codice incorporato, i log sono contenuti nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, trovi l'output del registro nella sezione Dettagli.

## Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

## Per visualizzare i log sulla console CloudWatch

1. Aprire la pagina dei gruppi di [log](#) sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (/aws/lambda/***your-function-name***).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'[istanza della funzione](#). Un flusso di log viene visualizzato quando si aggiorna la funzione Lambda e quando vengono create istanze aggiuntive per gestire chiamate simultanee. Per trovare i log per una chiamata specifica, ti consigliamo di strumentare la tua funzione con. AWS X-Ray X-Ray registra i dettagli sulla richiesta e sul flusso di log nella traccia.

## TypeScript Codice di tracciamento in AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzarlo X-Ray per tracciare una richiesta mentre attraversa le risorse dell'applicazione, che possono includere funzioni Lambda e altri servizi. AWS

Per inviare dati di tracciamento a X-Ray, puoi utilizzare una delle tre librerie SDK:

- [AWS Distro for OpenTelemetry \(ADOT\)](#): una distribuzione sicura, pronta per la produzione e AWS supportata dell'SDK (OtEL). OpenTelemetry
- [AWS X-Ray SDK per Node.js](#): un SDK per la generazione e l'invio di dati di traccia a. X-Ray
- [Powertools for AWS Lambda \(TypeScript\)](#): un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori.

Ciascuno degli SDK offre modi per inviare i dati di telemetria al servizio. X-Ray Puoi quindi utilizzarlo X-Ray per visualizzare, filtrare e ottenere informazioni dettagliate sulle metriche delle prestazioni dell'applicazione per identificare problemi e opportunità di ottimizzazione.

### Important

The X-Ray e Powertools for AWS Lambda SDK fanno parte di una soluzione di strumentazione strettamente integrata offerta da. AWS I livelli Lambda ADOT fanno parte di uno standard di settore per la strumentazione di tracciamento che in generale raccoglie più dati, ma potrebbero non essere adatti a tutti i casi d'uso. È possibile implementare il tracciamento end-to-end utilizzando entrambe le soluzioni. X-Ray Per saperne di più sulla scelta tra di esse, consulta [Scelta tra AWS Distro for Open Telemetry e SDK. X-Ray](#)

## Sections

- [Usare Powertools per AWS Lambda \(TypeScript\) e AWS SAM per tracciare](#)
- [Usare Powertools per AWS Lambda \(TypeScript\) e il AWS CDK per tracciare](#)
- [Interpretazione e tracciamento X-Ray](#)

Usare Powertools per AWS Lambda (TypeScript) e AWS SAM per tracciare

Segui i passaggi seguenti per scaricare, creare e distribuire un' TypeScript applicazione Hello World di esempio con [moduli](#) Powertools for AWS Lambda (TypeScript) integrati utilizzando. AWS SAM

Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format a e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio `hello world`.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Node.js
- [AWS CLI versione 2](#)
- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI. AWS SAM](#)

## Distribuisci un esempio AWS SAM applicazione

1. Inizializza l'applicazione utilizzando il modello Hello World. TypeScript

```
sam init --app-template hello-world-powertools-typescript --name sam-app --package-type Zip --runtime nodejs24.x --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi `Enter`.

### Note

Perché HelloWorldFunction potrebbe non avere l'autorizzazione definita, va bene? , assicurati di entrarey.

5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query  
'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Richiama l'endpoint dell'API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere le tracce per la funzione, esegui [sam traces](#).

```
sam traces
```

L'output della traccia ha il seguente aspetto:

```
XRay Event [revision 1] at (2023-01-31T11:29:40.527000) with id  
(1-11a2222-111a222222cb33de3b95daf9) and duration (0.483s)  
- 0.425s - sam-app/Prod [HTTP: 200]  
- 0.422s - Lambda [HTTP: 200]  
- 0.406s - sam-app-HelloWorldFunction-XYZv11a1bcde [HTTP: 200]  
- 0.172s - sam-app-HelloWorldFunction-XYZv11a1bcde  
- 0.179s - Initialization  
- 0.112s - Invocation  
- 0.052s - ## app.lambdaHandler  
- 0.001s - ### MySubSegment  
- 0.059s - Overhead
```

8. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

X-Ray non traccia tutte le richieste alla tua applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento sia efficiente, fornendo comunque un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e

del 5% delle altre richieste. Non puoi configurare la frequenza di X-Ray campionamento per le tue funzioni.

Usare Powertools per AWS Lambda (TypeScript) e il AWS CDK per tracciare

Segui i passaggi seguenti per scaricare, creare e distribuire un' TypeScript applicazione Hello World di esempio con [moduli](#) Powertools for AWS Lambda (TypeScript) integrati utilizzando AWS CDK. Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format a e invia tracce a CloudWatch AWS X-Ray. La funzione restituisce un messaggio hello world.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Node.js
- [AWS CLI versione 2](#)
- [AWS CDK versione 2](#)
- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI. AWS SAM](#)

Distribuisci un esempio Kit di sviluppo per il cloud AWS (AWS CDK) applicazione

1. Crea una directory di progetto per la nuova applicazione.

```
mkdir hello-world  
cd hello-world
```

2. Inizializza l'app.

```
cdk init app --language typescript
```

3. Aggiungi il [pacchetto](#) `@types/aws-lambda` come dipendenza di sviluppo.

```
npm install -D @types/aws-lambda
```

4. Installa l'[utilità Tracer](#) di Powertools.



```
npm install @aws-lambda-powertools/tracer
```

5. Apri la directory lib. Dovrebbe essere visualizzato un file chiamato hello-world-stack.ts. Crea due nuovi file in questa directory: hello-world.function.ts e hello-world.ts.
6. Apri hello-world.function.ts e aggiungi il seguente codice al file. Questo è il codice per la funzione Lambda.

```
import { APIGatewayEvent, APIGatewayProxyResult, Context } from 'aws-lambda';
import { Tracer } from '@aws-lambda-powertools/tracer';
const tracer = new Tracer();

export const handler = async (event: APIGatewayEvent, context: Context):
  Promise<APIGatewayProxyResult> => {
  // Get facade segment created by Lambda
  const segment = tracer.getSegment();

  // Create subsegment for the function and set it as active
  const handlerSegment = segment.addNewSubsegment(`## ${process.env._HANDLER}`);
  tracer.setSegment(handlerSegment);

  // Annotate the subsegment with the cold start and serviceName
  tracer.annotateColdStart();
  tracer.addServiceNameAnnotation();

  // Add annotation for the awsRequestId
  tracer.putAnnotation('awsRequestId', context.awsRequestId);
  // Create another subsegment and set it as active
  const subsegment = handlerSegment.addNewSubsegment('### MySubSegment');
  tracer.setSegment(subsegment);
  let response: APIGatewayProxyResult = {
    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
    }),
  };
  // Close subsegments (the Lambda one is closed automatically)
  subsegment.close(); // (### MySubSegment)
  handlerSegment.close(); // (## index.handler)

  // Set the facade segment as active again (the one created by Lambda)
  tracer.setSegment(segment);
  return response;
}
```

```
};
```

7. Apri `hello-world.ts` e aggiungi il seguente codice al file. Contiene il [NodejsFunction costruito](#), che crea la funzione Lambda, configura le variabili di ambiente per Powertools e imposta la conservazione dei log su una settimana. Include anche il [LambdaRestApi costruito](#), che crea l'API REST.

```
import { Construct } from 'constructs';
import { NodejsFunction } from 'aws-cdk-lib/aws-lambda-nodejs';
import { LambdaRestApi } from 'aws-cdk-lib/aws-apigateway';
import { CfnOutput } from 'aws-cdk-lib';
import { Tracing } from 'aws-cdk-lib/aws-lambda';

export class HelloWorld extends Construct {
  constructor(scope: Construct, id: string) {
    super(scope, id);
    const helloFunction = new NodejsFunction(this, 'function', {
      environment: {
        POWERTOOLS_SERVICE_NAME: 'helloWorld',
      },
      tracing: Tracing.ACTIVE,
    });
    const api = new LambdaRestApi(this, 'apigw', {
      handler: helloFunction,
    });
    new CfnOutput(this, 'apiUrl', {
      exportName: 'apiUrl',
      value: api.url,
    });
  }
}
```

8. Apri `hello-world-stack.ts`. Questo è il codice che definisce lo [stack AWS CDK](#). Sostituisci il codice con il seguente:

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { HelloWorld } from './hello-world';

export class HelloWorldStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new HelloWorld(this, 'hello-world');
```

```
}  
}
```

9. Distribuisci l'applicazione.

```
cd ..  
cdk deploy
```

10. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query  
'Stacks[0].Outputs[?ExportName==`apiUrl`].OutputValue' --output text
```

11. Richiama l'endpoint dell'API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

12. Per ottenere le tracce per la funzione, esegui [sam traces](#).

```
sam traces
```

L'output della traccia ha il seguente aspetto:

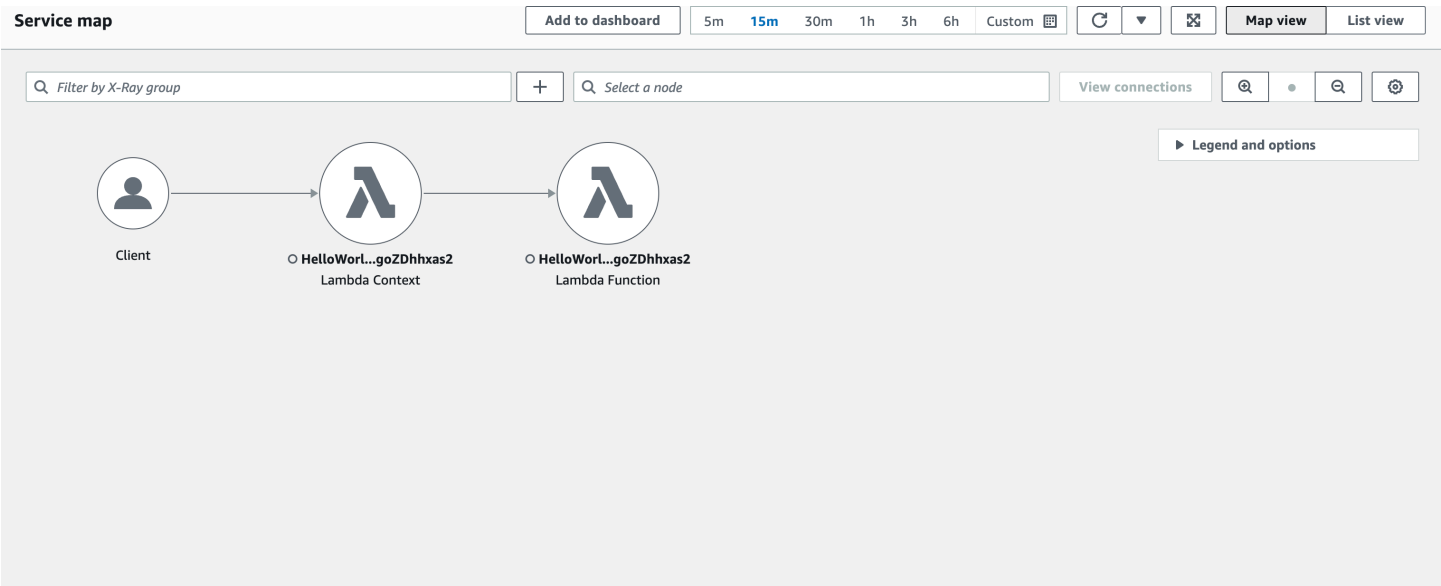
```
XRay Event [revision 1] at (2023-01-31T11:50:06.997000) with id  
(1-11a2222-111a22222cb33de3b95daf9) and duration (0.449s)  
- 0.350s - HelloWorldStack-helloworldfunction111A2BCD-Xyzv11a1bcde [HTTP: 200]  
- 0.157s - HelloWorldStack-helloworldfunction111A2BCD-Xyzv11a1bcde  
- 0.169s - Initialization  
- 0.058s - Invocation  
- 0.055s - ## index.handler  
- 0.000s - ### MySubSegment  
- 0.099s - Overhead
```

13. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
cdk destroy
```

## Interpretazione e tracciamento X-Ray

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. La mappa di [X-Ray tracciamento](#) fornisce informazioni sull'applicazione e su tutti i suoi componenti. L'esempio seguente mostra una traccia dall'applicazione di esempio:



## Compilazione di funzioni Lambda con Python

Puoi eseguire il codice Python in AWS Lambda. Lambda fornisce [Runtime](#) per Python che eseguono il tuo codice per elaborare gli eventi. Il codice viene eseguito in un ambiente che include l'SDK per Python (Boto3), con le credenziali di un ruolo (IAM) che gestisci. AWS Identity and Access Management Per ulteriori informazioni sulle versioni SDK incluse nei runtime Python, consulta [the section called “Runtime-included Versioni SDK”](#).

Lambda supporta i seguenti runtime di Python.

| Nome        | Identificatore | Sistema operativo | Data di ritiro  | Blocco creazione funzioni | Blocco aggiornam ento funzioni |
|-------------|----------------|-------------------|-----------------|---------------------------|--------------------------------|
| Python 3.15 | python3.15     | Amazon Linux 2023 | Non programmato | Non programmato           | Non programmato                |
| Python 3.14 | python3.14     | Amazon Linux 2023 | 30 giugno 2029  | 31 luglio 2029            | 31 agosto 2029                 |

| Nome        | Identificatore | Sistema operativo | Data di ritiro  | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|-------------|----------------|-------------------|-----------------|---------------------------|-------------------------------|
| Python 3.13 | python3.13     | Amazon Linux 2023 | 30 giugno 2029  | 31 luglio 2029            | 31 agosto 2029                |
| Python 3.12 | python3.12     | Amazon Linux 2023 | 31 ottobre 2028 | 30 novembre 2028          | 10 gennaio 2029               |
| Python 3.11 | python3.11     | Amazon Linux 2    | 30 giugno 2027  | 31 luglio 2027            | 31 agosto 2027                |
| Python 3.10 | python3.10     | Amazon Linux 2    | 31 ottobre 2026 | 1 febbraio 2027           | 3 marzo 2027                  |

Per creare una funzione Python

1. Aprire la [console Lambda](#).
2. Scegli Crea funzione.
3. Configura le impostazioni seguenti:
  - Nome della funzione: inserisci il nome della funzione.
  - Runtime: scegli Python 3.14.
4. Scegli Crea funzione.

La console crea una funzione Lambda con un singolo file di origine denominato `lambda_function`. È possibile modificare questo file e aggiungere altri file nell'editor di codice predefinito. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione. Quindi, per eseguire il codice, scegli Crea evento di test nella sezione EVENTI DI TEST.

La funzione Lambda è dotata di un gruppo di CloudWatch log Logs. Il runtime della funzione invia i dettagli su ogni chiamata a Logs. CloudWatch Si trasmette qualsiasi [log che la tua funzione emette](#) durante la chiamata. Se la funzione restituisce un errore, Lambda formatta l'errore e lo restituisce al chiamante.

Per indicazioni su come testare le tue funzioni Python, consulta [the section called “Test in corso”](#)

## Argomenti

- [Runtime-included Versioni SDK](#)
- [Funzionalità Python disattivate](#)
- [Formato della risposta](#)
- [Chiusura graduale per le estensioni](#)
- [Definire l'handler della funzione Lambda in Python](#)
- [Utilizzo di archivi di file .zip per le funzioni Lambda in Python](#)
- [Distribuire funzioni Lambda per Python con immagini di container](#)
- [Utilizzo dei livelli per le funzioni Lambda in Python](#)
- [Utilizzo dell'oggetto del contesto Lambda per recuperare le informazioni sulla funzione Python](#)
- [Registrare e monitorare le funzioni Lambda con Python](#)
- [AWS Lambda test delle funzioni in Python](#)
- [Strumentazione del codice Python in AWS Lambda](#)

## Runtime-included Versioni SDK

La versione dell' AWS SDK inclusa nel runtime di Python dipende dalla versione di runtime e dalla tua. Regione AWS Per trovare la versione dell'SDK incluso nel runtime che stai utilizzando, crea una funzione Lambda con il codice seguente.

```
import boto3
import botocore

def lambda_handler(event, context):
    print(f'boto3 version: {boto3.__version__}')
    print(f'botocore version: {botocore.__version__}')
```

## Funzionalità Python disattivate

La tabella seguente elenca le funzionalità di Python che sono disabilitate nei runtime gestiti da Lambda e nelle immagini di base dei contenitori per Python. Queste funzionalità devono essere abilitate quando l'eseguibile di runtime di Python viene compilato e non possono essere abilitate utilizzando un flag di esecuzione. Per utilizzare queste funzionalità in Lambda, puoi distribuire la tua build di runtime Python con queste funzionalità abilitate, utilizzando un'immagine contenitore o un runtime personalizzato. [??? ???](#)

| Funzionalità Python            | Versioni di Python interessate    | Stato                                                                                                                                                                                         |
|--------------------------------|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Just-in-Time compilatore (JIT) | Python 3.13 e versioni successive | Il compilatore JIT è sperimentale e non è consigliato per i carichi di lavoro di produzione. È quindi disabilitato nei runtime di Lambda Python.                                              |
| Free-threading                 | Python 3.13 e versioni successive | Il threading libero (opzione per disabilitare il blocco globale dell'interprete) è disabilitato nelle build Lambda Python a causa dell'impatto sulle prestazioni del codice a thread singolo. |

## Formato della risposta

Nei runtime Python 3.12 e successivi, le funzioni restituiscono caratteri Unicode come parte della loro risposta JSON. I runtime Python precedenti restituivano sequenze con escape per i caratteri Unicode nelle risposte. Ad esempio, in Python 3.11, se restituisci una stringa Unicode come "こんにちは", viene eseguito l'escape dei caratteri Unicode e restituito il valore "\u3053\u3093\u306b\u3061\u306f". Il runtime Python 3.12 restituisce il valore "こんにちは" originale.

L'utilizzo delle risposte Unicode riduce le dimensioni delle risposte Lambda e ciò facilita l'inserimento di risposte più grandi nella dimensione del payload massima di 6 MB per le funzioni sincrone. Nell'esempio precedente, la versione con escape è di 32 byte, rispetto ai 17 della stringa Unicode.

Quando esegui l'aggiornamento a Python 3.12 o a runtime Python successivi, potrebbe essere necessario modificare il codice tenendo conto del nuovo formato di risposta. Se il chiamante prevede un codice Unicode con escape, devi aggiungere manualmente il codice alla funzione restituita per eseguire l'escape dell'Unicode o regolare il chiamante per gestire la restituzione dell'Unicode.

## Chiusura graduale per le estensioni

I runtime Python 3.12 e versioni successive offrono funzionalità di arresto graduale migliorate per funzioni con [estensioni esterne](#). Quando chiude un ambiente di esecuzione, Lambda invia un segnale SIGTERM al runtime e quindi un evento SHUTDOWN a ogni estensione esterna registrata. È possibile catturare il segnale SIGTERM nella funzione Lambda e ripulire risorse come le connessioni al database create dalla funzione.

Per ulteriori informazioni sul ciclo di vita dell'ambiente di esecuzione, consulta [Comprendere il ciclo di vita dell'ambiente di esecuzione Lambda](#). Per esempi di come utilizzare Graceful shutdown con le estensioni, consulta il repository Samples. [AWS GitHub](#)



## Definire l'handler della funzione Lambda in Python

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come lavorare con i gestori di funzioni Lambda in Python, incluse le convenzioni di denominazione, le firme degli handler e le migliori pratiche. Questa pagina include anche un esempio di funzione Python Lambda che acquisisce informazioni su un ordine, produce una ricevuta in un file di testo e lo inserisce in un bucket Amazon Simple Storage Service (Amazon S3).

### Argomenti

- [Esempio di codice della funzione Lambda in Python](#)
- [Convenzioni di denominazione dei gestori](#)
- [Utilizzo dell'oggetto evento Lambda](#)
- [Accesso e utilizzo dell'oggetto contestuale Lambda](#)
- [Firme dell'handler valide per gli handler Python](#)
- [Restituzione di un valore](#)
- [Utilizzo di AWS SDK per Python \(Boto3\) nel tuo gestore](#)
- [Accesso alle variabili d'ambiente](#)
- [Best practice di codice per le funzioni Lambda con Python Lambda](#)

### Esempio di codice della funzione Lambda in Python

Il seguente esempio di codice della funzione Lambda in Python raccoglie le informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3.

### Example Funzione Python Lambda

```
import json
import os
import logging
import boto3

# Initialize the S3 client outside of the handler
s3_client = boto3.client('s3')

# Initialize the logger
```

```
logger = logging.getLogger()
logger.setLevel("INFO")

def upload_receipt_to_s3(bucket_name, key, receipt_content):
    """Helper function to upload receipt to S3"""

    try:
        s3_client.put_object(
            Bucket=bucket_name,
            Key=key,
            Body=receipt_content
        )
    except Exception as e:
        logger.error(f"Failed to upload receipt to S3: {str(e)}")
        raise

def lambda_handler(event, context):
    """
    Main Lambda handler function
    Parameters:
        event: Dict containing the Lambda function event data
        context: Lambda runtime context
    Returns:
        Dict containing status message
    """
    try:
        # Parse the input event
        order_id = event['Order_id']
        amount = event['Amount']
        item = event['Item']

        # Access environment variables
        bucket_name = os.environ.get('RECEIPT_BUCKET')
        if not bucket_name:
            raise ValueError("Missing required environment variable RECEIPT_BUCKET")

        # Create the receipt content and key destination
        receipt_content = (
            f"OrderID: {order_id}\n"
            f"Amount: ${amount}\n"
            f"Item: {item}"
        )
        key = f"receipts/{order_id}.txt"
```

```
# Upload the receipt to S3
upload_receipt_to_s3(bucket_name, key, receipt_content)

logger.info(f"Successfully processed order {order_id} and stored receipt in S3
bucket {bucket_name}")

return {
    "statusCode": 200,
    "message": "Receipt processed successfully"
}

except Exception as e:
    logger.error(f"Error processing order: {str(e)}")
    raise
```

Questo file contiene le sezioni seguenti:

- **Blocco import:** utilizza questo blocco per includere le librerie richieste dalla funzione Lambda.
- **Inizializzazione globale del client e del logger SDK:** l'inclusione del codice di inizializzazione all'esterno del gestore sfrutta il [riutilizzo dell'ambiente di esecuzione](#) per migliorare le prestazioni della funzione. Per ulteriori informazioni, consulta [the section called “Best practice di codice per le funzioni Lambda con Python Lambda”](#).
- **def upload\_receipt\_to\_s3(bucket\_name, key, receipt\_content):** Questa è una funzione di supporto chiamata dalla funzione principale. `lambda_handler`
- **def lambda\_handler(event, context):** Questa è la principale funzione di gestione del codice, che contiene la logica principale dell'applicazione. Quando Lambda richiama il gestore della funzione, il runtime [Lambda](#) passa due argomenti alla funzione, l'oggetto [evento](#) che contiene i dati da elaborare per la funzione e l'oggetto [context](#) che contiene informazioni sull'invocazione della funzione.

## Convenzioni di denominazione dei gestori

Il nome del gestore di funzioni definito al momento della creazione di una funzione Lambda è derivato da:

- il nome del file in cui si trova la funzione del gestore Lambda.
- il nome della funzione del gestore Python.

Nell'esempio precedente, se il file è denominato `lambda_function.py`, il gestore verrà specificato come `lambda_function.lambda_handler`. Questo è il nome predefinito del gestore assegnato alle funzioni create utilizzando la console Lambda.

Se si crea una funzione nella console utilizzando un nome di file o un nome del gestore di funzione diverso, è necessario modificare il nome del gestore predefinito.

#### Modifica del nome del gestore funzioni (console)

1. Apri la pagina [Funzioni](#) della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Scorri verso il basso fino al riquadro Impostazioni di runtime e scegli Modifica.
4. In Gestore, inserisci il nuovo nome per il tuo gestore di funzioni.
5. Scegli Save (Salva).

#### Utilizzo dell'oggetto evento Lambda

Quando Lambda richiama la funzione, passa un [argomento](#) dell'oggetto evento al gestore della funzione. Gli oggetti JSON sono il formato di evento più comune per le funzioni Lambda. Nell'esempio di codice della sezione precedente, la funzione prevede un input nel seguente formato:

```
{
  "Order_id": "12345",
  "Amount": 199.99,
  "Item": "Wireless Headphones"
}
```

Se la funzione viene richiamata da un'altra Servizio AWS, anche l'evento di input è un oggetto JSON. Il formato esatto dell'oggetto evento dipende dal servizio che richiama la funzione. Per vedere il formato dell'evento per un particolare servizio, fate riferimento alla pagina appropriata nel [the section called "Integrazione con altri servizi"](#) capitolo.

Se l'evento di input ha la forma di un oggetto JSON, il runtime Lambda converte l'oggetto in un dizionario Python. Per assegnare valori nel JSON di input alle variabili del codice, usa i metodi standard del dizionario Python come illustrato nel codice di esempio.

Puoi anche passare i dati alla tua funzione come array JSON o come qualsiasi altro tipo di dati JSON valido. La tabella seguente definisce come il runtime Python converte questi tipi JSON.

| Tipo di dati JSON | Tipo di dati Python                                     |
|-------------------|---------------------------------------------------------|
| oggetto           | dizionario () dict                                      |
| array             | lista (list)                                            |
| numero            | numero intero (int) o numero in virgola mobile () float |
| stringa           | stringa () str                                          |
| Booleano          | Booleano () bool                                        |
| null              | NoneType (NoneType)                                     |

## Accesso e utilizzo dell'oggetto contestuale Lambda

L'oggetto di contesto Lambda contiene informazioni sull'ambiente di invocazione ed esecuzione della funzione. Lambda passa automaticamente l'oggetto di contesto alla funzione quando viene richiamata. Puoi utilizzare l'oggetto contestuale per generare informazioni sull'invocazione della funzione a scopo di monitoraggio.

L'oggetto di contesto è una classe Python definita nel client dell'interfaccia di runtime [Lambda](#). Per restituire il valore di una qualsiasi proprietà dell'oggetto di contesto, usa il metodo corrispondente sull'oggetto di contesto. Ad esempio, il seguente frammento di codice assegna il valore della `aws_request_id` proprietà (l'identificatore per la richiesta di invocazione) a una variabile denominata `request`

```
request = context.aws_request_id
```

Per saperne di più sull'uso dell'oggetto di contesto Lambda e per visualizzare un elenco completo dei metodi e delle proprietà disponibili, consulta [the section called “Contesto”](#)

## Firme dell'handler valide per gli handler Python

Quando si definisce la funzione handler in Python, la funzione deve accettare due argomenti. Il primo di questi argomenti è l'oggetto [evento Lambda](#) e il secondo è l'oggetto di contesto [Lambda](#). Per convenzione, questi argomenti di input sono generalmente denominati `event` e `context`, ma puoi dare loro qualsiasi nome desideri. Se dichiari la tua funzione handler con un singolo argomento

di input, Lambda genera un errore quando tenta di eseguire la funzione. Il modo più comune per dichiarare una funzione di gestione in Python è il seguente:

```
def lambda_handler(event, context):
```

Puoi anche usare i suggerimenti sui tipi di Python nella dichiarazione della funzione, come mostrato nell'esempio seguente:

```
from typing import Dict, Any

def lambda_handler(event: Dict[str, Any], context: Any) -> Dict[str, Any]:
```

Per utilizzare una AWS digitazione specifica per gli eventi generati da altri Servizi AWS e per l'oggetto context, aggiungete il `aws-lambda-typing` pacchetto al pacchetto di distribuzione della funzione. Puoi installare questa libreria nel tuo ambiente di sviluppo eseguendo. **`pip install aws-lambda-typing`** Il seguente frammento di codice mostra come utilizzare i suggerimenti di tipo AWS specifici. In questo esempio, l'evento previsto è un evento Amazon S3.

```
from aws_lambda_typing.events import S3Event
from aws_lambda_typing.context import Context
from typing import Dict, Any

def lambda_handler(event: S3Event, context: Context) -> Dict[str, Any]:
```

Non puoi usare il tipo di funzione Python per la tua `async` funzione handler.

### Restituzione di un valore

Facoltativamente, un gestore può restituire un valore, che deve essere serializzabile in formato JSON. I tipi di restituzione più comuni includono `dict`, `list`, `str`, `int`, `float`, `bool`

Ciò che accade al valore restituito dipende dal [tipo di chiamata](#) e dal [servizio](#) che ha invocato la funzione. Ad esempio:

- Se si utilizza il tipo di `RequestResponse` invocazione per [richiamare una funzione Lambda in modo sincrono](#), Lambda restituisce il risultato della chiamata alla funzione Python al client che richiama la funzione Lambda (nella risposta HTTP alla richiesta di invocazione, serializzata in JSON). Ad esempio, la AWS Lambda console utilizza il tipo di `RequestResponse` invocazione, quindi quando si richiama la funzione sulla console, la console visualizza il valore restituito.

- Se il gestore restituisce degli oggetti che non possono essere serializzati da `json.dumps`, il runtime restituisce un errore.
- Se il gestore restituisce `None`, come fanno implicitamente le funzioni Python senza un'istruzione `return`, il runtime restituisce `null`.
- Se utilizzi il tipo di invocazione Event, ovvero una [invocazione asincrona](#), il valore viene ignorato.

Nel codice di esempio, il gestore restituisce il seguente dizionario Python:

```
{  
  "statusCode": 200,  
  "message": "Receipt processed successfully"  
}
```

Il runtime Lambda serializza questo dizionario e lo restituisce al client che ha richiamato la funzione come stringa JSON.

#### Note

In Python 3.9 e nelle versioni successive, Lambda include il `requestId` dell'invocazione nella risposta di errore.

## Utilizzo di AWS SDK per Python (Boto3) nel tuo gestore

Spesso, utilizzerai le funzioni Lambda per interagire con altre risorse Servizi AWS. Il modo più semplice per interfacciarsi con queste risorse è usare la AWS SDK per Python (Boto3). Tutti i runtime Lambda Python [supportati](#) includono una versione dell'SDK per Python.

Tuttavia, consigliamo vivamente di includere l'SDK nel pacchetto di distribuzione della funzione se il codice deve utilizzarlo. L'inclusione dell'SDK nel pacchetto di distribuzione ti dà il pieno controllo sulle dipendenze e riduce il rischio di problemi di disallineamento della versione con altre librerie. Per ulteriori informazioni, consulta le pagine [the section called “Dipendenze di runtime in Python”](#) e [the section called “Compatibilità con le versioni precedenti”](#).

Per utilizzare l'SDK per Python nella tua funzione Lambda, aggiungi la seguente dichiarazione al blocco di importazione all'inizio del codice della funzione:

```
import boto3
```

Usa il `pip install` comando per aggiungere la `boto3` libreria al pacchetto di distribuzione della tua funzione. Per istruzioni dettagliate su come aggiungere dipendenze a un pacchetto di distribuzione .zip, consulta [the section called “Creazione di un pacchetto di implementazione .zip con dipendenze”](#) Per ulteriori informazioni sull'aggiunta di dipendenze alle funzioni Lambda distribuite come immagini di container, consulta o [the section called “Creazione di un'immagine da un'immagine di base”](#) [the section called “Creazione di un'immagine da un'immagine di base alternativa”](#)

Quando si utilizza `boto3` il codice, non è necessario fornire alcuna credenziale per inizializzare un client. Ad esempio, nel codice di esempio, utilizziamo la seguente riga di codice per inizializzare un client Amazon S3:

```
# Initialize the S3 client outside of the handler
s3_client = boto3.client('s3')
```

Con Python, Lambda crea automaticamente variabili di ambiente con credenziali. L'`boto3` SDK verifica la presenza di queste credenziali nelle variabili di ambiente della funzione durante l'inizializzazione.

### Accesso alle variabili d'ambiente

Nel codice del gestore, puoi fare riferimento alle variabili [di](#) ambiente utilizzando il metodo `os.environ.get` Nel codice di esempio, facciamo riferimento alla variabile di `RECEIPT_BUCKET` ambiente definita utilizzando la seguente riga di codice:

```
# Access environment variables
bucket_name = os.environ.get('RECEIPT_BUCKET')
```

Non dimenticate di includere un `import os` istruzione nel blocco di importazione all'inizio del codice.

### Best practice di codice per le funzioni Lambda con Python Lambda

Segui le linee guida riportate nell'elenco seguente per utilizzare le best practice di codifica durante la creazione delle funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. Ciò semplifica la creazione di una funzione più testabile da un'unità. Ad esempio, in Python, potrebbe assomigliare a:

```
def lambda_handler(event, context):
    foo = event['foo']
    bar = event['bar']
    result = my_lambda_function(foo, bar)
```



```
def my_lambda_function(foo, bar):  
    // MyLambdaFunction logic here
```

- Controlla le dipendenze nel pacchetto di distribuzione della tua funzione. L'ambiente di esecuzione AWS Lambda contiene diverse librerie. Per i runtime di Python Node.js e Python, questi includono gli AWS SDK.

Per abilitare l'ultimo set di funzionalità e aggiornamenti di sicurezza, Lambda aggiorna periodicamente queste librerie. Questi aggiornamenti potrebbero introdurre lievi modifiche al comportamento della funzione Lambda. Per mantenere il controllo completo delle dipendenze utilizzate dalla funzione, inserire tutte le dipendenze nel pacchetto di implementazione.

- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del [contesto di esecuzione](#).
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. Ciò riduce il tempo necessario per scaricare e decomprimere il pacchetto di distribuzione prima dell'invocazione.

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory /tmp. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedi [Riutilizzo delle connessioni con in. Keep-Alive Node.js](#)

Utilizzare [le variabili di ambiente](#) per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su 0 per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionali alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha una dipendenza su queste API non pubbliche. Consulta il [riferimento all'API](#) per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta [Come posso rendere idempotente la mia funzione Lambda?](#).

## Utilizzo di archivi di file .zip per le funzioni Lambda in Python

Il codice della AWS Lambda funzione è costituito da un file .py contenente il codice del gestore della funzione, insieme a eventuali pacchetti e moduli aggiuntivi da cui dipende il codice. Per implementare questo codice della funzione in Lambda, utilizza un pacchetto di implementazione. Questo pacchetto può essere un archivio di file .zip o un'immagine di container. Per ulteriori informazioni sull'uso delle immagini di container con Python, consulta la sezione [Implementazione di funzioni Lambda in Python con immagini di container](#).

Per creare un pacchetto di implementazione come archivio di file .zip, puoi utilizzare l'utilità di archiviazione di file .zip incorporata del tuo strumento della linea di comando o qualsiasi altra utilità file .zip, come ad esempio [7zip](#). Gli esempi mostrati nelle sezioni seguenti presuppongono che tu stia utilizzando uno strumento della linea di comando zip in un ambiente Linux o MacOS. Per utilizzare gli stessi comandi in Windows, puoi [installare il sottosistema Windows per Linux per](#) ottenere una Windows-integrated versione di Ubuntu e Bash.

Nota che Lambda utilizza le autorizzazioni dei file POSIX, quindi potresti aver bisogno di [impostare le autorizzazioni per la cartella del pacchetto di implementazione](#) prima di creare l'archivio di file .zip.

### Argomenti

- [Dipendenze di runtime in Python](#)
- [Creazione di un pacchetto di implementazione .zip senza dipendenze](#)
- [Creazione di un pacchetto di implementazione .zip con dipendenze](#)
- [Percorso di ricerca delle dipendenze e librerie incluse nel runtime](#)
- [Utilizzo delle cartelle \\_\\_pycache\\_\\_](#)
- [Creazione di un pacchetto di implementazione .zip con librerie native](#)
- [Creazione e aggiornamento delle funzioni Lambda di Python utilizzando file .zip](#)

### Dipendenze di runtime in Python

Per le funzioni Lambda che utilizzano il runtime Python, una dipendenza può essere qualsiasi pacchetto o modulo Python. Se implementi la funzione utilizzando un archivio .zip, puoi aggiungere queste dipendenze al file .zip con il tuo codice della funzione o utilizzare un [livello Lambda](#). Un livello è un file .zip separato che può contenere codice aggiuntivo o altri contenuti. Per ulteriori informazioni sull'uso dei livelli Lambda in Python, consulta [the section called “Livelli”](#).

I runtime di Lambda Python includono e le relative dipendenze. AWS SDK per Python (Boto3) Lambda fornisce l'SDK nel runtime per scenari di implementazione in cui non è possibile aggiungere le proprie dipendenze. Questi scenari includono la creazione di funzioni nella console utilizzando l'editor di codice integrato o l'utilizzo di funzioni in linea in AWS Serverless Application Model () o modelli.AWS SAM CloudFormation

Lambda aggiorna periodicamente le librerie nel runtime Python per includere gli aggiornamenti e le patch di sicurezza più recenti. Se la funzione utilizza la versione dell'SDK Boto3 inclusa nel runtime ma il pacchetto di implementazione include dipendenze SDK, ciò può causare problemi di disallineamento delle versioni. Ad esempio, il pacchetto di implementazione potrebbe includere la dipendenza SDK urllib3. Quando Lambda aggiorna l'SDK nel runtime, i problemi di compatibilità tra la nuova versione del runtime e la versione di urllib3 nel pacchetto di implementazione possono causare il fallimento della funzione.

#### Important

Per mantenere il pieno controllo sulle dipendenze ed evitare possibili problemi di disallineamento delle versioni, ti consigliamo di aggiungere tutte le dipendenze della funzione al pacchetto di distribuzione, anche se le relative versioni sono incluse nel runtime Lambda. Ciò include l'SDK Boto3.

Per scoprire quale versione dell'SDK per Python (Boto3) è inclusa nel runtime che stai utilizzando, consulta [the section called “Runtime-included Versions SDK”](#).

In base al [modello di responsabilità condivisa di AWS](#), è tua responsabilità gestire eventuali dipendenze nei pacchetti di implementazione delle tue funzioni. Ciò include l'applicazione di aggiornamenti e patch di sicurezza. Per aggiornare le dipendenze nel pacchetto di implementazione della funzione, crea prima un nuovo file .zip e poi caricalo su Lambda. Per ulteriori informazioni, consulta [Creazione di un pacchetto di implementazione .zip con dipendenze](#) e [Creazione e aggiornamento delle funzioni Lambda di Python utilizzando file .zip](#).

#### Creazione di un pacchetto di implementazione .zip senza dipendenze

Se il codice della funzione non ha dipendenze, il file .zip contiene solo il file .py con il codice del gestore della funzione. Usa la tua utilità zip preferita per creare un file .zip con il file .py nella directory principale. Se il file .py non è alla radice del file .zip, Lambda non sarà in grado di eseguire il codice.

Per informazioni su come implementare il file .zip per creare una nuova funzione Lambda o aggiornarne una esistente, consulta la sezione [Creazione e aggiornamento delle funzioni Lambda di Python utilizzando file .zip](#).

## Creazione di un pacchetto di implementazione .zip con dipendenze

Se il tuo codice della funzione dipende da pacchetti o moduli aggiuntivi, puoi aggiungere queste dipendenze al file .zip con il codice della funzione oppure [utilizzare un livello Lambda](#). Le istruzioni in questa sezione mostrano come includere le dipendenze nel pacchetto di implementazione .zip. Affinché Lambda esegua il codice, il file .py contenente il codice del gestore e tutte le dipendenze della funzione deve essere installato nella radice del file .zip.

Supponiamo che il codice della funzione sia salvato in un file denominato `lambda_function.py`. I seguenti comandi della CLI di esempio creano un file .zip denominato `my_deployment_package.zip` contenente il codice della funzione e le relative dipendenze. Puoi installare le tue dipendenze direttamente in una cartella nella directory del tuo progetto o utilizzare un ambiente virtuale Python.

### Creazione del pacchetto di implementazione (directory del progetto)

1. Passa alla directory del progetto contenente il file del codice sorgente `lambda_function.py`. In questo esempio, la directory è denominata `my_function`.

```
cd my_function
```

2. Crea una nuova directory denominata "pacchetto" in cui installare le tue dipendenze.

```
mkdir package
```

Tieni presente che per un pacchetto di implementazione .zip, Lambda prevede che il codice sorgente e le relative dipendenze siano tutti nella directory principale del file .zip. Tuttavia, l'installazione delle dipendenze direttamente nella directory del progetto può introdurre un gran numero di nuovi file e cartelle e rendere difficile la navigazione nell'IDE. Qui crei una directory `package` separata per mantenere le dipendenze separate dal codice sorgente.

3. Installa le dipendenze nella directory `package`. L'esempio seguente installa l'SDK Boto3 da Python Package Index utilizzando pip. Se il codice della funzione utilizza pacchetti Python creati da te, salvali nella directory `package`.

```
pip install --target ./package boto3
```

4. Crea un file `.zip` con le librerie installate nella directory principale.

```
cd package  
zip -r ../my_deployment_package.zip .
```

Verrà generato un file `my_deployment_package.zip` nella directory del progetto.

5. Aggiunta del file `lambda_function.py` alla directory principale del file `.zip`

```
cd ..  
zip my_deployment_package.zip lambda_function.py
```

Il tuo file `.zip` dovrebbe avere una struttura di directory semplice, con il codice del gestore della funzione e tutte le cartelle delle dipendenze installate nella directory principale come segue.

```
my_deployment_package.zip  
|- bin  
|  |-jp.py  
|- boto3  
|  |-compat.py  
|  |-data  
|  |-docs  
...  
|- lambda_function.py
```

Se il file `.py` contenente il codice del gestore della funzione non è alla radice del file `.zip`, Lambda non può eseguire il codice.

## Creazione del pacchetto di implementazione (ambiente virtuale)

1. Crea e attiva un ambiente virtuale nella directory del progetto. In questo esempio, la directory del progetto è denominata `my_function`.

```
~$ cd my_function  
~/my_function$ python3.14 -m venv my_virtual_env  
~/my_function$ source ./my_virtual_env/bin/activate
```

2. Installa le librerie richieste utilizzando pip. Nell'esempio seguente viene installato l'SDK Boto3

```
(my_virtual_env) ~/my_function$ pip install boto3
```

3. Utilizza `pip show` per trovare la posizione, all'interno del tuo ambiente virtuale, in cui pip ha installato le tue dipendenze.

```
(my_virtual_env) ~/my_function$ pip show <package_name>
```

La cartella in cui pip installa le tue librerie potrebbe essere denominata o. `site-packages` `dist-packages` Questa cartella potrebbe trovarsi nella `lib64/python3.x directory lib/python3.x` or (dove `python3.x` rappresenta la versione di Python che stai usando).

4. Disattivazione dell'ambiente virtuale

```
(my_virtual_env) ~/my_function$ deactivate
```

5. Accedi alla directory contenente le dipendenze che hai installato con pip e crea un file `.zip` nella directory principale del tuo progetto, nella quale sono installate le dipendenze. In questo esempio, pip ha installato le tue dipendenze nella directory `my_virtual_env/lib/python3.14/site-packages`.

```
~/my_function$ cd my_virtual_env/lib/python3.14/site-packages  
~/my_function/my_virtual_env/lib/python3.14/site-packages$ zip -r ../../../../  
my_deployment_package.zip .
```

6. Vai alla directory principale del tuo progetto, in cui si trova il file `.py` contenente il codice del gestore, e aggiungi quel file alla directory principale del tuo pacchetto `.zip`. In questo esempio, il file di codice della funzione è denominato `lambda_function.py`.

```
~/my_function/my_virtual_env/lib/python3.14/site-packages$ cd ../../../../  
~/my_function$ zip my_deployment_package.zip lambda_function.py
```

## Percorso di ricerca delle dipendenze e librerie incluse nel runtime

Quando utilizzi un'istruzione `import` nel codice, il runtime Python cerca nelle directory del suo percorso di ricerca finché non trova il modulo o il pacchetto. Per impostazione predefinita, la prima posizione cercata dal runtime è la directory in cui il pacchetto di implementazione `.zip` viene decompresso e montato (`/var/task`). Se includi una versione di una libreria inclusa nel runtime

nel pacchetto di distribuzione, la tua versione ha la precedenza sulla versione inclusa nel runtime. Le dipendenze nel pacchetto di implementazione hanno la precedenza anche sulle dipendenze nei livelli.

Quando aggiungi una dipendenza a un livello, Lambda la estrae in `/opt/python/lib/python3.x/site-packages`, dove `python3.x` rappresenta la versione del runtime che stai utilizzando, o `/opt/python`. Nel percorso di ricerca, queste directory hanno la precedenza sulle directory contenenti le librerie incluse nel runtime e le librerie installate con pip (`/var/runtime` e `/var/lang/lib/python3.x/site-packages`). Le librerie nei livelli di funzione hanno quindi la precedenza sulle versioni incluse nel runtime.

#### Note

Nel runtime gestito da Python 3.11 e nell'immagine di base, l' AWS SDK e le relative dipendenze sono installati nella directory. `/var/lang/lib/python3.11/site-packages`

Puoi visualizzare il percorso di ricerca completo per la tua funzione Lambda aggiungendo il seguente frammento di codice.

```
import sys

search_path = sys.path
print(search_path)
```

#### Note

Poiché le dipendenze nei livelli o nel pacchetto di implementazione hanno la precedenza sulle librerie incluse nel runtime, ciò può causare problemi di disallineamento delle versioni se si include una dipendenza SDK come `urllib3` nel pacchetto senza includere anche l'SDK. Se implementi la tua versione di una dipendenza `Boto3`, devi anche implementare `Boto3` come dipendenza nel tuo pacchetto di implementazione. Ti consigliamo di impacchettare tutte le dipendenze della tua funzione, anche se le relative versioni sono incluse nel runtime.

Puoi anche aggiungere dipendenze in una cartella separata all'interno del tuo pacchetto `.zip`. Ad esempio, potresti aggiungere una versione dell'SDK `Boto3` a una cartella del tuo pacchetto `.zip` chiamata `common`. Quando il pacchetto `.zip` viene decompresso e montato, questa cartella viene



inserita nella directory `/var/task`. Per utilizzare nel codice una dipendenza da una cartella del pacchetto di implementazione `.zip`, utilizza un'istruzione `import from`. Ad esempio, per utilizzare una versione di Boto3 da una cartella denominata `common` nel tuo pacchetto `.zip`, usa la seguente istruzione.

```
from common import boto3
```

### Utilizzo delle cartelle `__pycache__`

È consigliabile non includere cartelle `__pycache__` nel pacchetto di implementazione della funzione. Il bytecode Python compilato su un computer di compilazione con un'architettura o un sistema operativo diverso potrebbe non essere compatibile con l'ambiente di esecuzione Lambda.

### Creazione di un pacchetto di implementazione `.zip` con librerie native

Se la tua funzione utilizza solo pacchetti e moduli Python puri, puoi usare il comando `pip install` per installare le tue dipendenze su qualsiasi computer di compilazione locale e creare il file `.zip`. Molte delle librerie Python più diffuse, tra cui NumPy Pandas, non sono Python puro e contengono codice scritto in C o C++. Quando aggiungi librerie contenenti codice C/C++ al tuo pacchetto di distribuzione, devi creare il pacchetto correttamente per assicurarti che sia compatibile con l'ambiente di esecuzione Lambda.

La maggior parte dei pacchetti disponibili sul Python Package Index ([PyPI](#)) sono disponibili come «wheels» (file `.whl`). Un file `.whl` è un tipo di file ZIP che contiene una distribuzione compilata con file binari precompilati per un particolare sistema operativo e un'architettura di set di istruzioni. Per rendere il pacchetto di distribuzione compatibile con Lambda, installi il wheel per i sistemi operativi Linux e l'architettura del set di istruzioni della tua funzione.

Alcuni pacchetti possono essere disponibili solo come distribuzioni di origine. Per questi pacchetti, devi compilare e creare tu stesso i componenti C/C++.

Per vedere quali distribuzioni sono disponibili per il pacchetto richiesto, procedi come segue:

1. Cerca il nome del pacchetto nella [pagina principale di Python Package Index](#).
2. Seleziona la versione del pacchetto da utilizzare.
3. Scegli Scarica file.

### Utilizzo di distribuzioni integrate (wheel)

Per scaricare una ruota compatibile con Lambda, usi l'opzione `--platform pip`.

Se la tua funzione Lambda utilizza l'architettura del set di istruzioni x86\_64, esegui il comando `pip install` seguente per installare un wheel compatibile nella tua directory package. Sostituisci `--python 3.x` con la versione del runtime Python che stai utilizzando.

```
pip install \
--platform manylinux2014_x86_64 \
--target=package \
--implementation cp \
--python-version 3.x \
--only-binary=:all: --upgrade \
<package_name>
```

Se la funzione utilizza l'architettura del set di istruzioni arm64, esegui il seguente comando. Sostituisci `--python 3.x` con la versione del runtime Python che stai utilizzando.

```
pip install \
--platform manylinux2014_aarch64 \
--target=package \
--implementation cp \
--python-version 3.x \
--only-binary=:all: --upgrade \
<package_name>
```

## Utilizzo delle distribuzioni di origine

Se il tuo pacchetto è disponibile solo come distribuzione sorgente, devi creare tu stesso le librerie C/C++. Per rendere il pacchetto compatibile con l'ambiente di esecuzione Lambda, devi compilarlo in un ambiente che utilizzi lo stesso sistema operativo Amazon Linux. Puoi farlo creando il tuo pacchetto in un'istanza Linux Amazon Elastic Compute Cloud (Amazon EC2).

Per informazioni su come avviare e connettersi a un'istanza Amazon EC2 Linux, consulta la Guida introduttiva [ad Amazon EC2](#) nella Amazon EC2 User Guide.

## Creazione e aggiornamento delle funzioni Lambda di Python utilizzando file .zip

Dopo aver creato il pacchetto di implementazione .zip, puoi utilizzarlo per creare una nuova funzione Lambda o aggiornarne una esistente. Puoi distribuire il tuo pacchetto .zip utilizzando la console Lambda, l'API Lambda e l'AWS Command Line Interface API Lambda. Puoi anche creare e aggiornare le funzioni Lambda usando AWS Serverless Application Model (AWS SAM) e CloudFormation.

La dimensione massima per un pacchetto di implementazione .zip per Lambda è di 250 MB (dopo l'estrazione). Nota che questo limite si applica alla dimensione combinata di tutti i file caricati, inclusi eventuali livelli Lambda.

Il runtime Lambda necessita dell'autorizzazione per leggere i file nel pacchetto di distribuzione. Nella notazione ottale delle autorizzazioni di Linux, Lambda richiede 644 autorizzazioni per i file non eseguibili (rw-r--r--) e 755 autorizzazioni (rwxr-xr-x) per le directory e i file eseguibili.

In Linux e macOS, utilizza il comando `chmod` per modificare le autorizzazioni file su file e directory nel pacchetto di implementazione. Ad esempio, per assegnare a un file non eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 644 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta [Set, View, Change, or Remove Permissions on an Object](#) nella documentazione di Microsoft Windows.

#### Note

Se non concedi a Lambda le autorizzazioni necessarie per accedere alle directory del pacchetto di implementazione, Lambda imposta le autorizzazioni per tali directory su 755 (rwxr-xr-x).

## Creazione e aggiornamento delle funzioni con file .zip utilizzando la console

Per creare una nuova funzione, devi prima creare la funzione nella console, quindi devi caricare il tuo archivio .zip. Per aggiornare una funzione esistente, apri la pagina relativa alla funzione, quindi segui la stessa procedura per aggiungere il file .zip aggiornato.

Se il file .zip ha dimensioni inferiori a 50 MB, è possibile creare o aggiornare una funzione caricando il file direttamente dal computer locale. Per i file .zip di dimensioni superiori a 50 MB, prima è necessario caricare il pacchetto in un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il Console di gestione AWS, consulta [Guida introduttiva ad Amazon S3](#). Per caricare file utilizzando il AWS CLI, consulta [Sposta oggetti](#) nella Guida per l'AWS CLI utente.

 Note

Non è possibile modificare il [tipo di pacchetto di implementazione](#) (.zip o immagine di container) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio di file .zip. È necessario creare una nuova funzione.

## Creazione di una nuova funzione (console)

1. Apri la [pagina Funzioni](#) della console Lambda e scegli Crea funzione.
2. Scegli Author from scratch (Crea da zero).
3. In Basic information (Informazioni di base) eseguire queste operazioni:
  - a. In Nome funzione, inserisci il nome della funzione.
  - b. Per Runtime, seleziona il runtime che desideri utilizzare.
  - c. (Facoltativo) Per Architettura, scegli l'architettura del set di istruzioni per la funzione. L'architettura predefinita è x86\_64. Assicurati che il pacchetto di implementazione per la tua funzione sia compatibile con l'architettura del set di istruzioni scelta.
4. (Opzionale) In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito. Puoi creare un nuovo ruolo di esecuzione o utilizzare un ruolo esistente.
5. Scegli Crea funzione. Lambda crea una funzione di base "Hello world" utilizzando il runtime scelto.

## Caricamento di un archivio .zip dal computer locale (console)

1. Nella [pagina Funzioni](#) della console Lambda, scegli la funzione per cui vuoi caricare il file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli File .zip.
5. Per caricare il file .zip, procedi come segue:
  - a. Seleziona Carica, quindi seleziona il tuo file .zip nel selettore di file.
  - b. Seleziona Apri.
  - c. Scegli Save (Salva).

## Caricamento di un archivio .zip da un bucket Amazon S3 (console)

1. Nella [pagina Funzioni](#) della console Lambda, scegli la funzione per cui vuoi caricare un nuovo file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli Posizione Amazon S3.
5. Incolla l'URL del link Amazon S3 del tuo file .zip e scegli Salva.

## Aggiornamento delle funzioni dei file .zip tramite l'editor di codice della console

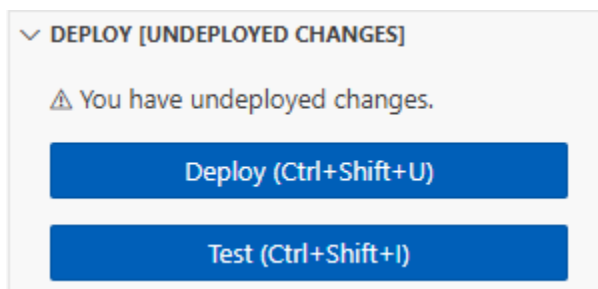
Per alcune funzioni con pacchetti di implementazione .zip, puoi utilizzare l'editor di codice integrato nella console Lambda per aggiornare direttamente il codice della funzione. Per utilizzare questa funzione, la funzione deve soddisfare i seguenti criteri:

- La funzione deve utilizzare uno dei runtime del linguaggio interpretato (Python o Node.js Ruby)
- Il pacchetto di implementazione della funzione deve avere dimensioni inferiori a 50 MB (non compresso).

Il codice della funzione per le funzioni con pacchetti di implementazione di immagini di container non può essere modificato direttamente nella console.

## Aggiornamento del codice della funzione utilizzando l'editor di codice della console

1. Apri la [pagina Funzioni](#) della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, seleziona il tuo file di codice sorgente e modificalo nell'editor di codice integrato.
4. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



## Creazione e aggiornamento di funzioni con file .zip utilizzando AWS CLI

È possibile utilizzare la [AWS CLI](#) per creare una nuova funzione o aggiornare una funzione esistente mediante un file .zip. Utilizza i comandi [create-function](#) e [update-function-code](#) per implementare il tuo pacchetto .zip. Se il file .zip ha dimensioni inferiori a 50 MB, è possibile caricare il pacchetto .zip da una posizione di file nella macchina di compilazione locale. Per i file di dimensioni maggiori, è necessario caricare il pacchetto .zip da un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il AWS CLI, consulta [Move objects](#) nella Guida per l'AWS CLI utente.

### Note

Se carichi il tuo file .zip da un bucket Amazon S3 utilizzando il AWS CLI, il bucket deve trovarsi nella stessa posizione della tua funzione. Regione AWS

Per creare una nuova funzione utilizzando un file .zip con AWS CLI, devi specificare quanto segue:

- Il nome della funzione (`--function-name`)
- Il runtime della tua funzione (`--runtime`)
- Il nome della risorsa Amazon (ARN) del [ruolo di esecuzione](#) della funzione (`--role`)
- Il nome del metodo del gestore nel codice della funzione (`--handler`)

È inoltre necessario specificare la posizione del file .zip. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda create-function --function-name myFunction \  
--runtime python3.14 --handler lambda_function.lambda_handler \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza l'opzione `--code` illustrata nel seguente comando di esempio. È necessario utilizzare il parametro `S3ObjectVersion` solo per gli oggetti con controllo delle versioni.

```
aws lambda create-function --function-name myFunction \  
--runtime python3.14 --handler lambda_function.lambda_handler \  
--code s3://my-bucket/my-function.zip
```

```
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--code S3Bucket=amzn-s3-demo-  
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Per aggiornare una funzione esistente mediante la CLI, specifica il nome della funzione utilizzando il parametro `--function-name`. È inoltre necessario specificare la posizione del file `.zip` che desideri utilizzare per aggiornare il codice della funzione. Se il file `.zip` si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda update-function-code --function-name myFunction \  
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file `.zip` in un bucket Amazon S3, utilizza le opzioni `--s3-bucket` e `--s3-key` come illustrato nel seguente comando di esempio. È necessario utilizzare il parametro `--s3-object-version` solo per gli oggetti con controllo delle versioni.

```
aws lambda update-function-code --function-name myFunction \  
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject  
Version
```

## Creazione e aggiornamento delle funzioni con file `.zip` utilizzando l'API Lambda

Per creare e aggiornare le funzioni mediante un archivio di file `.zip`, utilizza le seguenti operazioni API:

- [CreateFunction](#)
- [UpdateFunctionCode](#)

## Creazione e aggiornamento di funzioni con file `.zip` utilizzando AWS SAM

AWS Serverless Application Model (AWS SAM) è un toolkit che aiuta a semplificare il processo di creazione ed esecuzione di applicazioni serverless su AWS. Definisci le risorse per la tua applicazione in un modello YAML o JSON e usi l'interfaccia a riga di AWS SAM comando (AWS SAM CLI) per creare, impacchettare e distribuire le tue applicazioni. Quando crei una funzione Lambda da un AWS SAM modello, crea AWS SAM automaticamente un pacchetto di distribuzione `.zip` o un'immagine del contenitore con il codice della funzione e le dipendenze specificate. Per saperne di più sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda, consulta la [Guida introduttiva nella Developer Guide. AWS SAM](#) AWS Serverless Application Model

Puoi anche usare AWS SAM per creare una funzione Lambda utilizzando un archivio di file .zip esistente. Per creare una funzione Lambda utilizzando AWS SAM, puoi salvare il tuo file .zip in un bucket Amazon S3 o in una cartella locale sulla macchina di compilazione. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il AWS CLI, consulta [Move objects nella Guida per l'utente](#). AWS CLI

Nel AWS SAM modello, la `AWS::Serverless::Function` risorsa specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `CodeUri`- impostato sull'URI Amazon S3, sul percorso della cartella locale o sull'oggetto del codice funzione [FunctionCode](#)
- `Runtime`: imposta il runtime prescelto

Con AWS SAM, se il tuo file .zip è più grande di 50 MB, non devi prima caricarlo su un bucket Amazon S3. AWS SAM puoi caricare pacchetti .zip fino alla dimensione massima consentita di 250 MB (decompressi) da una posizione sul computer di compilazione locale.

Per saperne di più sulla distribuzione delle funzioni utilizzando il file .zip in AWS SAM, consulta [AWS::Serverless::Function nella Developer Guide](#). AWS SAM

Creazione e aggiornamento di funzioni con file .zip utilizzando CloudFormation

È possibile utilizzare CloudFormation per creare una funzione Lambda utilizzando un archivio di file .zip. Per creare una funzione Lambda da un file .zip, devi prima caricare il file su un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando AWS CLI, consulta [Move objects](#) nella Guida per l'utente. AWS CLI

Per i runtime di Python Node.js e Python, puoi anche fornire codice sorgente in linea nel tuo modello. CloudFormation quindi crea un file .zip contenente il codice durante la creazione della funzione.

Utilizzo di un file .zip esistente

Nel CloudFormation modello, la `AWS::Lambda::Function` risorsa specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`



- **Code**: inserisci il nome del bucket Amazon S3 e il nome del file .zip nei campi **S3Bucket** e **S3Key**
- **Runtime**: imposta il runtime prescelto

### Creazione di un file .zip da codice inline

Puoi dichiarare semplici funzioni scritte in Python o Node.js incorporate in un modello.

CloudFormation Poiché il codice è incorporato in YAML o JSON, non puoi aggiungere dipendenze esterne al tuo pacchetto di implementazione. Ciò significa che la tua funzione deve utilizzare la versione dell' AWS SDK inclusa nel runtime. I requisiti del modello, come la necessità di evitare determinati caratteri, rendono anche più difficile l'utilizzo delle funzionalità di controllo della sintassi e di completamento del codice dell'IDE. Ciò significa che il tuo modello potrebbe richiedere test aggiuntivi. A causa di queste limitazioni, la dichiarazione di funzioni in linea è più adatta per codice molto semplice che cambia raramente.

Per creare un file .zip dal codice in linea per Node.js e dai runtime di Python, imposta le seguenti proprietà nella risorsa del modello: `AWS::Lambda::Function`

- **PackageType**: imposta il valore su `Zip`
- **Code**: inserisci il codice della funzione nel campo `ZipFile`
- **Runtime**: imposta il runtime prescelto

Il file .zip generato non può superare i 4 MB CloudFormation . Per saperne di più sulla distribuzione delle funzioni utilizzando il file .zip in CloudFormation, vedi [AWS::Lambda: :Function](#) nella Guida per l'utente. CloudFormation

## Distribuisci funzioni Lambda per Python con immagini di container

Esistono tre modi per creare un'immagine di container per una funzione Lambda in Python:

- [Usando un AWS immagine base per Python](#)

[Le immagini di base AWS](#) sono precaricate con un runtime in linguaggio, un client di interfaccia di runtime per gestire l'interazione tra Lambda e il codice della funzione e un emulatore di interfaccia di runtime per i test locali.

- [Usando un AWS OS-only immagine di base](#)

[AWS OS-only le immagini di base](#) contengono una distribuzione Amazon Linux e l'emulatore dell'interfaccia [di runtime](#). Queste immagini vengono comunemente utilizzate per creare immagini contenitore per linguaggi compilati, come [Go](#) e [Rust](#), e per una lingua o una versione per la quale Lambda non fornisce un'immagine di base, come 19. Node.js Puoi anche utilizzare immagini di OS-only base per implementare un runtime [personalizzato](#). Per rendere l'immagine compatibile con Lambda, devi includere il [client di interfaccia di runtime per Python](#) nell'immagine.

- [Usare un'immagine non di AWS base](#)

È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere il [client di interfaccia di runtime per Python](#) nell'immagine.

### Tip

Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta [Utilizzo di compilazioni a più fasi](#) nella documentazione Docker. Per creare immagini di container efficienti, segui le [best practice per scrivere file Docker](#).

Questa pagina spiega come creare, testare e implementare le immagini di container per Lambda.

### Argomenti

- [AWS immagini di base per Python](#)
- [Usando un AWS immagine base per Python](#)
- [Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime](#)

## AWS immagini di base per Python

AWS fornisce le seguenti immagini di base per Python:

| Tag  | Runtime     | Sistema operativo | Dockerfile                                               | Raggiunta obsolescenza |
|------|-------------|-------------------|----------------------------------------------------------|------------------------|
| 3.15 | Python 3.15 | Amazon Linux 2023 | <a href="#">Dockerfile per Python 3.15 in poi GitHub</a> | Non programmato        |
| 3.14 | Python 3.14 | Amazon Linux 2023 | <a href="#">Dockerfile per Python 3.14 in poi GitHub</a> | 30 giugno 2029         |
| 3.13 | Python 3.13 | Amazon Linux 2023 | <a href="#">Dockerfile per Python 3.13 in poi GitHub</a> | 30 giugno 2029         |
| 3.12 | Python 3.12 | Amazon Linux 2023 | <a href="#">Dockerfile per Python 3.12 in poi GitHub</a> | 31 ottobre 2028        |
| 3.11 | Python 3.11 | Amazon Linux 2    | <a href="#">Dockerfile per Python 3.11 in poi GitHub</a> | 30 giugno 2027         |
| 3.10 | Python 3.10 | Amazon Linux 2    | <a href="#">Dockerfile per Python 3.10 in poi GitHub</a> | 31 ottobre 2026        |

Repository Amazon ECR: [gallery.ecr. aws/lambda/python](https://gallery.ecr.aws/aws/lambda/python)

Le immagini di base Python 3.12 e versioni successive si basano sull'[immagine di container minima di Amazon Linux 2023](#). Le immagini di base di Python 3.8-3.11 sono basate sull'immagine Amazon Linux 2. AL2023-based le immagini offrono diversi vantaggi rispetto ad Amazon Linux 2, tra cui un ingombro di distribuzione ridotto e versioni aggiornate di librerie come `glibc`.

AL2023-based le immagini utilizzano `microdnf` (con collegamenti simbolici `dnf`) come gestore di pacchetti anziché `yum`, che è il gestore di pacchetti predefinito in Amazon Linux 2. `microdnf` è un'implementazione autonoma di `dnf`. Per un elenco dei pacchetti inclusi nelle AL2023-based immagini, consulta le colonne Minimal Container in [Confronto dei pacchetti installati su Amazon Linux 2023 Container Images](#). Per ulteriori informazioni sulle differenze tra AL2023 e Amazon Linux 2, consulta [Introduzione del runtime di Amazon Linux 2023 per AWS Lambda](#) nel blog AWS Compute.

 Note

Per eseguire AL2023-based le immagini localmente, incluso con AWS Serverless Application Model (AWS SAM), devi utilizzare la versione Docker 20.10.10 o successiva.

## Percorso di ricerca delle dipendenze nelle immagini di base

Quando utilizzi un'istruzione `import` nel codice, il runtime Python cerca nelle directory del suo percorso di ricerca finché non trova il modulo o il pacchetto. Per impostazione predefinita, il runtime cerca prima nella directory `{LAMBDA_TASK_ROOT}`. Se includi una versione di una libreria inclusa nel runtime nella tua immagine, questa versione avrà la precedenza sulla versione inclusa nel runtime.

Gli altri passaggi del percorso di ricerca dipendono dalla versione dell'immagine base Lambda per il Python che stai utilizzando:

- Python 3.11 e versioni successive: le librerie e le Runtime-included librerie installate con pip sono installate nella directory. `/var/lang/lib/python3.11/site-packages` Questa directory ha la precedenza su `/var/runtime` nel percorso di ricerca. Puoi sovrascrivere l'SDK usando pip per installare una versione più recente. Puoi usare pip per verificare che l'SDK incluso nel runtime e le sue dipendenze siano compatibili con tutti i pacchetti che installi.
- Python 3.8-3.10: le librerie sono installate nella directory. Runtime-included `/var/runtime` Pip-installed le librerie sono installate nella directory. `/var/lang/lib/python3.x/site-packages` La directory `/var/runtime` ha la precedenza su `/var/lang/lib/python3.x/site-packages` nel percorso di ricerca.

Puoi visualizzare il percorso di ricerca completo per la tua funzione Lambda aggiungendo il seguente frammento di codice.

```
import sys

search_path = sys.path
print(search_path)
```

## Usando un AWS immagine base per Python

### Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [AWS CLI versione 2](#)
- [Docker](#) (versione minima 25.0.0)
- Il [plugin buildx](#) di Docker.
- Python

Creazione di un'immagine da un'immagine di base

Per creare un'immagine contenitore da un AWS immagine base per Python

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
cd example
```

2. Crea un nuovo file denominato `lambda_function.py`. A fini di test, puoi utilizzare il codice della funzione di esempio seguente o sostituirlo con il tuo codice personalizzato.

Example Funzione Python

```
import sys
def handler(event, context):
    return 'Hello from AWS Lambda using Python' + sys.version + '!'
```

3. Crea un nuovo file denominato `requirements.txt`. Se stai utilizzando il codice della funzione di esempio del passaggio precedente, puoi lasciare il file vuoto perché non ci sono dipendenze. Altrimenti, elenca ogni libreria richiesta. Ad esempio, ecco come dovrebbe apparire la tua versione di `requirements.txt` se la funzione utilizza AWS SDK per Python (Boto3):

Example requirements.txt

```
boto3
```

4. Crea un nuovo Dockerfile con la seguente configurazione:
  - Imposta la proprietà FROM sull'[URI dell'immagine di base](#).
  - Usa il comando COPY per copiare il codice della funzione e le dipendenze di runtime in una `{LAMBDA_TASK_ROOT}` variabile di [Lambda-defined ambiente](#).
  - Imposta l'argomento CMD specificando il gestore della funzione Lambda.

Nota che l'esempio Dockerfile non include un'[istruzione USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

### Example Dockerfile

```
FROM public.ecr.aws/lambda/python:3.12

# Copy requirements.txt
COPY requirements.txt ${LAMBDA_TASK_ROOT}

# Install the specified packages
RUN pip install -r requirements.txt

# Copy function code
COPY lambda_function.py ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler (could also be done as a parameter override outside
  of the Dockerfile)
CMD [ "lambda_function.handler" ]
```

5. Crea l'immagine Docker con il comando [docker build](#). L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il [tag](#) `test`. Per rendere l'immagine compatibile con Lambda, devi usare l'opzione `--provenance=false`.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test .
```

#### Note

Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Per creare una funzione Lambda utilizzando l'architettura del set di istruzioni ARM64, accertati di modificare il comando per utilizzare l'opzione `--platform linux/arm64`.

## (Facoltativo) Test dell'immagine in locale

1. Avvia l'immagine Docker con il comando `docker run`. In questo esempio, `docker-image` è il nome dell'immagine e `test` è il tag.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

### Note

Per creare l'immagine Docker per l'architettura del set di istruzioni ARM64, assicurati di utilizzare l'opzione `--platform linux/arm64` anziché `--platform linux/amd64`.

2. Da una nuova finestra di terminale, invia un evento all'endpoint locale.

### Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

### PowerShell

In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType "application/json"
```

3. Ottieni l'ID del container.

```
docker ps
```

4. Utilizza il comando [docker kill](#) per arrestare il container. In questo comando, sostituisci 3766c4ab331c con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

## Implementazione dell'immagine

### Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Per autenticare la CLI Docker nel registro Amazon ECR, esegui il comando [get-login-password](#).

- Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
- Sostituiscilo 111122223333 con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```



 Note

Il repository Amazon ECR deve essere nello stesso della funzione Regione AWS Lambda.

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando [docker tag](#) per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:
  - `docker-image:test` è il nome e [tag](#) dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
  - Sostituisci l'<ECRrepositoryUri> con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando [docker push](#) per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Crea un ruolo di esecuzione](#) per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puoi creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta [Autorizzazioni multiaccount Amazon ECR](#).

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nel repository Amazon ECR e quindi utilizzare il comando [update-function-code](#) per implementare l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per implementare la nuova immagine nella stessa funzione Lambda, devi utilizzare il comando [update-function-code](#) anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \  
  --function-name hello-world \  
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --publish
```

Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

Se si utilizza un'immagine di [OS-only base](#) o un'immagine di base alternativa, è necessario includere il client dell'interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime estende l'[API Runtime](#), che gestisce l'interazione tra Lambda e il codice della funzione.

Installa il [client di interfaccia di runtime per Python](#) utilizzando il gestore di pacchetti pip:

```
pip install awslambdarc
```

Puoi anche scaricare il client dell'interfaccia runtime [Python](#) da GitHub.

L'esempio seguente dimostra come creare un'immagine contenitore per Python utilizzando un'immagine non di base.AWS Il Dockerfile di esempio utilizza un'immagine di base Python ufficiale. Il Dockerfile include il client di interfaccia di runtime per Python.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [AWS CLI versione 2](#)

- [Docker](#) (versione minima 25.0.0)
- Il [plugin buildx](#) di Docker.
- Python

## Creazione di un'immagine da un'immagine di base alternativa

Per creare un'immagine contenitore da un file non-AWS immagine di base

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
cd example
```

2. Crea un nuovo file denominato `lambda_function.py`. A fini di test, puoi utilizzare il codice della funzione di esempio seguente o sostituirlo con il tuo codice personalizzato.

### Example Funzione Python

```
import sys
def handler(event, context):
    return 'Hello from AWS Lambda using Python' + sys.version + '!!'
```

3. Crea un nuovo file denominato `requirements.txt`. Se stai utilizzando il codice della funzione di esempio del passaggio precedente, puoi lasciare il file vuoto perché non ci sono dipendenze. Altrimenti, elenca ogni libreria richiesta. Ad esempio, ecco come dovrebbe apparire la tua versione di `requirements.txt` se la funzione utilizza AWS SDK per Python (Boto3):

### Example requirements.txt

```
boto3
```

4. Crea un nuovo Dockerfile. Il seguente Dockerfile utilizza un'immagine di base Python ufficiale anziché un'[immagine di base AWS](#). Il Dockerfile include il [client di interfaccia di runtime](#), che rende l'immagine compatibile con Lambda. Il seguente Dockerfile di esempio utilizza una [build multi-fase](#).
  - Imposta la proprietà FROM sull'immagine di base.
  - Imposta l'ENTRYPOINT sul modulo su cui desideri che il container Docker venga eseguito all'avvio. In questo caso, il modulo è il client di interfaccia di runtime.

- Imposta il CMD specificando il gestore della funzione Lambda.

Nota che l'esempio Dockerfile non include un'istruzione [USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente root quando non viene fornita alcuna istruzione USER.

### Example Dockerfile

```
# Define custom function directory
ARG FUNCTION_DIR="/function"

FROM python:3.12 AS build-image

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Copy function code
RUN mkdir -p ${FUNCTION_DIR}
COPY . ${FUNCTION_DIR}

# Install the function's dependencies
RUN pip install \
    --target ${FUNCTION_DIR} \
    awslambdarc

# Use a slim version of the base Python image to reduce the final image size
FROM python:3.12-slim

# Include global arg in this stage of the build
ARG FUNCTION_DIR
# Set working directory to function root directory
WORKDIR ${FUNCTION_DIR}

# Copy in the built dependencies
COPY --from=build-image ${FUNCTION_DIR} ${FUNCTION_DIR}

# Set runtime interface client as default command for the container runtime
ENTRYPOINT [ "/usr/local/bin/python", "-m", "awslambdarc" ]
# Pass the name of the function handler as an argument to the runtime
CMD [ "lambda_function.handler" ]
```

5. Crea l'immagine Docker con il comando [docker build](#). L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il [tag](#) `test`. Per rendere l'immagine compatibile con Lambda, devi usare l'opzione `--provenance=false`.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test .
```

#### Note

Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Per creare una funzione Lambda utilizzando l'architettura del set di istruzioni ARM64, accertati di modificare il comando per utilizzare l'opzione `--platform linux/arm64`.

### (Facoltativo) Test dell'immagine in locale

Usa il [simulatore dell'interfaccia di runtime](#) per testare localmente l'immagine. Puoi [creare l'emulatore nella tua immagine](#) o seguire la procedura riportata e installarlo sul tuo computer locale.

### Installazione ed esecuzione dell'emulatore di interfaccia di runtime sul computer locale

1. Dalla directory del progetto, esegui il comando seguente per scaricare l'emulatore di interfaccia di runtime (architettura x86-64) GitHub e installarlo sul tuo computer locale.

#### Linux/macOS

```
mkdir -p ~/.aws-lambda-rie && \
  curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-
runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
  chmod +x ~/.aws-lambda-rie/aws-lambda-rie
```

Per installare l'emulatore arm64, sostituite l'URL del GitHub repository nel comando precedente con il seguente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

## PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"
if (-not (Test-Path $dirPath)) {
    New-Item -Path $dirPath -ItemType Directory
}

$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/
releases/latest/download/aws-lambda-rie"
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Per installare l'emulatore arm64, sostituisci \$downloadLink con quanto segue:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Avvia l'immagine Docker con il comando `docker run`. Tenere presente quanto segue:

- `docker-image` è il nome dell'immagine e `test` è il tag.
- `/usr/local/bin/python -m awslambdarc lambda_function.handler` è l'ENTRYPOINT seguito dal CMD del Dockerfile.

## Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
9000:8080 \
    --entrypoint /aws-lambda/aws-lambda-rie \
    docker-image:test \
    /usr/local/bin/python -m awslambdarc lambda_function.handler
```

## PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
9000:8080 `
--entrypoint /aws-lambda/aws-lambda-rie `
docker-image:test `
    /usr/local/bin/python -m awslambdarc lambda_function.handler
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

#### Note

Per creare l'immagine Docker per l'architettura del set di istruzioni ARM64, assicurati di utilizzare l'opzione `--platform linux/arm64` anziché `--platform linux/amd64`.

### 3. Pubblica un evento nell'endpoint locale.

#### Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

#### PowerShell

In PowerShell, esegui il seguente comando: `Invoke-WebRequest`

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType  
"application/json"
```



#### 4. Ottieni l'ID del container.

```
docker ps
```

#### 5. Utilizza il comando [docker kill](#) per arrestare il container. In questo comando, sostituisci 3766c4ab331c con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

### Implementazione dell'immagine

#### Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

#### 1. Per autenticare la CLI Docker nel registro Amazon ECR, esegui il comando [get-login-password](#).

- Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
- Sostituiscilo 111122223333 con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

#### 2. Crea un repository in Amazon ECR utilizzando il comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

#### Note

Il repository Amazon ECR deve essere nello stesso della funzione Regione AWS Lambda.

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{  
  "repository": {
```

```

    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}

```

3. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando [docker tag](#) per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:
  - `docker-image:test` è il nome e [tag](#) dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
  - Sostituisci l'<ECRrepositoryUri> con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando [docker push](#) per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Crea un ruolo di esecuzione](#) per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.

7. Creazione della funzione Lambda Per ImageUri, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puoi creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta [Autorizzazioni multiaccount Amazon ECR](#).

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nel repository Amazon ECR e quindi utilizzare il comando [update-function-code](#) per implementare l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per implementare la nuova immagine nella stessa funzione Lambda, devi utilizzare il comando [update-function-code](#) anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio

seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \  
  --function-name hello-world \  
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --publish
```

Per un esempio di come creare un'immagine Python da un'immagine di base Alpine, consulta [Supporto delle immagini di container per Lambda](#) sul Blog AWS .

## Utilizzo dei livelli per le funzioni Lambda in Python

Usa i livelli [Lambda](#) per impacchettare codice e dipendenze che desideri riutilizzare in più funzioni. I livelli di solito contengono dipendenze dalla libreria, un [runtime personalizzato](#) o file di configurazione. La creazione di un livello prevede tre passaggi generali:

1. Crea un pacchetto per il contenuto del livello. Ciò significa creare un archivio di file con estensione .zip che contiene le dipendenze che desideri utilizzare nelle funzioni.
2. Crea il livello in Lambda.
3. Aggiungi il livello alle tue funzioni.

### Argomenti

- [Crea un pacchetto per il contenuto del livello](#)
- [Crea il livello in Lambda](#)
- [Aggiungi il livello alla tua funzione](#)
- [App di esempio](#)

### Crea un pacchetto per il contenuto del livello

Per creare un layer, raggruppa i pacchetti in un archivio di file .zip che soddisfi i seguenti requisiti:

- Crea il livello utilizzando la stessa versione di Python che intendi utilizzare per la funzione Lambda. Ad esempio, se crei il tuo layer usando Python 3.14, usa il runtime Python 3.14 per la tua funzione.
- Il file .zip deve includere una python directory al livello principale.
- I pacchetti del livello devono essere compatibili con Linux. Le funzioni Lambda vengono eseguite su Amazon Linux.

Puoi creare livelli che contengono librerie Python di terze parti installate con pip (come `requests` o `orandas`) o moduli e pacchetti Python personalizzati.

### Third-party dipendenze

Per creare un livello usando i pacchetti pip

1. Scegliete uno dei seguenti metodi per installare pip i pacchetti nella directory di primo livello richiesta `()python/`:

## pip install

Per pacchetti Python puri (come requests o boto3):

```
pip install requests -t python/
```

Alcuni pacchetti Python, come NumPy e Pandas, includono componenti C compilati. Se stai creando un layer con questi pacchetti su macOS o Windows, potresti dover usare questo comando per installare una ruota: Linux-compatible

```
pip install numpy --platform manylinux2014_x86_64 --only-binary=:all: -t python/
```

Per ulteriori informazioni su come lavorare con i pacchetti Python che contengono componenti compilati, consulta. [Creazione di un pacchetto di implementazione .zip con librerie native](#)

## requirements.txt

L'uso di un `requirements.txt` file consente di gestire le versioni dei pacchetti e garantire installazioni coerenti.

### Example requirements.txt

```
requests==2.31.0  
boto3==1.37.34  
numpy==1.26.4
```

Se il `requirements.txt` file include solo pacchetti Python puri (come requests o boto3):

```
pip install -r requirements.txt -t python/
```

Alcuni pacchetti Python, come NumPy e Pandas, includono componenti C compilati. Se stai creando un layer con questi pacchetti su macOS o Windows, potresti dover usare questo comando per installare una ruota: Linux-compatible

```
pip install -r requirements.txt --platform manylinux2014_x86_64 --only-binary=:all: -t python/
```

Per ulteriori informazioni su come lavorare con i pacchetti Python che contengono componenti compilati, consulta. [Creazione di un pacchetto di implementazione .zip con librerie native](#)

## 2. Comprimi il contenuto della directory. python

Linux/macOS

```
zip -r layer.zip python/
```

PowerShell

```
Compress-Archive -Path .\python -DestinationPath .\layer.zip
```

La struttura delle cartelle del tuo file .zip dovrebbe essere la seguente:

```
python/           # Required top-level directory
### requests/
### boto3/
### numpy/
### (dependencies of the other packages)
```

### Note

Se si utilizza un ambiente virtuale Python (venv) per installare i pacchetti, la struttura delle cartelle è diversa (ad esempio,). python/lib/python3.x/site-packages  
Se il file .zip include la python directory a livello principale, Lambda può localizzare e importare i pacchetti.

## Moduli Python personalizzati

Per creare un livello usando il tuo codice

### 1. Crea la directory di primo livello richiesta per il tuo layer:

```
mkdir python
```

2. Crea i tuoi moduli Python nella `python` directory. Il seguente modulo di esempio convalida gli ordini confermando che contengano le informazioni richieste.

Example modulo personalizzato: `validator.py`

```
import json

def validate_order(order_data):
    """Validates an order and returns formatted data."""
    required_fields = ['product_id', 'quantity']

    # Check required fields
    missing_fields = [field for field in required_fields if field not in
order_data]
    if missing_fields:
        raise ValueError(f"Missing required fields: {' '.join(missing_fields)}")

    # Validate quantity
    quantity = order_data['quantity']
    if not isinstance(quantity, int) or quantity < 1:
        raise ValueError("Quantity must be a positive integer")

    # Format and return the validated data
    return {
        'product_id': str(order_data['product_id']),
        'quantity': quantity,
        'shipping_priority': order_data.get('priority', 'standard')
    }

def format_response(status_code, body):
    """Formats the API response."""
    return {
        'statusCode': status_code,
        'body': json.dumps(body)
    }
```

3. Comprimi il contenuto della `python` cartella.

Linux/macOS

```
zip -r layer.zip python/
```



## PowerShell

```
Compress-Archive -Path .\python -DestinationPath .\layer.zip
```

La struttura delle cartelle del tuo file .zip dovrebbe essere la seguente:

```
python/      # Required top-level directory
### validator.py
```

4. Nella tua funzione, importa e usa i moduli come faresti con qualsiasi pacchetto Python. Esempio:

```
from validator import validate_order, format_response
import json

def lambda_handler(event, context):
    try:
        # Parse the order data from the event body
        order_data = json.loads(event.get('body', '{}'))

        # Validate and format the order
        validated_order = validate_order(order_data)

        return format_response(200, {
            'message': 'Order validated successfully',
            'order': validated_order
        })
    except ValueError as e:
        return format_response(400, {
            'error': str(e)
        })
    except Exception as e:
        return format_response(500, {
            'error': 'Internal server error'
        })
```

È possibile utilizzare il seguente evento [di test](#) per richiamare la funzione:

```
{
  "body": "{\"product_id\": \"ABC123\", \"quantity\": 2, \"priority\": \"express\"}"
}
```

```
}
```

Risposta prevista:

```
{
  "statusCode": 200,
  "body": "{\"message\": \"Order validated successfully\", \"order\": {\"product_id\": \"ABC123\", \"quantity\": 2, \"shipping_priority\": \"express\"}}"
```

Crea il livello in Lambda

Puoi pubblicare il tuo layer utilizzando la console AWS CLI o Lambda.

AWS CLI

Esegui il [AWS CLI comando](#) `publish-layer-version` per creare il layer Lambda:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip
--compatible-runtimes python3.14
```

Il parametro runtime [compatibile è facoltativo](#). Quando specificato, Lambda utilizza questo parametro per filtrare i livelli nella console Lambda.

Console

Creazione di un livello (console)

1. Apri la [pagina Layers](#) (Livelli) nella console Lambda.
2. Scegli Create layer (Crea livello).
3. Scegli Carica un file .zip, quindi carica l'archivio .zip che hai creato in precedenza.
4. (Facoltativo) Per i runtime compatibili, scegli il runtime Python che corrisponde alla versione Python utilizzata per creare il layer.
5. Scegli Create (Crea).

## Aggiungi il livello alla tua funzione

### AWS CLI

Per collegare il livello alla tua funzione, esegui il comando <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/update-function-configuration.html> AWS CLI `update-function-configuration`. Per il `--layers` parametro, usate il livello ARN. L'ARN deve specificare la versione (ad esempio, `arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`). Per ulteriori informazioni, consulta [Livelli e versioni di livelli](#).

```
aws lambda update-function-configuration --function-name my-function --cli-binary-format raw-in-base64-out --layers "arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1"
```

L'`cli-binary-format` opzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

### Console

#### Aggiunta di un livello a una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione.
3. Scorri verso il basso fino alla sezione Livelli, quindi scegli Aggiungi un livello.
4. In Scegli un livello, seleziona Livelli personalizzati, quindi scegli il tuo livello.

#### Note

Se non hai aggiunto un runtime [compatibile](#) durante la creazione del layer, il layer non verrà elencato qui. Puoi invece specificare l'ARN del livello.

5. Scegliere Aggiungi.

### App di esempio

Per altri esempi su come utilizzare i livelli Lambda, consulta l'applicazione di [esempio](#) `layer-python` nel repository Developer Guide. AWS Lambda GitHub Questa applicazione include due livelli che

contengono librerie Python. Dopo aver creato i livelli, è possibile distribuire e richiamare le funzioni corrispondenti per confermare che i livelli funzionino come previsto.

## Utilizzo dell'oggetto del contesto Lambda per recuperare le informazioni sulla funzione Python

Quando Lambda esegue la funzione, passa un oggetto Context al [gestore](#). Questo oggetto fornisce i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. Per ulteriori informazioni su come l'oggetto contesto viene passato al gestore di funzioni, consulta [Definire l'handler della funzione Lambda in Python](#).

### Metodi del contesto

- `get_remaining_time_in_millis`: restituisce il numero di millisecondi rimasti prima del timeout dell'esecuzione.

### Proprietà del contesto

- `function_name`: il nome della funzione Lambda.
- `function_version`: la [versione](#) della funzione.
- `invoked_function_arn`: l'Amazon Resource Name (ARN) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `memory_limit_in_mb`: la quantità di memoria allocata per la funzione.
- `aws_request_id`: l'identificatore della richiesta di invocazione.
- `log_group_name`: il gruppo di log per la funzione.
- `log_stream_name`: il flusso di log per l'istanza della funzione.
- `identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
  - `cognito_identity_id`: l'identità autenticata di Amazon Cognito.
  - `cognito_identity_pool_id`: il pool di identità Amazon Cognito che ha autorizzato l'invocazione.
- `client_context`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`

- `client.app_package_name`
- `custom`: un dict di valori personalizzati impostati dall'applicazione client per dispositivi mobili.
- `env`: un dict di informazioni di ambiente fornite dall'SDK AWS.

Powertools for Lambda (Python) fornisce una definizione di interfaccia per l'oggetto del contesto Lambda. È possibile utilizzare la definizione dell'interfaccia per i suggerimenti sul tipo o per esaminare ulteriormente la struttura dell'oggetto del contesto Lambda. Per la definizione dell'interfaccia, consulta [lambda\\_context.py](#) nel repository `powertools-lambda-python` su GitHub.

L'esempio seguente mostra una funzione di gestione che registra le informazioni di contesto.

Example handler.py

```
import time

def lambda_handler(event, context):
    print("Lambda function ARN:", context.invoked_function_arn)
    print("CloudWatch log stream name:", context.log_stream_name)
    print("CloudWatch log group name:", context.log_group_name)
    print("Lambda Request ID:", context.aws_request_id)
    print("Lambda function memory limits in MB:", context.memory_limit_in_mb)
    # We have added a 1 second delay so you can see the time remaining in
    get_remaining_time_in_millis.
    time.sleep(1)
    print("Lambda time remaining in MS:", context.get_remaining_time_in_millis())
```

Oltre alle opzioni elencate in precedenza, è possibile utilizzare l'SDK AWS X-Ray per [Strumentazione del codice Python in AWS Lambda](#) per identificare i percorsi del codice critici, tenere traccia delle relative prestazioni e acquisire i dati per l'analisi.

## Registrare e monitorare le funzioni Lambda con Python

AWS Lambda monitora automaticamente le funzioni Lambda e invia le voci di registro ad Amazon. CloudWatch La tua funzione Lambda include un gruppo di CloudWatch log Logs e un flusso di log per ogni istanza della tua funzione. L'ambiente di runtime di Lambda invia al flusso di log i dettagli su ogni invocazione e altri output dal codice della funzione. Per ulteriori informazioni sui CloudWatch log, consulta [Invio dei log delle funzioni Lambda ai CloudWatch log](#).

Per l'output dei log dal codice della funzione, puoi utilizzare il modulo di [logging](#) integrato. Per ottenere voci più dettagliate, puoi utilizzare qualunque libreria di log che scrive in `stdout` o `stderr`.

### Stampa nel log

Per inviare l'output base ai log, puoi utilizzare un metodo `print` nella funzione. L'esempio seguente registra i valori del gruppo e dello stream di CloudWatch log Logs e l'oggetto evento.

Nota che se la tua funzione genera log utilizzando `print` istruzioni Python, Lambda può inviare output di CloudWatch log a Logs solo in formato di testo semplice. Per acquisire i log in formato JSON strutturato, è necessario utilizzare una libreria di registrazione supportata. Per ulteriori informazioni, consulta [the section called "Utilizzo dei controlli di registrazione avanzati di Lambda con Python"](#).

### Example lambda\_function.py

```
import os
def lambda_handler(event, context):
    print('## ENVIRONMENT VARIABLES')
    print(os.environ['AWS_LAMBDA_LOG_GROUP_NAME'])
    print(os.environ['AWS_LAMBDA_LOG_STREAM_NAME'])
    print('## EVENT')
    print(event)
```

### Example Output log

```
START RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95 Version: $LATEST
## ENVIRONMENT VARIABLES
/aws/lambda/my-function
2025/08/31/[$LATEST]3893xmpl7fac4485b47bb75b671a283c
## EVENT
{'key': 'value'}
END RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95
```

```
REPORT RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95 Duration: 15.74 ms Billed
Duration: 147 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 130.49 ms
XRAY TraceId: 1-5e34a614-10bdxmpl1f1fb44f07bc535a1 SegmentId: 07f5xmpl2d1f6f85
Sampled: true
```

Il runtime di Python registra START, END e REPORT per ogni chiamata. La riga REPORT include i seguenti dati:

### Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per l'invocazione.
- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.
- Quantità max utilizzata – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- Durata Init – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- XRAY TraceId: per le richieste tracciate, l'ID di traccia. [AWS X-Ray](#)
- SegmentId— Per le richieste tracciate, l'ID del X-Ray segmento.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

### Utilizzo di una libreria di log

Per registri più dettagliati, utilizza il modulo di [log](#) nella libreria standard o qualunque libreria di log di terzi che scrive in stdout o stderr.

Per i runtime Python supportati, puoi scegliere se i log creati utilizzando il modulo logging standard vengono acquisiti in testo normale o JSON. Per ulteriori informazioni, consulta [the section called "Utilizzo dei controlli di registrazione avanzati di Lambda con Python"](#).

Attualmente, il formato di log predefinito per tutti i runtime di Python è il testo normale. L'esempio seguente mostra come gli output dei log creati utilizzando il logging modulo standard vengono acquisiti in testo semplice in Logs. CloudWatch

```
import os
```



```
import logging
logger = logging.getLogger()
logger.setLevel("INFO")

def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES')
    logger.info(os.environ['AWS_LAMBDA_LOG_GROUP_NAME'])
    logger.info(os.environ['AWS_LAMBDA_LOG_STREAM_NAME'])
    logger.info('## EVENT')
    logger.info(event)
```

L'output di logger include il livello del log, il timestamp e l'ID della richiesta.

```
START RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Version: $LATEST
[INFO] 2025-08-31T22:12:58.534Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 ##
ENVIRONMENT VARIABLES
[INFO] 2025-08-31T22:12:58.534Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 /aws/
lambda/my-function
[INFO] 2025-08-31T22:12:58.534Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 2025/01/31/
[$LATEST]1bbe51xmplb34a2788dbaa7433b0aa4d
[INFO] 2025-08-31T22:12:58.535Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 ## EVENT
[INFO] 2025-08-31T22:12:58.535Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 {'key':
'value'}
END RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125
REPORT RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Duration: 2.75 ms Billed
Duration: 117 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 113.51 ms
XRAY TraceId: 1-5e34a66a-474xmpl7c2534a87870b4370 SegmentId: 073cxmpl3e442861
Sampled: true
```

### Note

Quando il formato di log della funzione è impostato su testo semplice, l'impostazione predefinita a livello di log per i runtime Python è WARN. Ciò significa che Lambda invia solo output di log di livello WARN e inferiore a Logs. CloudWatch Per modificare il livello di log predefinito, utilizza il metodo `logging.setLevel()` di Python come mostrato in questo codice di esempio. Se imposti il formato di log della funzione su JSON, consigliamo di configurare il livello di log della funzione utilizzando i controlli di registrazione avanzati di Lambda e non impostando il livello di log nel codice. Per ulteriori informazioni, consulta [the section called “Utilizzo del filtraggio a livello di log con Python”](#)

## Utilizzo dei controlli di registrazione avanzati di Lambda con Python

Per darti un maggiore controllo su come i log delle tue funzioni vengono acquisiti, elaborati e utilizzati, puoi configurare le seguenti opzioni di registrazione per i runtime Lambda Python supportati:

- **Formato di registro:** seleziona tra testo normale e formato JSON strutturato per i log della tua funzione
- **Livello di registro:** per i log in formato JSON, scegli il livello di dettaglio dei log che Lambda invia ad Amazon CloudWatch, ad esempio ERROR, DEBUG o INFO
- **Gruppo di log:** scegli il gruppo di CloudWatch log a cui la tua funzione invia i log

Per ulteriori informazioni su queste opzioni di registrazione e istruzioni su come configurare la funzione per utilizzarle, consulta la pagina [the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”](#).

Per ulteriori informazioni sull'utilizzo delle opzioni di formato e livello di log con le funzioni Lambda in Python, consulta le istruzioni nelle sezioni seguenti.

### Utilizzo di log JSON strutturati con Python

Se selezioni JSON come formato di registro della tua funzione, Lambda invia l'output dei log dalla libreria di registrazione standard Python come JSON strutturato. CloudWatch Ogni oggetto di log JSON contiene almeno quattro coppie chiave-valore con le seguenti chiavi:

- `"timestamp"`: l'ora in cui è stato generato il messaggio di log
- `"level"`: il livello di log assegnato al messaggio
- `"message"`: il contenuto del messaggio di log
- `"requestId"`: l'ID di richiesta univoco dell'invocazione alla funzione

La libreria `logging` di Python può anche aggiungere a tale oggetto JSON ulteriori coppie chiave-valore, come `"logger"`.

Gli esempi nelle sezioni seguenti mostrano come gli output dei log generati utilizzando la `logging` libreria Python vengono acquisiti in CloudWatch Logs quando configuri il formato di log della funzione come JSON.

Nota che se usi il `print` metodo per produrre output di log di base come descritto in [the section called “Stampa nel log”](#), Lambda acquisisce questi output come testo semplice, anche se configuri il formato di registrazione della tua funzione come JSON.

Output log JSON standard che utilizzano la libreria di registrazione di Python

Il frammento di codice di esempio e l'output del log seguenti mostrano come gli output di log standard generati utilizzando la `logging` libreria Python vengono acquisiti in CloudWatch Logs quando il formato di log della funzione è impostato su JSON.

Example Codice di registrazione di Python

```
import logging
logger = logging.getLogger()

def lambda_handler(event, context):
    logger.info("Inside the handler function")
```

Example Record di log JSON

```
{
  "timestamp": "2025-10-27T19:17:45.586Z",
  "level": "INFO",
  "message": "Inside the handler function",
  "logger": "root",
  "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Registrazione di parametri aggiuntivi in JSON

Quando il formato dei log della funzione è impostato su JSON, è possibile registrare ulteriori parametri con la libreria standard `logging` di Python utilizzando la parola chiave `extra` per passare un dizionario Python all'output dei log.

Example Codice di registrazione di Python

```
import logging

def lambda_handler(event, context):
    logging.info(
        "extra parameters example",
```

```
        extra={"a":"b", "b": [3]},  
    )
```

### Example Record di log JSON

```
{  
  "timestamp": "2025-11-02T15:26:28Z",  
  "level": "INFO",  
  "message": "extra parameters example",  
  "logger": "root",  
  "requestId": "3dbd5759-65f6-45f8-8d7d-5bdc79a3bd01",  
  "a": "b",  
  "b": [  
    3  
  ]  
}
```

### Registrazione delle eccezioni in JSON

Il seguente frammento di codice mostra come le eccezioni Python vengono acquisite nell'output log della funzione quando si configura JSON come formato di log. Nota che agli output log generati utilizzando `logging.exception` viene assegnato il livello di log `ERROR`.

### Example Codice di registrazione di Python

```
import logging  
  
def lambda_handler(event, context):  
    try:  
        raise Exception("exception")  
    except:  
        logging.exception("msg")
```

### Example Record di log JSON

```
{  
  "timestamp": "2025-11-02T16:18:57Z",  
  "level": "ERROR",  
  "message": "msg",  
  "logger": "root",  
  "stackTrace": [  
    "
```

```

    " File \"/var/task/lambda_function.py\", line 15, in lambda_handler\n    raise
Exception(\"exception\")\n\"
  ],
  \"errorType\": \"Exception\",
  \"errorMessage\": \"exception\",
  \"requestId\": \"3f9d155c-0f09-46b7-bdf1-e91dab220855\",
  \"location\": \"/var/task/lambda_function.py:lambda_handler:17\"
}

```

## Log JSON strutturati con altri strumenti di registrazione

Se il codice utilizza già un'altra libreria di registrazione, come Powertools for, per produrre log strutturati JSON AWS Lambda, non è necessario apportare alcuna modifica. AWS Lambda non codifica due volte i log già codificati in JSON. Anche se configuri la tua funzione per utilizzare il formato di log JSON, i tuoi output di registrazione vengono visualizzati nella CloudWatch struttura JSON definita.

L'esempio seguente mostra come gli output dei log generati utilizzando il pacchetto Powertools for AWS Lambda vengono acquisiti in Logs. CloudWatch Il formato di questo output di log è lo stesso indipendentemente dal fatto che la configurazione di registrazione della funzione sia impostata su JSON o TEXT. Per ulteriori informazioni sull'utilizzo di Powertools per AWS Lambda, vedere e [the section called “Utilizzo di Powertools per AWS Lambda \(Python\) e AWS SAM per la registrazione strutturata”](#) [the section called “Utilizzo di Powertools per AWS Lambda \(Python\) e AWS CDK per la registrazione strutturata”](#)

Example frammento di codice di registrazione di Python (usando Powertools per AWS Lambda)

```

from aws_lambda_powertools import Logger

logger = Logger()

def lambda_handler(event, context):
    logger.info("Inside the handler function")

```

Example Record di registro JSON (utilizzando Powertools per AWS Lambda)

```

{
  "level": "INFO",
  "location": "lambda_handler:7",
  "message": "Inside the handler function",
  "timestamp": "2025-10-31 22:38:21,010+0000",

```

```
"service": "service_undefined",  
"xray_trace_id": "1-654181dc-65c15d6b0fecbdd1531ecb30"  
}
```

## Utilizzo del filtraggio a livello di log con Python

Configurando il filtro a livello di registro, puoi scegliere di inviare solo i log con un determinato livello di registrazione o inferiore a Logs. CloudWatch Per informazioni su come configurare il filtraggio a livello di log della funzione, consulta la pagina [the section called “Log-level filtraggio”](#).

Per AWS Lambda filtrare i log delle applicazioni in base al relativo livello di registro, la funzione deve utilizzare i log in formato JSON. Puoi farlo in due modi:

- Crea output di log utilizzando la `logging` libreria Python standard e configura la tua funzione per utilizzare la formattazione dei log JSON. AWS Lambda quindi filtra gli output del log utilizzando la coppia chiave-valore «level» nell'oggetto JSON descritto in [the section called “Utilizzo di log JSON strutturati con Python”](#) Per informazioni su come configurare il formato di registro della funzione, consulta [the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”](#)
- Utilizzate un'altra libreria o metodo di registrazione per creare log strutturati JSON nel codice che includono una coppia chiave-valore «level» che definisce il livello dell'output del log. Ad esempio, puoi utilizzare Powertools per generare output AWS Lambda di log strutturati JSON dal tuo codice.

Inoltre, è possibile utilizzare un'istruzione di stampa per generare un oggetto JSON contenente un identificatore a livello di log. La seguente istruzione `print` produce un output in formato JSON in cui il livello di registro è impostato su INFO. AWS Lambda invia l'oggetto JSON a CloudWatch Logs se il livello di registrazione della funzione è impostato su INFO, DEBUG o TRACE.

```
print({'msg':"My log message", "level":"info"})
```

Per consentire a Lambda di filtrare i log della funzione, è necessario includere anche una coppia chiave-valore `"timestamp"` nell'output log JSON. L'ora deve essere specificata in un formato di timestamp [RFC 3339](#) valido. Se non fornisci un timestamp valido, Lambda assegna al log il livello INFO e aggiunge un timestamp per te.

## Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor di codice incorporato, i log sono contenuti nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, trovi l'output del registro nella sezione Dettagli.

### Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Aprire la pagina dei gruppi di [log](#) sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (/aws/lambda/***your-function-name***).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'[istanza della funzione](#). Un flusso di log viene visualizzato quando si aggiorna la funzione Lambda e quando vengono create istanze aggiuntive per gestire chiamate simultanee. Per trovare i log per una chiamata specifica, ti consigliamo di strumentare la tua funzione con AWS X-Ray. X-Ray registra i dettagli sulla richiesta e sul flusso di log nella traccia.

### Visualizzazione dei log con AWS CLI

AWS CLI È uno strumento open source che puoi utilizzare per interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della [AWS CLI versione 2](#).

È possibile utilizzare [AWS CLI](#) per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

### Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
```

```

    "StatusCode": 200,
    "LogResult":
      "U1RBULQgUmVxdWVzdElkOiA4N2QwNDRIOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2l1vb...",
    "ExecutedVersion": "$LATEST"
  }

```

## Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'base64 utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per my-function.

```

aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode

```

L'cli-binary-format opzione è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```

START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
Duration: 80 ms      Memory Size: 128 MB      Max Memory Used: 73 MB

```

L'utilità base64 è disponibile su Linux, macOS e [Ubuntu su Windows](#). Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

## Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.



L'cli-binary-formatopzione è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example(solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il comando seguente per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
```

```

    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\${LATEST}\",
\r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
MB\t\n",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
  "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}

```

## Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o [configurare un periodo di conservazione](#) trascorso il quale i log vengono eliminati automaticamente.

## Utilizzo di altri strumenti di registrazione e librerie

[Powertools for AWS Lambda \(Python\)](#) è un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori. L'[utilità di registrazione](#) fornisce un logger ottimizzato per Lambda che include informazioni aggiuntive sul contesto delle funzioni in tutte le funzioni con output strutturato come JSON. Utilizza l'utility per eseguire le seguenti operazioni:

- Acquisizione di campi essenziali dal contesto Lambda, avvio a freddo e output di registrazione della struttura come JSON
- Registrazione degli eventi di chiamata Lambda quando richiesto (disabilitata per impostazione predefinita)
- Stampa di tutti i log solo per una percentuale di chiamate tramite campionamento dei log (disabilitata per impostazione predefinita)
- Aggiunta di chiavi supplementari al log strutturato in qualsiasi momento
- Utilizzo di un formattatore di log personalizzato (Bring Your Own Formatter) per generare i log in una struttura compatibile con Logging RFC dell'organizzazione

Utilizzo di Powertools per AWS Lambda (Python) e AWS SAM per la registrazione strutturata

Segui i passaggi riportati sotto per scaricare, creare e implementare un'applicazione Python Hello World di esempio con moduli [Powertools per Python](#) integrati utilizzando AWS SAM. Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a CloudWatch. AWS X-Ray La funzione restituisce un messaggio `hello world`.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Python 3.9
- [AWS CLI versione 2](#)
- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI. AWS SAM](#)

Distribuisci un esempio AWS SAM applicazione

1. Inizializza l'applicazione utilizzando il modello Python Hello World.

```
sam init --app-template hello-world-powertools-python --name sam-app --package-type Zip --runtime python3.9 --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

### 3. Distribuire l'app.

```
sam deploy --guided
```

### 4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi Enter.

#### Note

Perché HelloWorldFunction potrebbe non avere un'autorizzazione definita, va bene? , assicurati di entrare.

### 5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query  
'Stacks[0].Outputs[?OutputKey=='HelloWorldApi'].OutputValue' --output text
```

### 6. Richiama l'endpoint dell'API:

```
curl GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

### 7. Per ottenere i log per la funzione, esegui [sam logs](#). Per ulteriori informazioni, consulta l'argomento relativo all'[utilizzo dei log](#) nella Guida per sviluppatori AWS Serverless Application Model .

```
sam logs --stack-name sam-app
```

L'output del log ha la struttura seguente:

```
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04  
2025-02-03T14:59:50.371000 INIT_START Runtime Version:  
python:3.9.v16 Runtime Version ARN: arn:aws:lambda:us-  
east-1::runtime:07a48df201798d627f2b950f03bb227aab4a655a1d019c3296406f95937e2525
```

```

2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.112000
START RequestId: d455cfc4-7704-46df-901b-2a5cce9405be Version: $LATEST
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.114000 {
  "level": "INFO",
  "location": "hello:23",
  "message": "Hello world API - HTTP 200",
  "timestamp": "2025-02-03 14:59:51,113+0000",
  "service": "PowertoolsHelloWorld",
  "cold_start": true,
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYt0c9j",
  "function_memory_size": "128",
  "function_arn": "arn:aws:lambda:us-east-1:111122223333:function:sam-app-
HelloWorldFunction-YBg8yfYt0c9j",
  "function_request_id": "d455cfc4-7704-46df-901b-2a5cce9405be",
  "correlation_id": "e73f8aef-5e07-436e-a30b-63e4b23f0047",
  "xray_trace_id": "1-63dd2166-434a12c22e1307ff2114f299"
}
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "function_name",
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYt0c9j",
  "service": "PowertoolsHelloWorld",
  "ColdStart": [
    1.0
  ]
}

```

```

2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "HelloWorldInvocations",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "service": "PowertoolsHelloWorld",
  "HelloWorldInvocations": [
    1.0
  ]
}
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.128000 END
RequestId: d455cfc4-7704-46df-901b-2a5cce9405be
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.128000
REPORT RequestId: d455cfc4-7704-46df-901b-2a5cce9405be    Duration: 16.33 ms
Billed Duration: 756 ms    Memory Size: 128 MB    Max Memory Used: 64 MB    Init
Duration: 739.46 ms
XRAY TraceId: 1-63dd2166-434a12c22e1307ff2114f299    SegmentId: 3c5d18d735a1ced0
Sampled: true

```

8. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

## Gestione della conservazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, elimina il gruppo di log o configura un periodo di conservazione dopo il quale i log CloudWatch vengono eliminati automaticamente. Per impostare la conservazione dei log, aggiungi quanto segue al tuo AWS SAM modello:

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

## Utilizzo di Powertools per AWS Lambda (Python) e AWS CDK per la registrazione strutturata

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Hello World Python di esempio con moduli [Powertools for AWS Lambda \(Python\) integrati utilizzando](#). AWS CDK Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format a e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio hello world.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Python 3.9
- [AWS CLI versione 2](#)
- [AWS CDK versione 2](#)
- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI. AWS SAM](#)

## Distribuisci un esempio AWS CDK applicazione

1. Crea una directory di progetto per la nuova applicazione.

```
mkdir hello-world  
cd hello-world
```

2. Inizializza l'app.

```
cdk init app --language python
```

3. Installa le dipendenze di Python.

```
pip install -r requirements.txt
```

4. Crea una directory `lambda_function` nella cartella root.

```
mkdir lambda_function  
cd lambda_function
```

5. Crea un file `app.py` e aggiungi il seguente codice al file. Questo è il codice per la funzione Lambda.

```
from aws_lambda_powertools.event_handler import APIGatewayRestResolver  
from aws_lambda_powertools.utilities.typing import LambdaContext  
from aws_lambda_powertools.logging import correlation_paths  
from aws_lambda_powertools import Logger  
from aws_lambda_powertools import Tracer  
from aws_lambda_powertools import Metrics  
from aws_lambda_powertools.metrics import MetricUnit  
  
app = APIGatewayRestResolver()  
tracer = Tracer()  
logger = Logger()  
metrics = Metrics(namespace="PowertoolsSample")  
  
@app.get("/hello")  
@tracer.capture_method  
def hello():  
    # adding custom metrics  
    # See: https://docs.aws.amazon.com/powertools/python/latest/latest/core/metrics/
```



```

    metrics.add_metric(name="HelloWorldInvocations", unit=MetricUnit.Count,
value=1)

    # structured log
    # See: https://docs.aws.amazon.com/powertools/python/latest/latest/core/
logger/
    logger.info("Hello world API - HTTP 200")
    return {"message": "hello world"}

# Enrich logging with contextual information from Lambda
@logger.inject_lambda_context(correlation_id_path=correlation_paths.API_GATEWAY_REST)
# Adding tracer
# See: https://docs.aws.amazon.com/powertools/python/latest/latest/core/tracer/
@tracer.capture_lambda_handler
# ensures metrics are flushed upon request completion/failure and capturing
ColdStart metric
@metrics.log_metrics(capture_cold_start_metric=True)
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    return app.resolve(event, context)

```

6. Apri la directory `hello_world`. Dovrebbe essere visualizzato un file denominato `hello_world_stack.py`.

```

cd ..
cd hello_world

```

7. Apri `hello_world_stack.py` e aggiungi il seguente codice al file. Contiene il [Lambda Constructor](#), che crea la funzione Lambda, configura le variabili di ambiente per Powertools e imposta la conservazione dei log su una settimana, e il Constructor [ApiGatewayv 1, che crea l'API REST](#).

```

from aws_cdk import (
    Stack,
    aws_apigateway as apigwv1,
    aws_lambda as lambda_,
    CfnOutput,
    Duration
)
from constructs import Construct

class HelloWorldStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

```

```

# Powertools Lambda Layer
powertools_layer = lambda_.LayerVersion.from_layer_version_arn(
    self,
    id="lambda-powertools",
    # Using AWS Lambda Powertools with Lambda Layer
    # This imports the Powertools layer which provides observability
    features for Lambda functions
    # For available versions, see: https://docs.aws.amazon.com/powertools/
    python/latest/#lambda-layer
)

function = lambda_.Function(self,
    'sample-app-lambda',
    runtime=lambda_.Runtime.PYTHON_3_9,
    layers=[powertools_layer],
    code = lambda_.Code.from_asset("./lambda_function/"),
    handler="app.lambda_handler",
    memory_size=128,
    timeout=Duration.seconds(3),
    architecture=lambda_.Architecture.X86_64,
    environment={
        "POWERTOOLS_SERVICE_NAME": "PowertoolsHelloWorld",
        "POWERTOOLS_METRICS_NAMESPACE": "PowertoolsSample",
        "LOG_LEVEL": "INFO"
    }
)

apigw = apigwv1.RestApi(self, "PowertoolsAPI",
    deploy_options=apigwv1.StageOptions(stage_name="dev"))

hello_api = apigw.root.add_resource("hello")
hello_api.add_method("GET", apigwv1.LambdaIntegration(function,
    proxy=True))

CfnOutput(self, "apiUrl", value=f"{apigw.url}hello")

```

## 8. Distribuisci l'applicazione.

```

cd ..
cdk deploy

```

## 9. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
'Stacks[0].Outputs[?OutputKey==`apiUrl`].OutputValue' --output text
```

## 10. Richiama l'endpoint dell'API:

```
curl GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

## 11. Per ottenere i log per la funzione, esegui [sam logs](#). Per ulteriori informazioni, consulta l'argomento relativo all'[utilizzo dei log](#) nella Guida per sviluppatori AWS Serverless Application Model .

```
sam logs --stack-name HelloWorldStack
```

L'output del log ha la struttura seguente:

```
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04
2025-02-03T14:59:50.371000 INIT_START Runtime Version:
python:3.9.v16 Runtime Version ARN: arn:aws:lambda:us-
east-1::runtime:07a48df201798d627f2b950f03bb227aab4a655a1d019c3296406f95937e2525
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.112000
START RequestId: d455cfc4-7704-46df-901b-2a5cce9405be Version: $LATEST
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.114000 {
  "level": "INFO",
  "location": "hello:23",
  "message": "Hello world API - HTTP 200",
  "timestamp": "2025-02-03 14:59:51,113+0000",
  "service": "PowertoolsHelloWorld",
  "cold_start": true,
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYt0c9j",
  "function_memory_size": "128",
  "function_arn": "arn:aws:lambda:us-east-1:111122223333:function:sam-app-
HelloWorldFunction-YBg8yfYt0c9j",
  "function_request_id": "d455cfc4-7704-46df-901b-2a5cce9405be",
  "correlation_id": "e73f8aef-5e07-436e-a30b-63e4b23f0047",
  "xray_trace_id": "1-63dd2166-434a12c22e1307ff2114f299"
}
```

```

2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "function_name",
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYt0c9j",
  "service": "PowertoolsHelloWorld",
  "ColdStart": [
    1.0
  ]
}
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "HelloWorldInvocations",
            "Unit": "Count"
          }
        ]
      }
    ]
  }
}

```

```
    }  
  ]  
},  
"service": "PowertoolsHelloWorld",  
"HelloWorldInvocations": [  
  1.0  
]  
}  
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.128000 END  
RequestId: d455cfc4-7704-46df-901b-2a5cce9405be  
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.128000  
REPORT RequestId: d455cfc4-7704-46df-901b-2a5cce9405be    Duration: 16.33 ms  
Billed Duration: 756 ms    Memory Size: 128 MB    Max Memory Used: 64 MB    Init  
Duration: 739.46 ms  
XRAY TraceId: 1-63dd2166-434a12c22e1307ff2114f299    SegmentId: 3c5d18d735a1ced0  
Sampled: true
```

12. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
cdk destroy
```

## AWS Lambda test delle funzioni in Python

### Note

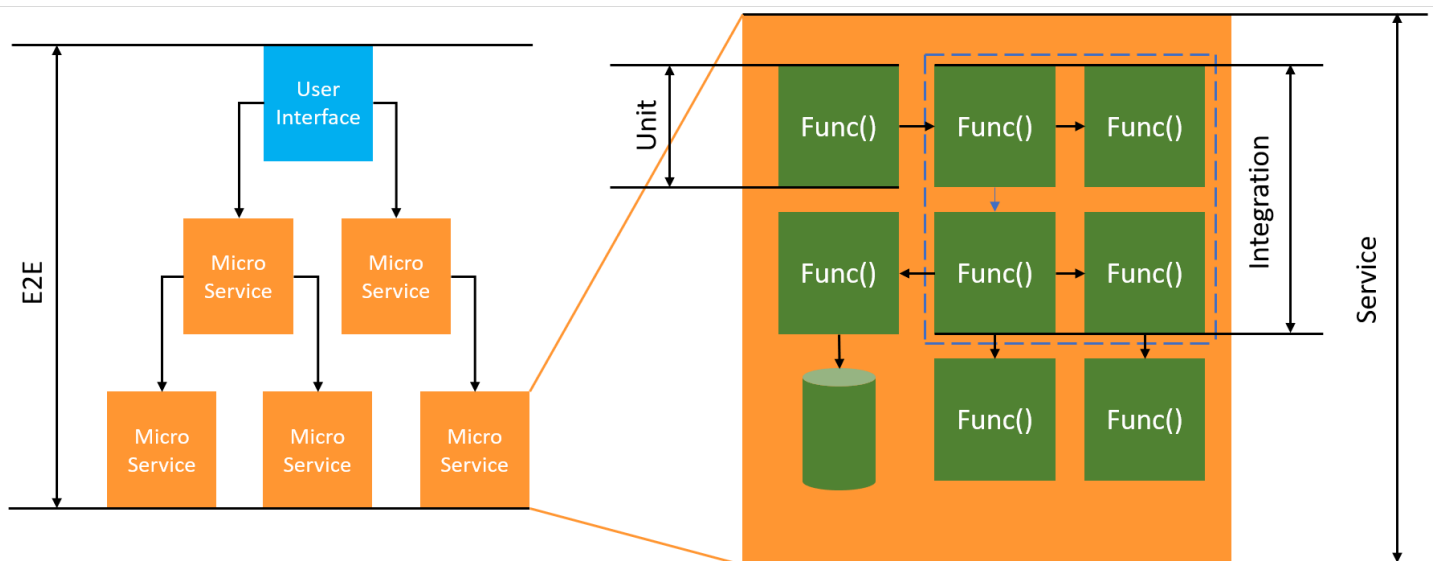
Consulta il capitolo sul [Test delle funzioni](#) per un'introduzione completa alle tecniche e alle best practice per testare soluzioni serverless.

Il test delle funzioni serverless utilizza tipi e tecniche di test tradizionali, ma è necessario considerare anche il test delle applicazioni serverless nel loro insieme. Cloud-based i test forniscono la misura più accurata della qualità sia delle funzioni che delle applicazioni serverless.

Un'architettura applicativa serverless include servizi gestiti che forniscono funzionalità applicative critiche tramite chiamate API. Pertanto, è fondamentale che il ciclo di sviluppo preveda test automatici in grado di verificare il corretto funzionamento dell'interazione tra le funzioni e i servizi.

Se non crei test basati sul cloud, potresti riscontrare problemi dovuti alle differenze tra l'ambiente locale e quello implementato. Il processo di integrazione continua dovrebbe eseguire test su un insieme di risorse con provisioning nel cloud prima di promuovere il codice nell'ambiente di implementazione successivo, come quello di controllo qualità (QA), staging o produzione.

Continua a leggere questa breve guida per scoprire le strategie di test per le applicazioni serverless oppure visita il [repository Serverless Test Samples](#) per approfondire esempi pratici, specifici per il linguaggio e il runtime selezionati.



Per i test serverless, è comunque necessario scrivere test unitari, di integrazione e end-to-end.

- **Test unitari:** vengono eseguiti su un blocco di codice isolato. Ad esempio, possono essere utilizzati per verificare la logica aziendale per calcolare le spese di spedizione in base a un articolo e a una destinazione specifici.
- **Test di integrazione:** coinvolgono due o più componenti o servizi che interagiscono, in genere in un ambiente cloud. Ad esempio, possono essere utilizzati per verificare che una funzione elabori gli eventi di una coda.
- **End-to-end test:** test che verificano il comportamento in un'intera applicazione. Ad esempio, possono essere utilizzati per verificare che l'infrastruttura sia configurata correttamente e che gli eventi fluiscano tra i servizi come previsto per registrare l'ordine di un cliente.

## Test delle applicazioni serverless

In genere si utilizzano una combinazione di approcci per testare il codice dell'applicazione serverless, tra cui test nel cloud, test con simulazioni e, occasionalmente, test con emulatori.

### Test nel cloud

I test nel cloud sono utili per tutte le fasi di test, inclusi i test unitari, i test di integrazione e i test end-to-end. Esegui test su codice implementato nel cloud e interagisci con servizi basati su cloud. Questo approccio fornisce la misura più accurata della qualità del codice.

Un modo conveniente per eseguire il debug di una funzione Lambda nel cloud è con un evento di test nella console. Un evento di test è un input JSON per la funzione. Se la funzione non richiede input, l'evento può essere un documento JSON ( `{}` ) vuoto. La console fornisce eventi di esempio per una varietà di integrazioni di servizi. Dopo aver creato un evento nella console, puoi condividerlo con il tuo team per rendere i test più semplici e coerenti.

#### Note

[Testare una funzione nella console](#) è un modo rapido per iniziare, ma l'automazione dei cicli di test garantisce la qualità delle applicazioni e la velocità di sviluppo.

## Strumenti di test

Esistono strumenti e tecniche per accelerare i cicli di feedback sullo sviluppo. Ad esempio, [AWS SAM Accelerate](#) e [AWS CDK watch mode](#) riducono entrambi il tempo necessario per aggiornare gli ambienti cloud.

[Moto](#) è una libreria Python per simulare AWS servizi e risorse, in modo da poter testare le funzioni con poche o nessuna modifica utilizzando decoratori per intercettare e simulare le risposte.

La funzionalità di convalida di [Powertools for AWS Lambda \(Python\)](#) fornisce decoratori che consentono di convalidare gli eventi di input e le risposte di output delle funzioni Python.

Per ulteriori informazioni, leggi il post sul blog [Unit Testing Lambda with Python and Mock Services.](#)  
[AWS](#)

Per ridurre la latenza associata alle iterazioni di implementazione del cloud, consulta la modalità di controllo [AWS Serverless Application Model \(AWS SAM\) Accelerate](#), [AWS Cloud Development Kit \(CDK\).](#)[AWS](#) Questi strumenti monitorano l'infrastruttura e il codice per rilevare eventuali modifiche. Reagiscono a questi cambiamenti creando e implementando automaticamente aggiornamenti incrementali nell'ambiente cloud.

Alcuni esempi che utilizzano questi strumenti sono disponibili nel repository di codice [Python Test Samples](#).



## Strumentazione del codice Python in AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzarlo X-Ray per tracciare una richiesta mentre attraversa le risorse dell'applicazione, che possono includere funzioni Lambda e altri servizi. AWS

Per inviare dati di tracciamento a X-Ray, puoi utilizzare una delle tre librerie SDK:

- [AWS Distro for OpenTelemetry \(ADOT\)](#): una distribuzione sicura, pronta per la produzione e AWS supportata dell'SDK (OtEL). OpenTelemetry
- [SDK AWS X-Ray per Python](#)— Un SDK per la generazione e l'invio di dati di traccia a X-Ray
- [Powertools for AWS Lambda \(Python\)](#): un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori.

Ciascuno degli SDK offre modi per inviare i dati di telemetria al servizio. X-Ray Puoi quindi utilizzarlo X-Ray per visualizzare, filtrare e ottenere informazioni dettagliate sulle metriche delle prestazioni dell'applicazione per identificare problemi e opportunità di ottimizzazione.

### Important

The X-Ray e Powertools for AWS Lambda SDK fanno parte di una soluzione di strumentazione strettamente integrata offerta da AWS I livelli Lambda ADOT fanno parte di uno standard di settore per la strumentazione di tracciamento che in generale raccoglie più dati, ma potrebbero non essere adatti a tutti i casi d'uso. È possibile implementare il tracciamento end-to-end utilizzando entrambe le soluzioni. X-Ray Per saperne di più sulla scelta tra di esse, consulta [Scelta tra AWS Distro for Open Telemetry e SDK. X-Ray](#)

## Sections

- [Usare Powertools per AWS Lambda \(Python\) e AWS SAM per tracciare](#)
- [Usare Powertools per AWS Lambda \(Python\) e il AWS CDK per tracciare](#)
- [Utilizzo di ADOT per strumentare le funzioni Python](#)
- [Usare l' X-Ray SDK per strumentare le tue funzioni Python](#)
- [Attivazione del tracciamento con la console Lambda](#)
- [Attivazione del tracciamento con l'API Lambda](#)
- [Attivazione del tracciamento con CloudFormation](#)

- [Interpretazione e tracciamento X-Ray](#)
- [Archiviazione delle dipendenze di runtime in un livello \(SDK\) X-Ray](#)

Usare Powertools per AWS Lambda (Python) e AWS SAM per tracciare

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Hello World Python di esempio con moduli [Powertools for AWS Lambda \(Python\) integrati utilizzando](#). AWS SAM Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format a e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio hello world.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Python 3.11
- [AWS CLI versione 2](#)
- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI. AWS SAM](#)

## Distribuisci un esempio AWS SAM applicazione

1. Inizializza l'applicazione utilizzando il modello Python Hello World.

```
sam init --app-template hello-world-powertools-python --name sam-app --package-type Zip --runtime python3.11 --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi Enter.

 Note

Perché HelloWorldFunction potrebbe non avere un'autorizzazione definita, va bene? , assicurati di entrarey.

## 5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query  
'Stacks[0].Outputs[?OutputKey=='HelloWorldApi'].OutputValue' --output text
```

## 6. Richiama l'endpoint dell'API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere le tracce per la funzione, esegui [sam traces](#).

```
sam traces
```

L'output della traccia ha il seguente aspetto:

```
New XRay Service Graph  
  Start time: 2023-02-03 14:59:50+00:00  
  End time: 2023-02-03 14:59:50+00:00  
  Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-YBg8yfYt0c9j -  
  Edges: [1]  
    Summary_statistics:  
      - total requests: 1  
      - ok count(2XX): 1  
      - error count(4XX): 0  
      - fault count(5XX): 0  
      - total response time: 0.924  
  Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-YBg8yfYt0c9j  
  - Edges: []  
    Summary_statistics:  
      - total requests: 1  
      - ok count(2XX): 1
```

```

- error count(4XX): 0
- fault count(5XX): 0
- total response time: 0.016
Reference Id: 2 - client - sam-app-HelloWorldFunction-YBg8yfYt0c9j - Edges: [0]
Summary_statistics:
- total requests: 0
- ok count(2XX): 0
- error count(4XX): 0
- fault count(5XX): 0
- total response time: 0

XRay Event [revision 1] at (2023-02-03T14:59:50.204000) with id
(1-63dd2166-434a12c22e1307ff2114f299) and duration (0.924s)
- 0.924s - sam-app-HelloWorldFunction-YBg8yfYt0c9j [HTTP: 200]
- 0.016s - sam-app-HelloWorldFunction-YBg8yfYt0c9j
- 0.739s - Initialization
- 0.016s - Invocation
- 0.013s - ## lambda_handler
- 0.000s - ## app.hello
- 0.000s - Overhead

```

8. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

X-Ray non traccia tutte le richieste alla tua applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento sia efficiente, fornendo comunque un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non puoi configurare la frequenza di X-Ray campionamento per le tue funzioni.

Usare Powertools per AWS Lambda (Python) e il AWS CDK per tracciare

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Hello World Python di esempio con moduli [Powertools for AWS Lambda \(Python\) integrati utilizzando](#). AWS CDK Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log

e metriche utilizzando Embedded Metric Format a e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio hello world.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Python 3.11
- [AWS CLI versione 2](#)
- [AWS CDK versione 2](#)
- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI. AWS SAM](#)

## Distribuisci un esempio AWS CDK applicazione

1. Crea una directory di progetto per la nuova applicazione.

```
mkdir hello-world  
cd hello-world
```

2. Inizializza l'app.

```
cdk init app --language python
```

3. Installa le dipendenze di Python.

```
pip install -r requirements.txt
```

4. Crea una directory lambda\_function nella cartella root.

```
mkdir lambda_function  
cd lambda_function
```

5. Crea un file app.py e aggiungi il seguente codice al file. Questo è il codice per la funzione Lambda.

```
from aws_lambda_powertools.event_handler import APIGatewayRestResolver  
from aws_lambda_powertools.utilities.typing import LambdaContext  
from aws_lambda_powertools.logging import correlation_paths  
from aws_lambda_powertools import Logger
```

```

from aws_lambda_powertools import Tracer
from aws_lambda_powertools import Metrics
from aws_lambda_powertools.metrics import MetricUnit

app = APIGatewayRestResolver()
tracer = Tracer()
logger = Logger()
metrics = Metrics(namespace="PowertoolsSample")

@app.get("/hello")
@tracer.capture_method
def hello():
    # adding custom metrics
    # See: https://docs.powertools.aws.dev/lambda-python/latest/core/metrics/
    metrics.add_metric(name="HelloWorldInvocations", unit=MetricUnit.Count,
value=1)

    # structured log
    # See: https://docs.powertools.aws.dev/lambda-python/latest/core/logger/
    logger.info("Hello world API - HTTP 200")
    return {"message": "hello world"}

# Enrich logging with contextual information from Lambda
@logger.inject_lambda_context(correlation_id_path=correlation_paths.API_GATEWAY_REST)
# Adding tracer
# See: https://docs.powertools.aws.dev/lambda-python/latest/core/tracer/
@tracer.capture_lambda_handler
# ensures metrics are flushed upon request completion/failure and capturing
ColdStart metric
@metrics.log_metrics(capture_cold_start_metric=True)
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    return app.resolve(event, context)

```

6. Apri la directory `hello_world`. Dovrebbe essere visualizzato un file denominato `hello_world_stack.py`.

```

cd ..
cd hello_world

```

7. Apri `hello_world_stack.py` e aggiungi il seguente codice al file. Contiene il [Lambda Constructor](#), che crea la funzione Lambda, configura le variabili di ambiente per Powertools e imposta la conservazione dei log su una settimana, e il Constructor [ApiGatewayv 1, che crea l'API](#) REST.

```

from aws_cdk import (
    Stack,
    aws_apigateway as apigwv1,
    aws_lambda as lambda_,
    CfnOutput,
    Duration
)
from constructs import Construct

class HelloWorldStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # Powertools Lambda Layer
        powertools_layer = lambda_.LayerVersion.from_layer_version_arn(
            self,
            id="lambda-powertools",
            # At the moment we wrote this example, the aws_lambda_python_alpha CDK
            # constructor is in Alpha, so we use layer to make the example simpler
            # See https://docs.aws.amazon.com/cdk/api/v2/python/
            aws_cdk.aws_lambda_python_alpha/README.html
            # Check all Powertools layers versions here: https://
            docs.powertools.aws.dev/lambda-python/latest/#lambda-layer
            layer_version_arn=f"arn:aws:lambda:
{self.region}:017000801446:layer:AWSLambdaPowertoolsPythonV2:21"
        )

        function = lambda_.Function(self,
            'sample-app-lambda',
            runtime=lambda_.Runtime.PYTHON_3_11,
            layers=[powertools_layer],
            code = lambda_.Code.from_asset("./lambda_function/"),
            handler="app.lambda_handler",
            memory_size=128,
            timeout=Duration.seconds(3),
            architecture=lambda_.Architecture.X86_64,
            environment={
                "POWERTOOLS_SERVICE_NAME": "PowertoolsHelloWorld",
                "POWERTOOLS_METRICS_NAMESPACE": "PowertoolsSample",
                "LOG_LEVEL": "INFO"
            }
        )

```

```

    apigw = apigwv1.RestApi(self, "PowertoolsAPI",
    deploy_options=apigwv1.StageOptions(stage_name="dev"))

    hello_api = apigw.root.add_resource("hello")
    hello_api.add_method("GET", apigwv1.LambdaIntegration(function,
    proxy=True))

    CfnOutput(self, "apiUrl", value=f"{apigw.url}hello")

```

## 8. Distribuisci l'applicazione.

```

cd ..
cdk deploy

```

## 9. Ottieni l'URL dell'applicazione implementata:

```

aws cloudformation describe-stacks --stack-name HelloWorldStack --query
'Stacks[0].Outputs[?OutputKey==`apiUrl`].OutputValue' --output text

```

## 10. Richiama l'endpoint dell'API:

```

curl -X GET <URL_FROM_PREVIOUS_STEP>

```

In caso di esito positivo, vedrai questa risposta:

```

{"message":"hello world"}

```

## 11. Per ottenere le tracce per la funzione, esegui [sam traces](#).

```

sam traces

```

L'output delle tracce ha la struttura seguente:

```

New XRay Service Graph
  Start time: 2023-02-03 14:59:50+00:00
  End time: 2023-02-03 14:59:50+00:00
  Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-YBg8yfYt0c9j -
  Edges: [1]
  Summary_statistics:
    - total requests: 1
    - ok count(2XX): 1

```



```

- error count(4XX): 0
- fault count(5XX): 0
- total response time: 0.924
Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-YBg8yfYt0c9j
- Edges: []
Summary_statistics:
- total requests: 1
- ok count(2XX): 1
- error count(4XX): 0
- fault count(5XX): 0
- total response time: 0.016
Reference Id: 2 - client - sam-app-HelloWorldFunction-YBg8yfYt0c9j - Edges: [0]
Summary_statistics:
- total requests: 0
- ok count(2XX): 0
- error count(4XX): 0
- fault count(5XX): 0
- total response time: 0

XRay Event [revision 1] at (2023-02-03T14:59:50.204000) with id
(1-63dd2166-434a12c22e1307ff2114f299) and duration (0.924s)
- 0.924s - sam-app-HelloWorldFunction-YBg8yfYt0c9j [HTTP: 200]
- 0.016s - sam-app-HelloWorldFunction-YBg8yfYt0c9j
- 0.739s - Initialization
- 0.016s - Invocation
- 0.013s - ## lambda_handler
  - 0.000s - ## app.hello
- 0.000s - Overhead

```

12. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
cdk destroy
```

## Utilizzo di ADOT per strumentare le funzioni Python

ADOT fornisce [livelli](#) Lambda completamente gestiti che mettono insieme tutto il necessario per raccogliere i dati di telemetria utilizzando l'SDK OTel. Usando questo livello, è possibile strumentare le funzioni Lambda senza dover modificare alcun codice funzione. È inoltre possibile configurare il livello per eseguire l'inizializzazione personalizzata di OTel. Per ulteriori informazioni, consulta

la sezione relativa alla [configurazione personalizzata per ADOT Collector su Lambda](#) nella documentazione di ADOT.

Per i runtime Python, è possibile aggiungere il livello Lambda gestito da AWS per ADOT Python per la strumentazione automatica delle funzioni. Questo livello funziona sia per architetture arm64 che x86\_64. Per istruzioni dettagliate su come aggiungere questo livello, vedi [AWS Distro for OpenTelemetry Lambda Support for Python nella documentazione ADOT](#).

Usare l' X-Ray SDK per strumentare le tue funzioni Python

Per registrare i dettagli sulle chiamate effettuate dalla funzione Lambda ad altre risorse nell'applicazione, è anche possibile utilizzare il SDK AWS X-Ray per Python. Per ottenere l'SDK, aggiungere il pacchetto `aws-xray-sdk` alle dipendenze dell'applicazione.

Example [requirements.txt](#)

```
jsonpickle==1.3
aws-xray-sdk==2.4.3
```

Nel codice della funzione, puoi configurare i client AWS SDK applicando una patch alla `boto3` libreria con il modulo `aws_xray_sdk.core`

Example [funzione — Tracciamento di un AWS Client SDK](#)

```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

logger = logging.getLogger()
logger.setLevel(logging.INFO)
patch_all()

client = boto3.client('lambda')
client.get_account_settings()

def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES\r' + jsonpickle.encode(dict(**os.environ)))
    ...
```

Dopo aver aggiunto le dipendenze corrette e apportato le modifiche al codice necessarie, attiva il tracciamento nella configurazione della funzione tramite la console Lambda o l'API.

## Attivazione del tracciamento con la console Lambda

Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. In Strumenti di monitoraggio aggiuntivi scegli Modifica.
5. In CloudWatch Application Signals and AWS X-Ray, scegli **Abilita** per le tracce del servizio Lambda.
6. Scegli Save (Salva).

## Attivazione del tracciamento con l'API Lambda

Configura il tracciamento sulla tua funzione Lambda con AWS CLI or AWS SDK, utilizza le seguenti operazioni API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

Il seguente AWS CLI comando di esempio abilita la traccia attiva su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

## Attivazione del tracciamento con CloudFormation

Per attivare la traccia su una `AWS::Lambda::Function` risorsa in un CloudFormation modello, utilizzate la proprietà `TracingConfig`

Example [function-inline.yml](#) — Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Per una `AWS::Serverless::Function` risorsa AWS Serverless Application Model (AWS SAM), usa la `Tracing` proprietà.

Example [template.yml](#) — Configurazione di tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

## Interpretazione e tracciamento X-Ray

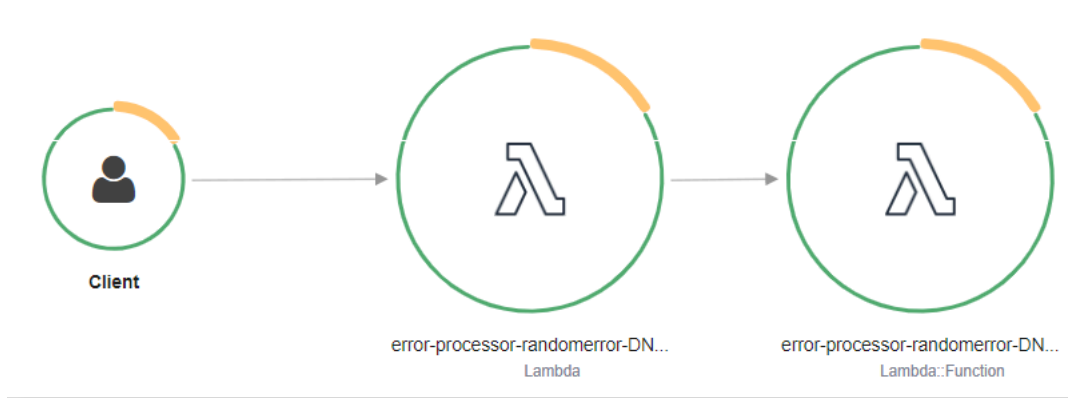
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al [ruolo di esecuzione](#) della funzione. Altrimenti, aggiungi la [AWSXRayDaemonWriteAccess](#) policy al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico del [X-Ray servizio](#) mostra le informazioni sull'applicazione e su tutti i suoi componenti. Il seguente esempio mostra un'applicazione con due funzioni. La funzione principale elabora gli eventi e talvolta restituisce errori. La seconda funzione in alto elabora gli errori che appaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon CloudWatch Logs.

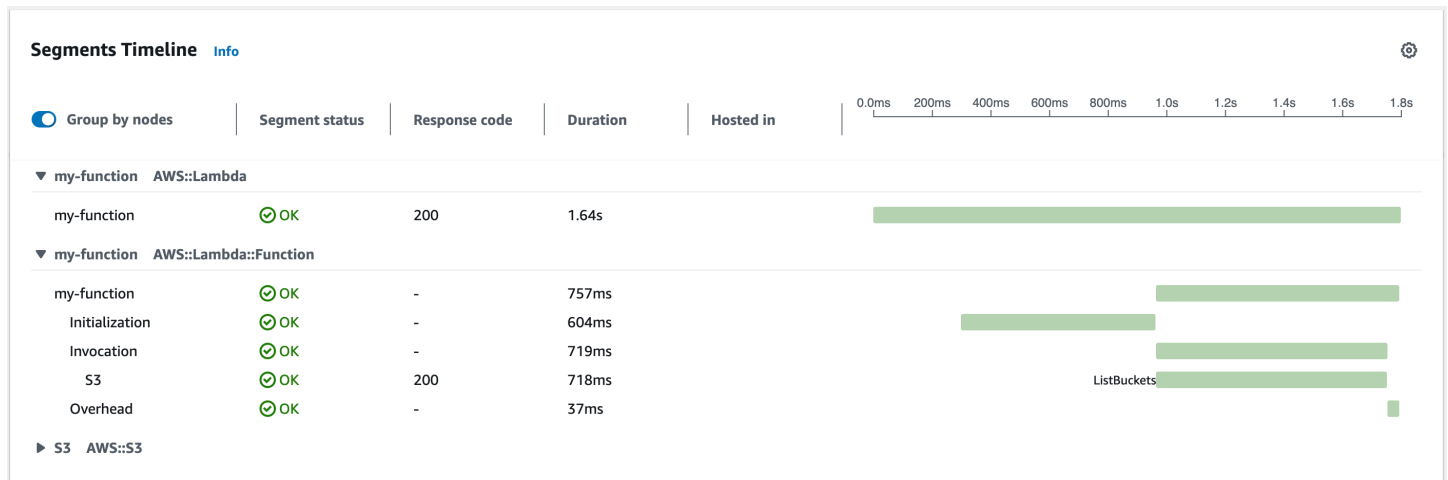


X-Ray non traccia tutte le richieste all'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento sia efficiente, fornendo comunque un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non puoi configurare la frequenza di X-Ray campionamento per le tue funzioni.

In X-Ray, una traccia registra le informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.

### Note

AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti.

Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regions, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia nel nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- **Inizializzazione** – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il [codice di inizializzazione](#). Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.
- **Chiamata**: rappresenta il tempo impiegato per eseguire il codice del gestore.
- **Overhead**: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla

struttura dei segmenti di funzioni vecchio e nuovo stile, consulta [the section called “Comprensione delle tracce X-Ray”](#).

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta [SDK AWS X-Ray per Python](#) nella Guida per gli sviluppatori di AWS X-Ray .

### Prezzi

Puoi utilizzare il X-Ray tracciamento gratuitamente ogni mese fino a un determinato limite come parte del piano gratuito. AWS Oltre tale soglia, vengono X-Ray addebitati i costi per l'archiviazione e il recupero delle tracce. Per ulteriori informazioni, consultare [Prezzi di AWS X-Ray](#).

Archiviazione delle dipendenze di runtime in un livello (SDK) X-Ray

Se si utilizza l' X-Ray SDK per configurare il codice funzione dei client AWS SDK, il pacchetto di distribuzione può diventare piuttosto ingombrante. Per evitare di caricare le dipendenze di runtime ogni volta che aggiorni il codice funzione, impacchetta l' X-Ray SDK in un layer Lambda. [???](#)

L'esempio seguente mostra una risorsa `AWS::Serverless::LayerVersion` che memorizza SDK AWS X-Ray per Python.

Example [template.yml](#) — Livello delle dipendenze

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/.
      Tracing: Active
      Layers:
        - !Ref libs
        ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-python-lib
      Description: Dependencies for the blank-python sample app.
      ContentUri: package/.
```

**CompatibleRuntimes:**  
- **python3.11**

Con questa configurazione, si aggiorna il livello della libreria solo se si modificano le dipendenze di runtime. Poiché il pacchetto di implementazione della funzione contiene solo il codice, questo può contribuire a ridurre i tempi di caricamento.

La creazione di un livello per le dipendenze richiede modifiche alla compilazione per generare l'archivio dei livelli prima della distribuzione. Per un esempio funzionante, vedere l'applicazione di esempio [blank-python](#).

## Compilazione di funzioni Lambda con Ruby

Puoi eseguire il codice Ruby in AWS Lambda. Lambda fornisce [Runtime](#) per Ruby che eseguono il tuo codice per elaborare gli eventi. Il codice viene eseguito in un ambiente che include AWS SDK per Ruby, con le credenziali di un ruolo AWS Identity and Access Management (IAM) che gestisci. Per saperne di più sulle versioni degli SDK incluse nei runtime di Ruby, consulta [the section called “Runtime-included Versions SDK”](#).

Lambda supporta i seguenti runtime di Ruby.

| Nome     | Identificatore | Sistema operativo | Data di ritiro | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|----------|----------------|-------------------|----------------|---------------------------|-------------------------------|
| Ruby 4.0 | ruby4.0        | Amazon Linux 2023 | 31 marzo 2029  | 30 aprile 2029            | 31 maggio 2029                |
| Ruby 3.4 | ruby3.4        | Amazon Linux 2023 | 31 marzo 2028  | 30 aprile 2028            | 31 maggio 2028                |
| Ruby 3.3 | ruby3.3        | Amazon Linux 2023 | 31 marzo 2027  | 30 aprile 2027            | 31 maggio 2027                |

Per creare una funzione Ruby

1. Aprire la [console Lambda](#).
2. Scegli Crea funzione.



### 3. Configura le impostazioni seguenti:

- Nome della funzione: inserisci il nome della funzione.
- Runtime: scegli Ruby 4.0.

### 4. Scegli Crea funzione.

La console crea una funzione Lambda con un singolo file di origine denominato `lambda_function.rb`. È possibile modificare questo file e aggiungere altri file nell'editor di codice predefinito. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione. Quindi, per eseguire il codice, scegli Crea evento di test nella sezione EVENTI DI TEST.

Il file `lambda_function.rb` esporta una funzione denominata `lambda_handler` che richiede un oggetto evento e un oggetto contesto. Questa è la [funzione del gestore](#) chiamata da Lambda quando la funzione viene richiamata. Il runtime della funzione Ruby riceve gli eventi di chiamata da Lambda e li passa al gestore. Nella configurazione della funzione il valore del gestore è `lambda_function.lambda_handler`.

Quando si salva il codice funzione, la console Lambda crea un pacchetto di implementazione dell'archivio di file .zip. Quando sviluppi il codice funzione al di fuori della console (utilizzando un IDE) devi [creare un pacchetto di implementazione](#) per caricare il codice nella funzione Lambda.

Il runtime della funzione passa un oggetto contesto al gestore, oltre all'evento di chiamata. L'[oggetto contesto](#) contiene ulteriori informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. Altre informazioni sono disponibili con le variabili di ambiente.

La funzione Lambda include un gruppo di CloudWatch log Logs. Il runtime della funzione invia i dettagli su ogni chiamata a Logs. CloudWatch Si trasmette qualsiasi [log che la tua funzione emette](#) durante la chiamata. Se la funzione restituisce un errore, Lambda formatta l'errore e lo restituisce al chiamante.

### Argomenti

- [Runtime-included Versioni SDK](#)
- [Abilitazione di Yet Another Ruby JIT \(YJIT\)](#)
- [Definire l'handler di funzioni Lambda in Ruby](#)
- [Distribuire le funzioni Ruby Lambda con gli archivi di file .zip](#)
- [Distribuisce funzioni Lambda per Ruby con immagini di container](#)
- [Utilizzo dei livelli per le funzioni Lambda di Ruby](#)

- [Utilizzo dell'oggetto del contesto Lambda per recuperare le informazioni sulla funzione Ruby](#)
- [Registrare e monitorare le funzioni Lambda con Ruby](#)
- [Strumentazione del codice Ruby in AWS Lambda](#)

## Runtime-included Versioni SDK

La versione dell' AWS SDK inclusa nel runtime di Ruby dipende dalla versione di runtime e dalla tua. Regione AWS L' AWS SDK for Ruby è progettato per essere modulare ed è separato da. Servizio AWS Per trovare la versione di una determinata gemma di servizio inclusa nel runtime che stai utilizzando, crea una funzione Lambda con il codice nel seguente formato. Sostituisci `aws-sdk-s3` e `Aws::S3` con il nome delle gemme di servizio utilizzate dal tuo codice.

```
require 'aws-sdk-s3'

def lambda_handler(event:, context:)
  puts "Service gem version: #{Aws::S3::GEM_VERSION}"
  puts "Core version: #{Aws::CORE_GEM_VERSION}"
end
```

## Abilitazione di Yet Another Ruby JIT (YJIT)

I runtime Ruby supportano [YJIT, un compilatore Ruby JIT](#) leggero e minimalista. YJIT offre prestazioni significativamente più elevate, ma utilizza anche più memoria rispetto all'interprete Ruby. YJIT è consigliato per i carichi di lavoro Ruby on Rails.

YJIT non è abilitato per impostazione predefinita. Per abilitare YJIT per una funzione Ruby, imposta la variabile di ambiente su. `RUBY_YJIT_ENABLE 1` Per verificare che YJIT sia abilitato, stampa il risultato del metodo `RubyVM::YJIT.enabled?`.

Example- Conferma che YJIT è abilitato

```
puts(RubyVM::YJIT.enabled?())
# => true
```

## Definire l'handler di funzioni Lambda in Ruby

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

### Argomenti

- [Nozioni di base sull'handler Ruby](#)
- [Best practice di codice per le funzioni Lambda con Ruby](#)

### Nozioni di base sull'handler Ruby

In questo esempio il file `function.rb` definisce un metodo del gestore denominato `handler`. La funzione del gestore richiede due oggetti come input e restituisce un documento JSON.

#### Example function.rb

```
require 'json'

def handler(event:, context:)
  { event: JSON.generate(event), context: JSON.generate(context.inspect) }
end
```

Nella configurazione della funzione, l'impostazione `handler` indica a Lambda dove trovare il gestore. Nell'esempio precedente, il valore corretto per questa impostazione è **`function.handler`**. Include due nomi separati da un punto: il nome del file e il nome del metodo del gestore.

È inoltre possibile definire il metodo del gestore in una classe. L'esempio seguente definisce il metodo del gestore `process` nella classe `Handler` nel modulo `LambdaFunctions`.

#### Example source.rb

```
module LambdaFunctions
  class Handler
    def self.process(event:, context:)
      "Hello!"
    end
  end
end
```

In questo caso, l'impostazione del gestore è **`source.LambdaFunctions::Handler.process`**.

I due oggetti che il gestore accetta sono il contesto e l'evento di chiamata. L'evento è un oggetto Ruby che contiene il payload fornito dal chiamante. Se il payload è un documento JSON, l'oggetto evento è un hash Ruby. In caso contrario, è una stringa. L'[oggetto contesto](#) include i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

Il gestore della funzione viene eseguito ogni volta che la funzione Lambda viene richiamata. Il codice statico all'esterno del gestore viene eseguito una volta per istanza della funzione. Se il gestore utilizza risorse come client SDK e connessioni database, puoi crearle al di fuori del metodo del gestore per riutilizzarle per più chiamate.

Ogni istanza della funzione può elaborare più eventi di chiamata, ma elabora un solo evento alla volta. Il numero di istanze che elaborano un evento in un dato momento costituisce la simultaneità della funzione. Per ulteriori informazioni sull'ambiente di esecuzione Lambda, consulta [Comprendere il ciclo di vita dell'ambiente di esecuzione Lambda](#).

Best practice di codice per le funzioni Lambda con Ruby

Segui le linee guida riportate nell'elenco seguente per utilizzare le best practice di codifica durante la creazione delle funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. Ciò semplifica la creazione di una funzione più testabile dall'unità. Ad esempio, in Ruby, potrebbe assomigliare a:

```
def lambda_handler(event:, context:)
  foo = event['foo']
  bar = event['bar']

  result = my_lambda_function(foo:, bar:)
end

def my_lambda_function(foo:, bar:)
  // MyLambdaFunction logic here
end
```

- Controllare le dipendenze nel pacchetto di distribuzione della funzione. L'ambiente di esecuzione AWS Lambda contiene diverse librerie. Per il runtime di Ruby, questi includono l' AWS SDK.

Per abilitare l'ultimo set di funzionalità e aggiornamenti di sicurezza, Lambda aggiorna periodicamente queste librerie. Questi aggiornamenti potrebbero introdurre lievi modifiche al comportamento della funzione Lambda. Per mantenere il controllo completo delle dipendenze utilizzate dalla funzione, inserire tutte le dipendenze nel pacchetto di implementazione.

- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del [contesto di esecuzione](#).
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. Ciò riduce il tempo necessario per scaricare e decomprimere il pacchetto di distribuzione prima dell'invocazione. Per le funzioni create in Ruby, evita di caricare l'intera libreria AWS SDK come parte del pacchetto di distribuzione. Dipendi invece in modo selettivo dalle gemme che raccolgono i componenti dell'SDK di cui hai bisogno (ad esempio, le gemme SDK DynamoDB o Amazon S3).

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, consulta [Reusing Connections with in. Keep-Alive Node.js](#)

Utilizzare [le variabili di ambiente](#) per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume

indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su `0` per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionali alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha una dipendenza su queste API non pubbliche. Consulta il [riferimento all'API](#) per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta [Come posso rendere idempotente la mia funzione Lambda?](#).

## Distribuire le funzioni Ruby Lambda con gli archivi di file .zip

Il codice della tua AWS Lambda funzione comprende un file .rb contenente il codice del gestore della tua funzione, insieme a eventuali dipendenze aggiuntive (gemme) da cui dipende il tuo codice. Per implementare questo codice della funzione in Lambda, utilizza un pacchetto di implementazione. Questo pacchetto può essere un archivio di file .zip o un'immagine di container. Per ulteriori informazioni sull'uso delle immagini di container con Ruby, consulta la pagina [Implementazione di funzioni Lambda in Ruby con immagini di container](#).

Per creare un pacchetto di implementazione come archivio di file .zip, puoi utilizzare l'utilità di archiviazione di file .zip incorporata del tuo strumento della linea di comando o qualsiasi altra utilità file .zip, come ad esempio [7zip](#). Gli esempi mostrati nelle sezioni seguenti presuppongono che tu stia utilizzando uno strumento della linea di comando zip in un ambiente Linux o MacOS. Per utilizzare gli stessi comandi in Windows, puoi [installare il sottosistema Windows per Linux per](#) ottenere una Windows-integrated versione di Ubuntu e Bash.

Nota che Lambda utilizza le autorizzazioni dei file POSIX, quindi potresti aver bisogno di [impostare le autorizzazioni per la cartella del pacchetto di implementazione](#) prima di creare l'archivio di file .zip.

I comandi di esempio nelle sezioni seguenti utilizzano l'utilità [Creatore di bundle](#) per aggiungere dipendenze al pacchetto di implementazione. Per installare il creatore di bundle, esegui il comando sotto riportato.

```
gem install bundler
```

### Sections

- [Dipendenze in Ruby](#)
- [Creazione di un pacchetto di implementazione .zip senza dipendenze](#)
- [Creazione di un pacchetto di implementazione .zip con dipendenze](#)
- [Creazione di un livello Ruby per dipendenze](#)
- [Creazione di un pacchetto di implementazione .zip con librerie native](#)
- [Creazione e aggiornamento delle funzioni Lambda di Ruby utilizzando file .zip](#)

### Dipendenze in Ruby

Per le funzioni Lambda che utilizzano il runtime di Ruby, una dipendenza può essere una qualsiasi gemma Ruby. Se implementi la funzione utilizzando un archivio .zip, puoi aggiungere queste

dependenze al file .zip con il tuo codice della funzione o utilizzare un livello Lambda. Un livello è un file .zip separato che può contenere codice aggiuntivo o altri contenuti. Per maggiori informazioni sull'uso dei livelli Lambda, consulta [the section called “Livelli”](#).

Il runtime di Ruby include AWS SDK per Ruby. Se la tua funzione utilizza l'SDK, non è necessario raggrupparla con il codice. Tuttavia, per mantenere il pieno controllo delle tue dipendenze o per utilizzare una versione specifica dell'SDK, puoi aggiungerlo al pacchetto di implementazione della tua funzione.

Puoi includere l'SDK nel tuo file .zip o aggiungerlo utilizzando un livello Lambda. Le dipendenze nel file .zip o nei livelli Lambda hanno la precedenza sulle versioni incluse nel runtime. Per vedere quale versione dell'SDK per Ruby è inclusa nella tua versione di runtime, consulta [the section called “Runtime-included Versions SDK”](#).

In base al [modello di responsabilità condivisa di AWS](#), è tua responsabilità gestire eventuali dipendenze nei pacchetti di implementazione delle tue funzioni. Ciò include l'applicazione di aggiornamenti e patch di sicurezza. Per aggiornare le dipendenze nel pacchetto di implementazione della funzione, crea prima un nuovo file .zip e poi caricalo su Lambda. Per ulteriori informazioni, consulta [Creazione di un pacchetto di implementazione .zip con dipendenze](#) e [Creazione e aggiornamento delle funzioni Lambda di Ruby utilizzando file .zip](#).

### Creazione di un pacchetto di implementazione .zip senza dipendenze

Se il codice della funzione non ha dipendenze, il file .zip contiene solo il file .rb con il codice del gestore della funzione. Utilizza il tuo strumento di compressione preferito per creare un file .zip con il file .rb nella directory principale. Se il file .rb non è alla radice del file .zip, Lambda non sarà in grado di eseguire il codice.

Per informazioni su come implementare il file .zip per creare una nuova funzione Lambda o aggiornarne una esistente, consulta la sezione [Creazione e aggiornamento delle funzioni Lambda di Ruby utilizzando file .zip](#).

### Creazione di un pacchetto di implementazione .zip con dipendenze

Se il codice della funzione dipende da gem Ruby aggiuntivi, puoi aggiungere queste dipendenze al file .zip con il codice della funzione o utilizzare un [livello Lambda](#). Le istruzioni in questa sezione mostrano come includere le dipendenze nel pacchetto di implementazione .zip. Per istruzioni sull'inclusione di dipendenze in un livello, consulta [the section called “Creazione di un livello Ruby per dipendenze”](#).



Supponiamo che il codice della funzione sia salvato in un file denominato `lambda_function.rb` nella directory del progetto. I seguenti comandi della CLI di esempio creano un file `.zip` denominato `my_deployment_package.zip` contenente il codice della funzione e le relative dipendenze.

Per creare il pacchetto di implementazione

1. Nella directory del progetto, crea un `Gemfile` in cui specificare le tue dipendenze.

```
bundle init
```

2. Utilizzando il tuo editor di testo preferito, modifica il `Gemfile` per specificare le dipendenze della tua funzione. Ad esempio, per utilizzare la gemma `TZInfo`, modifica il tuo `Gemfile` in modo che assomigli a quanto segue.

```
source "https://rubygems.org"  
gem "tzinfo"
```

3. Esegui il comando sotto riportato per installare le gemme specificate nel `Gemfile` nella tua directory del progetto. Questo comando imposta `vendor/bundle` come percorso predefinito per le installazioni delle gemme.

```
bundle config set --local path 'vendor/bundle' && bundle install
```

Verrà visualizzato un output simile al seguente.

```
Fetching gem metadata from https://rubygems.org/.....  
Resolving dependencies...  
Using bundler 2.4.13  
Fetching tzinfo 2.0.6  
Installing tzinfo 2.0.6  
...
```

#### Note

Per installare nuovamente le gemme a livello globale in un secondo momento, esegui il comando sotto riportato.

```
bundle config set --local system 'true'
```

4. Crea un archivio di file .zip contenente il file `lambda_function.rb` con il codice del gestore della funzione e le dipendenze che hai installato nel passaggio precedente.

```
zip -r my_deployment_package.zip lambda_function.rb vendor
```

Verrà visualizzato un output simile al seguente.

```
adding: lambda_function.rb (deflated 37%)
  adding: vendor/ (stored 0%)
  adding: vendor/bundle/ (stored 0%)
  adding: vendor/bundle/ruby/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/build_info/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/cache/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/cache/aws-eventstream-1.0.1.gem (deflated 36%)
...
```

## Creazione di un livello Ruby per dipendenze

Per informazioni su come impacchettare le dipendenze di Ruby in un livello Lambda, consulta [the section called “Livelli”](#).

## Creazione di un pacchetto di implementazione .zip con librerie native

Molte gemme comuni di Rubynokogiri, ad esempio `io4r`, e `mysql` contengono estensioni native scritte in C. Quando aggiungi librerie contenenti codice C al tuo pacchetto di distribuzione, devi compilare il pacchetto correttamente per assicurarti che sia compatibile con l'ambiente di esecuzione Lambda.

Per le applicazioni di produzione, consigliamo di compilare e distribuire il codice utilizzando `aws-sam build`. AWS Serverless Application Model (AWS SAM) In AWS SAM uso `aws-sam build --use-container` opzione per creare la funzione all'interno di un contenitore Lambda-like Docker. Per saperne di più sull'utilizzo AWS SAM del codice della funzione, consulta la sezione [Building applications](#) nella AWS SAM Developer Guide.

Per creare un pacchetto di distribuzione .zip contenente gemme con estensioni native senza utilizzarlo AWS SAM, puoi in alternativa utilizzare un contenitore per raggruppare le tue dipendenze in un ambiente uguale all'ambiente di runtime Lambda Ruby. Per completare questi passaggi, Docker

deve essere installato sulla tua macchina di compilazione. Per ulteriori informazioni sull'installazione di Docker, consulta la pagina [Install Docker Engine](#).

### Creazione di un pacchetto di implementazione .zip in un container Docker

1. Sulla tua macchina di compilazione locale, crea una cartella in cui salvare il container. All'interno di quella cartella, crea un file denominato `dockerfile` e incolla il seguente codice.

```
FROM public.ecr.aws/sam/build-ruby3.2:latest-x86_64
RUN gem update bundler
CMD "/bin/bash"
```

2. All'interno della cartella in cui hai creato il tuo `dockerfile`, esegui il comando sotto riportato per creare il container Docker.

```
docker build -t awsruby32 .
```

3. Accedi alla directory del progetto contenente il file `.rb` con il codice del gestore della funzione e il `Gemfile` specificato nelle dipendenze della funzione. Dall'interno di quella directory, esegui il comando sotto riportato per avviare il container Lambda Ruby.

### Linux/MacOS

```
docker run --rm -it -v $PWD:/var/task -w /var/task awsruby32
```

#### Note

In MacOS, potresti visualizzare un avviso che ti informa che la piattaforma dell'immagine richiesta non corrisponde alla piattaforma host rilevata. Ignora questo avviso.

### Windows PowerShell

```
docker run --rm -it -v ${pwd}:/var/task -w /var/task awsruby32
```

All'avvio del container, dovresti vedere un prompt bash.

```
bash-4.2#
```

- Configura l'utilità di creazione di bundle per installare le gemme specificate nel Gemfile in una directory `vendor/bundle` locale e installare le tue dipendenze.

```
bash-4.2# bundle config set --local path 'vendor/bundle' && bundle install
```

- Crea il pacchetto di implementazione `.zip` con il codice della funzione e le relative dipendenze. In questo esempio, il file contenente il codice del gestore della funzione è denominato `lambda_function.rb`.

```
bash-4.2# zip -r my_deployment_package.zip lambda_function.rb vendor
```

- Esci dal container e torna alla directory del progetto locale.

```
bash-4.2# exit
```

Ora puoi usare il pacchetto di implementazione di file `.zip` per creare o aggiornare la tua funzione Lambda. Per informazioni, consultare [Creazione e aggiornamento delle funzioni Lambda di Ruby utilizzando file .zip](#).

## Creazione e aggiornamento delle funzioni Lambda di Ruby utilizzando file `.zip`

Dopo aver creato il pacchetto di implementazione `.zip`, puoi utilizzarlo per creare una nuova funzione Lambda o aggiornarne una esistente. Puoi distribuire il tuo pacchetto `.zip` utilizzando la console Lambda, l'API e Lambda. AWS Command Line Interface. Puoi anche creare e aggiornare le funzioni Lambda usando AWS Serverless Application Model (AWS SAM) e CloudFormation.

La dimensione massima per un pacchetto di implementazione `.zip` per Lambda è di 250 MB (dopo l'estrazione). Nota che questo limite si applica alla dimensione combinata di tutti i file caricati, inclusi eventuali livelli Lambda.

Il runtime Lambda necessita dell'autorizzazione per leggere i file nel pacchetto di distribuzione. Nella notazione ottale delle autorizzazioni di Linux, Lambda richiede 644 autorizzazioni per i file non eseguibili (`rw-r--r--`) e 755 autorizzazioni (`rw-r--r--`) per le directory e i file eseguibili.

In Linux e macOS, utilizza il comando `chmod` per modificare le autorizzazioni file su file e directory nel pacchetto di implementazione. Ad esempio, per assegnare a un file non eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 644 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta [Set, View, Change, or Remove Permissions on an Object](#) nella documentazione di Microsoft Windows.

#### Note

Se non concedi a Lambda le autorizzazioni necessarie per accedere alle directory del pacchetto di implementazione, Lambda imposta le autorizzazioni per tali directory su 755 (rwxr-xr-x).

## Creazione e aggiornamento delle funzioni con file .zip utilizzando la console

Per creare una nuova funzione, devi prima creare la funzione nella console, quindi devi caricare il tuo archivio .zip. Per aggiornare una funzione esistente, apri la pagina relativa alla funzione, quindi segui la stessa procedura per aggiungere il file .zip aggiornato.

Se il file .zip ha dimensioni inferiori a 50 MB, è possibile creare o aggiornare una funzione caricando il file direttamente dal computer locale. Per i file .zip di dimensioni superiori a 50 MB, prima è necessario caricare il pacchetto in un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il Console di gestione AWS, consulta [Guida introduttiva ad Amazon S3](#). Per caricare file utilizzando il AWS CLI, consulta [Sposta oggetti](#) nella Guida per l'AWS CLI utente.

#### Note

Non è possibile modificare il [tipo di pacchetto di implementazione](#) (.zip o immagine di container) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio di file .zip. È necessario creare una nuova funzione.

## Creazione di una nuova funzione (console)

1. Apri la [pagina Funzioni](#) della console Lambda e scegli Crea funzione.
2. Scegli Author from scratch (Crea da zero).
3. In Basic information (Informazioni di base) eseguire queste operazioni:
  - a. In Nome funzione, inserisci il nome della funzione.
  - b. Per Runtime, seleziona il runtime che desideri utilizzare.
  - c. (Facoltativo) Per Architettura, scegli l'architettura del set di istruzioni per la funzione. L'architettura predefinita è x86\_64. Assicurati che il pacchetto di implementazione per la tua funzione sia compatibile con l'architettura del set di istruzioni scelta.
4. (Opzionale) In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito. Puoi creare un nuovo ruolo di esecuzione o utilizzare un ruolo esistente.
5. Scegli Crea funzione. Lambda crea una funzione di base "Hello world" utilizzando il runtime scelto.

## Caricamento di un archivio .zip dal computer locale (console)

1. Nella [pagina Funzioni](#) della console Lambda, scegli la funzione per cui vuoi caricare il file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli File .zip.
5. Per caricare il file .zip, procedi come segue:
  - a. Seleziona Carica, quindi seleziona il tuo file .zip nel selettore di file.
  - b. Seleziona Apri.
  - c. Scegli Save (Salva).

## Caricamento di un archivio .zip da un bucket Amazon S3 (console)

1. Nella [pagina Funzioni](#) della console Lambda, scegli la funzione per cui vuoi caricare un nuovo file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.

4. Scegli Posizione Amazon S3.
5. Incolla l'URL del link Amazon S3 del tuo file .zip e scegli Salva.

### Aggiornamento delle funzioni dei file .zip tramite l'editor di codice della console

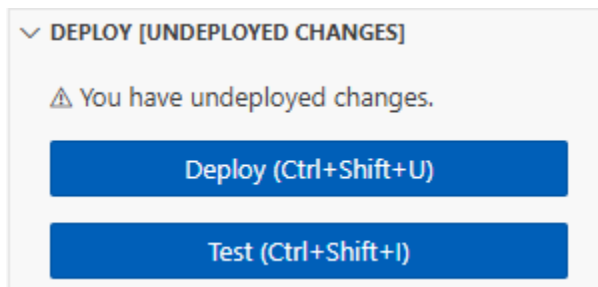
Per alcune funzioni con pacchetti di implementazione .zip, puoi utilizzare l'editor di codice integrato nella console Lambda per aggiornare direttamente il codice della funzione. Per utilizzare questa funzione, la funzione deve soddisfare i seguenti criteri:

- La funzione deve utilizzare uno dei runtime del linguaggio interpretato (Python o Node.js Ruby)
- Il pacchetto di implementazione della funzione deve avere dimensioni inferiori a 50 MB (non compresso).

Il codice della funzione per le funzioni con pacchetti di implementazione di immagini di container non può essere modificato direttamente nella console.

### Aggiornamento del codice della funzione utilizzando l'editor di codice della console

1. Apri la [pagina Funzioni](#) della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, seleziona il tuo file di codice sorgente e modificalo nell'editor di codice integrato.
4. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



### Creazione e aggiornamento di funzioni con file .zip utilizzando AWS CLI

È possibile utilizzare la [AWS CLI](#) per creare una nuova funzione o aggiornare una funzione esistente mediante un file .zip. Utilizza i comandi [create-function](#) e [update-function-code](#) per implementare il tuo pacchetto .zip. Se il file .zip ha dimensioni inferiori a 50 MB, è possibile caricare il pacchetto .zip da una posizione di file nella macchina di compilazione locale. Per i file di dimensioni maggiori, è

necessario caricare il pacchetto .zip da un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il AWS CLI, consulta [Move objects](#) nella Guida per l'AWS CLI utente.

#### Note

Se carichi il tuo file .zip da un bucket Amazon S3 utilizzando il AWS CLI, il bucket deve trovarsi nella stessa posizione della tua funzione. Regione AWS

Per creare una nuova funzione utilizzando un file .zip con AWS CLI, devi specificare quanto segue:

- Il nome della funzione (`--function-name`)
- Il runtime della tua funzione (`--runtime`)
- Il nome della risorsa Amazon (ARN) del [ruolo di esecuzione](#) della funzione (`--role`)
- Il nome del metodo del gestore nel codice della funzione (`--handler`)

È inoltre necessario specificare la posizione del file .zip. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda create-function --function-name myFunction \  
--runtime ruby3.2 --handler lambda_function.lambda_handler \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza l'opzione `--code` illustrata nel seguente comando di esempio. È necessario utilizzare il parametro `S3ObjectVersion` solo per gli oggetti con controllo delle versioni.

```
aws lambda create-function --function-name myFunction \  
--runtime ruby3.2 --handler lambda_function.lambda_handler \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--code S3Bucket=amzn-s3-demo-  
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Per aggiornare una funzione esistente mediante la CLI, specifica il nome della funzione utilizzando il parametro `--function-name`. È inoltre necessario specificare la posizione del file .zip che desideri



utilizzare per aggiornare il codice della funzione. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda update-function-code --function-name myFunction \  
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza le opzioni `--s3-bucket` e `--s3-key` come illustrato nel seguente comando di esempio. È necessario utilizzare il parametro `--s3-object-version` solo per gli oggetti con controllo delle versioni.

```
aws lambda update-function-code --function-name myFunction \  
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject  
Version
```

### Creazione e aggiornamento delle funzioni con file .zip utilizzando l'API Lambda

Per creare e aggiornare le funzioni mediante un archivio di file .zip, utilizza le seguenti operazioni API:

- [CreateFunction](#)
- [UpdateFunctionCode](#)

### Creazione e aggiornamento di funzioni con file .zip utilizzando AWS SAM

AWS Serverless Application Model (AWS SAM) è un toolkit che aiuta a semplificare il processo di creazione ed esecuzione di applicazioni serverless su AWS. Definisci le risorse per la tua applicazione in un modello YAML o JSON e usi l'interfaccia a riga di AWS SAM comando (AWS SAM CLI) per creare, impacchettare e distribuire le tue applicazioni. Quando crei una funzione Lambda da un AWS SAM modello, crea AWS SAM automaticamente un pacchetto di distribuzione .zip o un'immagine del contenitore con il codice della funzione e le dipendenze specificate. Per saperne di più sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda, consulta la [Guida introduttiva nella Developer Guide. AWS SAM](#) AWS Serverless Application Model

Puoi anche usare AWS SAM per creare una funzione Lambda utilizzando un archivio di file .zip esistente. Per creare una funzione Lambda utilizzando AWS SAM, puoi salvare il tuo file .zip in un bucket Amazon S3 o in una cartella locale sulla macchina di compilazione. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il AWS CLI, consulta [Move objects nella Guida per l'utente.](#) AWS CLI

Nel AWS SAM modello, la `AWS::Serverless::Function` risorsa specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `CodeUri`- impostato sull'URI Amazon S3, sul percorso della cartella locale o sull'oggetto del codice funzione [FunctionCode](#)
- `Runtime`: imposta il runtime prescelto

Con AWS SAM, se il tuo file .zip è più grande di 50 MB, non devi prima caricarlo su un bucket Amazon S3. AWS SAM puoi caricare pacchetti .zip fino alla dimensione massima consentita di 250 MB (decompressi) da una posizione sul computer di compilazione locale.

Per saperne di più sulla distribuzione delle funzioni utilizzando il file .zip in AWS SAM, consulta [AWS::Serverless: :Function nella Developer Guide](#). AWS SAM

Creazione e aggiornamento di funzioni con file .zip utilizzando CloudFormation

È possibile utilizzare CloudFormation per creare una funzione Lambda utilizzando un archivio di file .zip. Per creare una funzione Lambda da un file .zip, devi prima caricare il file su un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando AWS CLI, consulta [Move objects](#) nella Guida per l'utente. AWS CLI

Nel CloudFormation modello, la `AWS::Lambda::Function` risorsa specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `Code`: inserisci il nome del bucket Amazon S3 e il nome del file .zip nei campi `S3Bucket` e `S3Key`
- `Runtime`: imposta il runtime prescelto

Il file .zip CloudFormation generato non può superare i 4 MB. Per saperne di più sulla distribuzione delle funzioni utilizzando il file .zip in CloudFormation, vedi [AWS::Lambda: :Function](#) nella Guida per l'utente. CloudFormation

## Distribuisci funzioni Lambda per Ruby con immagini di container

Esistono tre modi per creare un'immagine di container per una funzione Lambda in Ruby:

- [Usare un'immagine di AWS base per Ruby](#)

[Le immagini di base AWS](#) sono precaricate con un runtime in linguaggio, un client di interfaccia di runtime per gestire l'interazione tra Lambda e il codice della funzione e un emulatore di interfaccia di runtime per i test locali.

- [Usando un AWS OS-only immagine di base](#)

[AWS OS-only le immagini di base](#) contengono una distribuzione Amazon Linux e l'[emulatore di interfaccia di runtime](#). Queste immagini vengono comunemente utilizzate per creare immagini contenitore per linguaggi compilati, come [Go](#) e [Rust](#), e per un linguaggio o una versione linguistica per cui Lambda non fornisce un'immagine di base, ad Node.js esempio 19. Puoi anche usare immagini di OS-only base per implementare un [runtime personalizzato](#). Per rendere l'immagine compatibile con Lambda, devi includere il [client di interfaccia di runtime per Ruby](#) nell'immagine.

- [Utilizzo di un'immagine non di AWS base](#)

È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere il [client di interfaccia di runtime per Ruby](#) nell'immagine.

### Tip

Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta [Utilizzo di compilazioni a più fasi](#) nella documentazione Docker. Per creare immagini di container efficienti, segui le [best practice per scrivere file Docker](#).

Questa pagina spiega come creare, testare e implementare le immagini di container per Lambda.

### Argomenti

- [AWS immagini di base per Ruby](#)
- [Usare un'immagine di AWS base per Ruby](#)
- [Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime](#)

## AWS immagini di base per Ruby

AWS fornisce le seguenti immagini di base per Ruby:

| Tag | Runtime    | Sistema operativo | Dockerfile                                                           | Raggiunta obsolescenza |
|-----|------------|-------------------|----------------------------------------------------------------------|------------------------|
| 4.0 | Ruby 4.0   | Amazon Linux 2023 | <a href="#">Dockerfile per Ruby 4.0 su GitHub</a>                    | 31 marzo 2029          |
| 3.4 | Rubino 3.4 | Amazon Linux 2023 | <a href="#">Dockerfile per Ruby 3.4 e versioni successive GitHub</a> | 31 marzo 2028          |
| 3.3 | Ruby 3.3   | Amazon Linux 2023 | <a href="#">Dockerfile per Ruby 3.3 su GitHub</a>                    | 31 marzo 2027          |

[Archivio Amazon ECR: `gallery.ecr.aws/lambda/rubino`](#)

Usare un'immagine di AWS base per Ruby

### Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [AWS CLI versione 2](#)
- [Docker](#) (versione minima 25.0.0)
- Il [plugin buildx](#) di Docker.
- Ruby

Creazione di un'immagine da un'immagine di base

Creazione di un'immagine di container per Ruby

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
cd example
```

2. Crea un nuovo file denominato `Gemfile`. Qui è possibile elencare i RubyGems pacchetti richiesti dall'applicazione. AWS SDK per Ruby È disponibile da RubyGems. Dovresti scegliere AWS service gem specifici da installare. Ad esempio, per usare la [gemma Ruby per Lambda](#), il tuo Gemfile dovrebbe avere questo aspetto:

```
source 'https://rubygems.org'

gem 'aws-sdk-lambda'
```

In alternativa, la gemma [aws-sdk](#) contiene tutte le gem di servizio disponibili. AWS Questa gemma è molto grande. Ti consigliamo di utilizzarlo solo se dipendi da molti servizi. AWS

3. Installa le dipendenze specificate nel Gemfile utilizzando l'[installazione in bundle](#).

```
bundle install
```

4. Crea un nuovo file denominato `lambda_function.rb`. A fini di test, puoi utilizzare il codice della funzione di esempio seguente o sostituirlo con il tuo codice personalizzato.

#### Example Funzione Ruby

```
module LambdaFunction
  class Handler
    def self.process(event:, context:)
      "Hello from Lambda!"
    end
  end
end
```

5. Crea un nuovo Dockerfile. Il Dockerfile di esempio seguente utilizza l'[immagine di base AWS](#). Il Dockerfile utilizza la seguente configurazione:
  - Imposta la proprietà FROM sull'URI dell'immagine di base.
  - Utilizzate il comando COPY per copiare il codice della funzione e le dipendenze di `{LAMBDA_TASK_ROOT}` runtime in una [variabile di Lambda-defined ambiente](#).
  - Imposta l'argomento CMD specificando il gestore della funzione Lambda.

Nota che l'esempio Dockerfile non include un'[istruzione USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con

autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

### Example Dockerfile

```
FROM public.ecr.aws/lambda/ruby:3.4

# Copy Gemfile and Gemfile.lock
COPY Gemfile Gemfile.lock ${LAMBDA_TASK_ROOT}/

# Install Bundler and the specified gems
RUN gem install bundler:2.4.20 && \
    bundle config set --local path 'vendor/bundle' && \
    bundle install

# Copy function code
COPY lambda_function.rb ${LAMBDA_TASK_ROOT}/

# Set the CMD to your handler (could also be done as a parameter override outside
# of the Dockerfile)
CMD [ "lambda_function.LambdaFunction::Handler.process" ]
```

6. Crea l'immagine Docker con il comando [docker build](#). L'esempio seguente assegna un nome all'immagine docker-image e le assegna il [tag](#) test. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'opzione `--provenance=false`.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test .
```

#### Note

Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Per creare una funzione Lambda utilizzando l'architettura del set di istruzioni ARM64, accertati di modificare il comando per utilizzare l'opzione `--platform linux/arm64`.

## (Facoltativo) Test dell'immagine in locale

1. Avvia l'immagine Docker con il comando `docker run`. In questo esempio, `docker-image` è il nome dell'immagine e `test` è il tag.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

### Note

Per creare l'immagine Docker per l'architettura del set di istruzioni ARM64, assicurati di utilizzare l'opzione `--platform linux/arm64` anziché `--platform linux/amd64`.

2. Da una nuova finestra di terminale, invia un evento all'endpoint locale.

### Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

### PowerShell

In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType "application/json"
```

3. Ottieni l'ID del container.

```
docker ps
```

4. Utilizza il comando [docker kill](#) per arrestare il container. In questo comando, sostituisci 3766c4ab331c con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

## Implementazione dell'immagine

### Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Per autenticare la CLI Docker nel registro Amazon ECR, esegui il comando [get-login-password](#).

- Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
- Sostituiscilo 111122223333 con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```



 Note

Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando [docker tag](#) per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:
  - `docker-image:test` è il nome e [tag](#) dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
  - Sostituisci l'<ECRrepositoryUri> con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando [docker push](#) per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Crea un ruolo di esecuzione](#) per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per ImageUri, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta [Autorizzazioni multiaccount Amazon ECR](#).

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nel repository Amazon ECR e quindi utilizzare il comando [update-function-code](#) per implementare l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per implementare la nuova immagine nella stessa funzione Lambda, devi utilizzare il comando [update-function-code](#) anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \  
  --function-name hello-world \  
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --publish
```

Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

Se si utilizza un'[immagine di OS-only base](#) o un'immagine di base alternativa, è necessario includere il client dell'interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime estende l'[API Runtime](#), che gestisce l'interazione tra Lambda e il codice della funzione.

Installa il [client dell'interfaccia runtime Lambda per Ruby](#) usando il RubyGems.org gestore di pacchetti:

```
gem install aws_lambda_ri
```

Puoi anche scaricare il client dell'[interfaccia di runtime Ruby](#) da GitHub

L'esempio seguente mostra come creare un'immagine contenitore per Ruby utilizzando un'immagine non di base.AWS Il Dockerfile di esempio utilizza un'immagine di base Ruby ufficiale. Il Dockerfile include il client di interfaccia di runtime.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [AWS CLI versione 2](#)
- [Docker](#) (versione minima 25.0.0)

- Il [plugin buildx](#) di Docker.
- Ruby

Creazione di un'immagine da un'immagine di base alternativa

Creazione di un'immagine di container per Ruby utilizzando un'immagine di base alternativa

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
cd example
```

2. Crea un nuovo file denominato `Gemfile`. Qui è possibile elencare i RubyGems pacchetti richiesti dall'applicazione. AWS SDK per Ruby È disponibile da RubyGems. Dovresti scegliere AWS service gem specifici da installare. Ad esempio, per usare la [gemma Ruby per Lambda](#), il tuo Gemfile dovrebbe avere questo aspetto:

```
source 'https://rubygems.org'

gem 'aws-sdk-lambda'
```

In alternativa, la gemma [aws-sdk](#) contiene tutte le gem di servizio disponibili. AWS Questa gemma è molto grande. Ti consigliamo di utilizzarlo solo se dipendi da molti servizi. AWS

3. Installa le dipendenze specificate nel Gemfile utilizzando l'[installazione in bundle](#).

```
bundle install
```

4. Crea un nuovo file denominato `lambda_function.rb`. A fini di test, puoi utilizzare il codice della funzione di esempio seguente o sostituirlo con il tuo codice personalizzato.

Example Funzione Ruby

```
module LambdaFunction
  class Handler
    def self.process(event:, context:)
      "Hello from Lambda!"
    end
  end
end
```

5. Crea un nuovo Dockerfile. Il seguente Dockerfile utilizza un'immagine di base Ruby anziché un'[immagine di base AWS](#). Il Dockerfile include il [client di interfaccia di runtime per Ruby](#), che rende l'immagine compatibile con Lambda. In alternativa, puoi aggiungere il client di interfaccia di runtime al Gemfile dell'applicazione.

- Imposta la proprietà FROM sull'immagine di base Ruby.
- Crea una directory per il codice della funzione e una variabile di ambiente che punti a quella directory. In questo esempio, la directory è /var/task, che rispecchia l'ambiente di esecuzione Lambda. Tuttavia, puoi scegliere qualsiasi directory per il codice della funzione perché il Dockerfile non utilizza un'immagine di AWS base.
- Imposta l'ENTRYPOINT sul modulo su cui desideri che il container Docker venga eseguito all'avvio. In questo caso, il modulo è il client di interfaccia di runtime.
- Imposta l'argomento CMD specificando il gestore della funzione Lambda.

Nota che l'esempio Dockerfile non include un'[istruzione USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente root quando non viene fornita alcuna istruzione USER.

### Example Dockerfile

```
FROM ruby:2.7

# Install the runtime interface client for Ruby
RUN gem install aws_lambda_ric

# Add the runtime interface client to the PATH
ENV PATH="/usr/local/bundle/bin:${PATH}"

# Create a directory for the Lambda function
ENV LAMBDA_TASK_ROOT=/var/task
RUN mkdir -p ${LAMBDA_TASK_ROOT}
WORKDIR ${LAMBDA_TASK_ROOT}

# Copy Gemfile and Gemfile.lock
COPY Gemfile Gemfile.lock ${LAMBDA_TASK_ROOT}/

# Install Bundler and the specified gems
RUN gem install bundler:2.4.20 && \
```

```
bundle config set --local path 'vendor/bundle' && \
bundle install

# Copy function code
COPY lambda_function.rb ${LAMBDA_TASK_ROOT}/

# Set runtime interface client as default command for the container runtime
ENTRYPOINT [ "aws_lambda_ric" ]

# Set the CMD to your handler (could also be done as a parameter override outside
of the Dockerfile)
CMD [ "lambda_function.LambdaFunction::Handler.process" ]
```

6. Crea l'immagine Docker con il comando [docker build](#). L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il [tag](#) `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'opzione `--provenance=false`.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
.
```

#### Note

Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Per creare una funzione Lambda utilizzando l'architettura del set di istruzioni ARM64, accertati di modificare il comando per utilizzare l'opzione `--platform linux/arm64`.

### (Facoltativo) Test dell'immagine in locale

Usa il [simulatore dell'interfaccia di runtime](#) per testare l'immagine in locale. Puoi [creare l'emulatore nella tua immagine](#) o seguire la procedura riportata e installarlo sul tuo computer locale.

### Installazione ed esecuzione dell'emulatore di interfaccia di runtime sul computer locale

1. Dalla directory del progetto, esegui il comando seguente per scaricare l'emulatore di interfaccia di runtime (architettura x86-64) GitHub e installarlo sul computer locale.

## Linux/macOS

```
mkdir -p ~/.aws-lambda-rie && \  
  curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-  
runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \  
  chmod +x ~/.aws-lambda-rie/aws-lambda-rie
```

Per installare l'emulatore arm64, sostituisci l'URL del GitHub repository nel comando precedente con il seguente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/  
download/aws-lambda-rie-arm64
```

## PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"  
if (-not (Test-Path $dirPath)) {  
    New-Item -Path $dirPath -ItemType Directory  
}  
  
$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/  
releases/latest/download/aws-lambda-rie"  
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"  
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Per installare l'emulatore arm64, sostituisci `$downloadLink` con quanto segue:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/  
download/aws-lambda-rie-arm64
```

2. Avvia l'immagine Docker con il comando `docker run`. Tenere presente quanto segue:

- `docker-image` è il nome dell'immagine e `test` è il tag.
- `aws_lambda_rie lambda_function.LambdaFunction::Handler.process` è l'ENTRYPOINT seguito dal CMD del Dockerfile.

## Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
9000:8080 \
  --entrypoint /aws-lambda/aws-lambda-rie \
  docker-image:test \
  aws_lambda_rie lambda_function.LambdaFunction::Handler.process
```

## PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
9000:8080 `
  --entrypoint /aws-lambda/aws-lambda-rie `
  docker-image:test `
  aws_lambda_rie lambda_function.LambdaFunction::Handler.process
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

### Note

Per creare l'immagine Docker per l'architettura del set di istruzioni ARM64, assicurati di utilizzare l'opzione `--platform linux/arm64` anziché `--platform linux/amd64`.

3. Pubblica un evento nell'endpoint locale.

## Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:



```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

## PowerShell

In PowerShell, esegui il seguente comando: Invoke-WebRequest

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{} ' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType  
"application/json"
```

### 4. Ottieni l'ID del container.

```
docker ps
```

### 5. Utilizza il comando [docker kill](#) per arrestare il container. In questo comando, sostituisci 3766c4ab331c con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

## Implementazione dell'immagine

### Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

#### 1. Per autenticare la CLI Docker nel registro Amazon ECR, esegui il comando [get-login-password](#).

- Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
- Sostituiscilo 111122223333 con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

#### Note

Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copia il repositoryUri dall'output del passaggio precedente.
4. Esegui il comando [docker tag](#) per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:

- `docker-image:test` è il nome e [tag](#) dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
- Sostituisci l'<ECRrepositoryUri> con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando [docker push](#) per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Crea un ruolo di esecuzione](#) per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta [Autorizzazioni multiaccount Amazon ECR](#).

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nel repository Amazon ECR e quindi utilizzare il comando [update-function-code](#) per implementare l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per implementare la nuova immagine nella stessa funzione Lambda, devi utilizzare il comando [update-function-code](#) anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \  
  --function-name hello-world \  
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --publish
```

## Utilizzo dei livelli per le funzioni Lambda di Ruby

Usa i [livelli Lambda](#) per impacchettare codice e dipendenze che desideri riutilizzare in più funzioni. I livelli di solito contengono dipendenze dalla libreria, un [runtime personalizzato](#) o file di configurazione. La creazione di un livello prevede tre passaggi generali:

1. Crea un pacchetto per il contenuto del livello. Ciò significa creare un archivio di file con estensione .zip che contiene le dipendenze che desideri utilizzare nelle funzioni.
2. Crea il livello in Lambda.
3. Aggiungi il livello alle tue funzioni.

### Argomenti

- [Crea un pacchetto per il contenuto del livello](#)
- [Crea il layer in Lambda](#)
- [Usare le gemme dei livelli in una funzione](#)
- [Aggiungi il livello alla tua funzione](#)
- [App di esempio](#)

### Crea un pacchetto per il contenuto del livello

Per creare un layer, raggruppa i pacchetti in un archivio di file.zip che soddisfi i seguenti requisiti:

- Crea il layer utilizzando la stessa versione di Ruby che intendi utilizzare per la funzione Lambda. Ad esempio, se crei il vostro layer per Ruby 4.0, usate il runtime Ruby 4.0 per la vostra funzione.
- Il file.zip del livello deve utilizzare una di queste strutture di directory:
  - `ruby/gems/x.x.x(x.x.x dov'è la tua versione di Ruby, ad esempio) 3.4.0`
  - `ruby/lib`

Per ulteriori informazioni, consulta [Percorsi dei livelli per ciascun runtime Lambda](#).

- I pacchetti del livello devono essere compatibili con Linux. Le funzioni Lambda vengono eseguite su Amazon Linux.

Puoi creare livelli che contengono gemme Ruby di terze parti o moduli e classi Ruby personalizzati. Molte gemme Ruby popolari contengono estensioni native (codice C) che devono essere compilate per l'ambiente Lambda Linux.

## Gemme Ruby pure

Le gemme Pure Ruby contengono solo codice Ruby e non richiedono la compilazione. Queste gemme sono più semplici da impacchettare e funzionano su piattaforme diverse.

Per creare un livello usando gemme Ruby pure

1. Crea uno `Gemfile` per specificare le gemme Ruby pure che desideri includere nel tuo livello:

### Example Gemfile

```
source 'https://rubygems.org'

gem 'tzinfo'
```

2. Installa le gemme nella `vendor/bundle` directory usando Bundler:

```
bundle config set --local path vendor/bundle
bundle install
```

3. Copia le gemme installate nella struttura di directory richiesta da `Lambdaruby/gems/3.4.0`:

```
mkdir -p ruby/gems/3.4.0
cp -r vendor/bundle/ruby/3.4.0*/ * ruby/gems/3.4.0/
```

4. Comprimi il contenuto del livello:

### Linux/macOS

```
zip -r layer.zip ruby/
```

### PowerShell

```
Compress-Archive -Path .\ruby -DestinationPath .\layer.zip
```

La struttura delle cartelle del file.zip dovrebbe essere simile alla seguente:

```
ruby/
### gems/
    ### 3.4.0/
        ### gems/
```

```
#   ## concurrent-ruby-1.3.5/  
#   ## tzinfo-2.0.6/  
## specifications/  
## cache/  
## build_info/  
## (other bundler directories)
```

#### Note

È necessario richiedere ogni gemma singolarmente nel codice della funzione. Non puoi usare `bundler/setup` o `Bundler.require`. Per ulteriori informazioni, consulta [Usare le gemme dei livelli in una funzione](#).

## Gemme con estensioni native

Molte gemme Ruby popolari contengono estensioni native (codice C) che devono essere compilate per la piattaforma di destinazione. [Le gemme popolari con estensioni native includono nokogiri, pg, mysql2, sqlite3 e ffi](#). Queste gemme devono essere create in un ambiente Linux compatibile con il runtime Lambda.

Per creare un livello utilizzando gemme con estensioni native

1. Creare un Gemfile.

### Example Gemfile

```
source 'https://rubygems.org'  
  
gem 'nokogiri'  
gem 'httparty'
```

2. Usa Docker per creare le gemme in un ambiente Linux compatibile con Lambda. Specificate un'[immagine di AWS base](#) nel vostro Dockerfile:

### Example Dockerfile per Ruby 4.0

```
FROM public.ecr.aws/lambda/ruby:3.4  
  
# Copy Gemfile  
COPY Gemfile ./
```

```
# Install system dependencies for native extensions
RUN dnf update -y && \
    dnf install -y gcc gcc-c++ make

# Configure bundler and install gems
RUN bundle config set --local path vendor/bundle && \
    bundle install

# Create the layer structure
RUN mkdir -p ruby/gems/3.4.0 && \
    cp -r vendor/bundle/ruby/3.4.0*/ ruby/gems/3.4.0/

# Create the layer zip file
RUN zip -r layer.zip ruby/
```

### 3. Costruisci l'immagine ed estrai il livello:

```
docker build -t ruby-layer-builder .
docker run --rm -v $(pwd):/output --entrypoint cp ruby-layer-builder layer.zip /
output/
```

Questo crea le gemme nell'ambiente Linux corretto e copia il `layer.zip` file nella directory locale. La struttura delle cartelle del file.zip dovrebbe assomigliare a questa:

```
ruby/
### gems/
  ### 3.4.0/
    ### gems/
      # ### bigdecimal-3.2.2/
      # ### csv-3.3.5/
      # ### httparty-0.23.1/
      # ### mini_mime-1.1.5/
      # ### multi_xml-0.7.2/
      # ### nokogiri-1.18.8-x86_64-linux-gnu/
      # ### racc-1.8.1/
    ### build_info/
    ### cache/
    ### specifications/
    ### (other bundler directories)
```



 Note

È necessario richiedere ogni gemma singolarmente nel codice della funzione. Non puoi usare `bundler/setup` o `Bundler.require`. Per ulteriori informazioni, consulta [Usare le gemme dei livelli in una funzione](#).

## Moduli Ruby personalizzati

Per creare un livello utilizzando il proprio codice

1. Crea la struttura di directory richiesta per il tuo livello:

```
mkdir -p ruby/lib
```

2. Crea i tuoi moduli Ruby nella `ruby/lib` directory. Il seguente modulo di esempio convalida gli ordini confermando che contengono le informazioni richieste.

Example `ruby/lib/order_validator.rb`

```
require 'json'

module OrderValidator
  class ValidationError < StandardError; end

  def self.validate_order(order_data)
    # Validates an order and returns formatted data
    required_fields = %w[product_id quantity]

    # Check required fields
    missing_fields = required_fields.reject { |field| order_data.key?(field) }
    unless missing_fields.empty?
      raise ValidationError, "Missing required fields: #{missing_fields.join(', ')}"
    end

    # Validate quantity
    quantity = order_data['quantity']
    unless quantity.is_a?(Integer) && quantity > 0
      raise ValidationError, 'Quantity must be a positive integer'
    end
  end
end
```

```
# Format and return the validated data
{
  'product_id' => order_data['product_id'].to_s,
  'quantity' => quantity,
  'shipping_priority' => order_data.fetch('priority', 'standard')
}
end

def self.format_response(status_code, body)
  # Formats the API response
  {
    statusCode: status_code,
    body: JSON.generate(body)
  }
end
end
```

### 3. Comprimi il contenuto del livello:

#### Linux/macOS

```
zip -r layer.zip ruby/
```

#### PowerShell

```
Compress-Archive -Path .\ruby -DestinationPath .\layer.zip
```

La struttura delle cartelle del file.zip dovrebbe essere simile alla seguente:

```
ruby/
### lib/
    ### order_validator.rb
```

4. Nella tua funzione, richiedi e usa i moduli. È necessario richiedere ogni gemma singolarmente nel codice della funzione. Non puoi usare `bundler/setup` o `Bundler.require`. Per ulteriori informazioni, consulta [Usare le gemme dei livelli in una funzione](#). Esempio:

```
require 'json'
require 'order_validator'
```

```
def lambda_handler(event:, context:)
  begin
    # Parse the order data from the event body
    order_data = JSON.parse(event['body'] || '{}')

    # Validate and format the order
    validated_order = OrderValidator.validate_order(order_data)

    OrderValidator.format_response(200, {
      message: 'Order validated successfully',
      order: validated_order
    })
  rescue OrderValidator::ValidationError => e
    OrderValidator.format_response(400, {
      error: e.message
    })
  rescue => e
    OrderValidator.format_response(500, {
      error: 'Internal server error'
    })
  end
end
```

È possibile utilizzare il seguente [evento di test](#) per richiamare la funzione:

```
{
  "body": "{\"product_id\": \"ABC123\", \"quantity\": 2, \"priority\": \"express\"}"
}
```

Risposta prevista:

```
{
  "statusCode": 200,
  "body": "{\"message\": \"Order validated successfully\", \"order\": {\"product_id\": \"ABC123\", \"quantity\": 2, \"shipping_priority\": \"express\"}}"
```

## Crea il layer in Lambda

È possibile pubblicare il layer utilizzando la console AWS CLI o la console Lambda.

## AWS CLI

Esegui il AWS CLI comando [publish-layer-version](#) per creare il livello Lambda:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip
--compatible-runtimes ruby4.0
```

Il parametro [compatible runtimes è facoltativo](#). Quando specificato, Lambda utilizza questo parametro per filtrare i livelli nella console Lambda.

## Console

### Creazione di un livello (console)

1. Apri la [pagina Layers](#) (Livelli) nella console Lambda.
2. Scegli Create layer (Crea livello).
3. Scegli Carica un file.zip, quindi carica l'archivio.zip che hai creato in precedenza.
4. (Facoltativo) Per i runtime compatibili, scegli il runtime di Ruby che corrisponde alla versione di Ruby che hai usato per creare il tuo layer.
5. Scegli Create (Crea).

### Usare le gemme dei livelli in una funzione

Nel codice della funzione, è necessario richiedere esplicitamente ogni gemma che si desidera utilizzare. I comandi Bundler come `bundler/setup` e `non Bundler.require` sono supportati. Ecco come usare correttamente le gemme di un livello in una funzione Lambda:

```
# Correct: Use explicit requires for each gem
require 'nokogiri'
require 'httparty'

def lambda_handler(event:, context:)
  # Use the gems directly
  doc = Nokogiri::HTML(event['html'])
  response = HTTParty.get(event['url'])
  # ... rest of your function
end
```

```
# Incorrect: These Bundler commands will not work
# require 'bundler/setup'
# Bundler.require
```

Aggiungi il livello alla tua funzione

## AWS CLI

Per collegare il layer alla tua funzione, esegui il comando [update-function-configuration](#) AWS CLI . Per il `--layers` parametro, utilizzate il livello ARN. L'ARN deve specificare la versione (ad esempio, `arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`). Per ulteriori informazioni, consulta [Livelli e versioni di livelli](#).

```
aws lambda update-function-configuration --function-name my-function --cli-binary-format raw-in-base64-out --layers "arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1"
```

L'`cli-binary-format` opzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

## Console

Aggiunta di un livello a una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione.
3. Scorri verso il basso fino alla sezione Livelli, quindi scegli Aggiungi un livello.
4. In Scegli un livello, seleziona Livelli personalizzati, quindi scegli il tuo livello.

### Note

Se non hai aggiunto un [runtime compatibile](#) quando hai creato il layer, il tuo layer non verrà elencato qui. È possibile specificare invece il livello ARN.

5. Scegliere Aggiungi.

## App di esempio

Per altri esempi di utilizzo dei layer Lambda, consultate l'applicazione di esempio [layer-ruby](#) nel repository Developer Guide. AWS Lambda GitHub [Questa applicazione include un livello che contiene la libreria tzinfo](#). Dopo aver creato il layer, è possibile implementare e richiamare la funzione corrispondente per confermare che il layer funzioni come previsto.

## Utilizzo dell'oggetto del contesto Lambda per recuperare le informazioni sulla funzione Ruby

Quando Lambda esegue la funzione, passa un oggetto Context al [gestore](#). Questo oggetto fornisce i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

### Metodi del contesto

- `get_remaining_time_in_millis`: restituisce il numero di millisecondi rimasti prima del timeout dell'esecuzione.

### Proprietà del contesto

- `function_name`: il nome della funzione Lambda.
- `function_version`: la [versione](#) della funzione.
- `invoked_function_arn`: l'Amazon Resource Name (ARN) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `memory_limit_in_mb`: la quantità di memoria allocata per la funzione.
- `aws_request_id`: l'identificatore della richiesta di invocazione.
- `log_group_name`: il gruppo di log per la funzione.
- `log_stream_name`: il flusso di log per l'istanza della funzione.
- `deadline_ms`: la data del timeout dell'esecuzione in millisecondi Unix.
- `identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
- `client_context`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.

## Registrare e monitorare le funzioni Lambda con Ruby

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon. CloudWatch La tua funzione Lambda include un gruppo di CloudWatch log Logs e un flusso di log per ogni istanza della tua funzione. L'ambiente del runtime Lambda invia i dettagli su ogni richiamo al flusso di log e inoltra i log e l'output del codice della funzione. Per ulteriori informazioni, consulta [Invio dei log delle funzioni Lambda ai CloudWatch log](#).

Questa pagina descrive come produrre un output di log dal codice della funzione Lambda e accedere ai log utilizzando la AWS Command Line Interface console Lambda o la console. CloudWatch

### Sections

- [Creazione di una funzione che restituisce i registri](#)
- [Utilizzo dei controlli di registrazione avanzati Lambda con Ruby](#)
- [Visualizzazione dei log nella console Lambda](#)
- [Visualizzazione dei log nella console CloudWatch](#)
- [Visualizzazione dei log utilizzando AWS Command Line Interface \(AWS CLI\)](#)
- [Eliminazione dei log](#)
- [Utilizzo della libreria dei logger Ruby](#)

### Creazione di una funzione che restituisce i registri

Per i log di output del codice della funzione, puoi usare le istruzioni `puts` o qualsiasi libreria di registrazione che scriva in `stdout` o `stderr`. L'esempio seguente registra i valori delle variabili di ambiente e l'oggetto evento. Per informazioni sull'oggetto di contesto disponibile per il tuo gestore, consulta [the section called "Contesto"](#)

### Example `lambda_function.rb`

```
# lambda_function.rb

def handler(event:, context:)
  puts "## ENVIRONMENT VARIABLES"
  puts ENV.to_a
  puts "## EVENT"
  puts event.to_a
end
```



## Example Formato dei log

```
START RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95 Version: $LATEST
## ENVIRONMENT VARIABLES
environ({'AWS_LAMBDA_LOG_GROUP_NAME': '/aws/lambda/my-function',
  'AWS_LAMBDA_LOG_STREAM_NAME': '2020/01/31/[$LATEST]3893xmpl7fac4485b47bb75b671a283c',
  'AWS_LAMBDA_FUNCTION_NAME': 'my-function', ...})
## EVENT
{'key': 'value'}
END RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95
REPORT RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95  Duration: 15.74 ms  Billed
  Duration: 147 ms Memory Size: 128 MB Max Memory Used: 56 MB  Init Duration: 130.49 ms
XRAY TraceId: 1-5e34a614-10bdxmplf1fb44f07bc535a1  SegmentId: 07f5xmpl2d1f6f85
  Sampled: true
```

Il runtime di Ruby registra START, END e REPORT per ogni chiamata. La riga del report fornisce i seguenti dettagli.

### Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per l'invocazione.
- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.
- Quantità max utilizzata – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- Durata Init – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- XRAY TraceId: per le richieste tracciate, l'ID di traccia. [AWS X-Ray](#)
- SegmentId— Per le richieste tracciate, l'ID del X-Ray segmento.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Per ottenere log più dettagliati, utilizza la [the section called “Utilizzo della libreria dei logger Ruby”](#).

## Utilizzo dei controlli di registrazione avanzati Lambda con Ruby

Per darti un maggiore controllo su come i log delle tue funzioni vengono acquisiti, elaborati e consumati, Lambda offre controlli di registrazione avanzati con Ruby. Per i runtime di Ruby 4.0 e versioni successive, puoi configurare le seguenti opzioni di registrazione:

- **Formato di registro:** seleziona tra testo normale e formato JSON strutturato per i log della tua funzione
- **Livello di registro:** per i log in formato JSON, scegli il livello di dettaglio dei log che Lambda invia ad Amazon CloudWatch, ad esempio ERROR, DEBUG o INFO
- **Gruppo di log:** scegli il gruppo di CloudWatch log a cui la tua funzione invia i log

Per ulteriori informazioni su queste opzioni di registrazione e istruzioni su come configurare la funzione per utilizzarle, consulta la pagina [the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”](#).

Per saperne di più sull'utilizzo del formato di registro e delle opzioni a livello di registro con le funzioni Ruby Lambda, consulta le linee guida nelle sezioni seguenti.

### Utilizzo di log JSON strutturati con Ruby

Se selezioni JSON come formato di registro della tua funzione, Lambda invierà i log generati dalla `Logger` libreria standard Ruby come JSON strutturato. CloudWatch Ogni oggetto di log JSON contiene almeno quattro coppie chiave-valore con le seguenti chiavi:

- `"timestamp"`: l'ora in cui è stato generato il messaggio di log
- `"level"`: il livello di log assegnato al messaggio
- `"message"`: il contenuto del messaggio di log
- `"requestId"`: l'ID di richiesta univoco dell'invocazione alla funzione

La `Logger` libreria Ruby può anche aggiungere ulteriori coppie chiave-valore, ad esempio a questo oggetto JSON. `"logger"`

Gli esempi nelle sezioni seguenti mostrano come gli output di log generati utilizzando la `Logger` libreria Ruby vengono acquisiti in CloudWatch Logs quando si configura il formato di log della funzione come JSON.

Nota che se usi il `puts` metodo per produrre output di log di base come descritto in [the section called “Creazione di una funzione che restituisce i registri”](#), Lambda acquisirà questi output come testo semplice, anche se configuri il formato di registrazione della tua funzione come JSON.

Uscite di log JSON standard che utilizzano la libreria Ruby Logger

Il seguente frammento di codice di esempio e l'output del log mostrano come gli output di log standard generati utilizzando la `Logger` libreria Ruby vengono acquisiti in CloudWatch Logs quando il formato di registro della funzione è impostato su JSON.

Example Esempio di codice di registrazione Ruby

```
require 'logger'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  logger.info("Inside the handler function")
end
```

Example Esempio di record di registro JSON

```
{
  "timestamp": "2025-10-27T19:17:45.586Z",
  "level": "INFO",
  "message": "Inside the handler function",
  "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Registrazione di parametri aggiuntivi in JSON

Quando il formato di registro della funzione è impostato su JSON, puoi anche registrare parametri aggiuntivi con la `Logger` libreria Ruby passando un hash di coppie chiave-valore aggiuntive all'output del log.

Example Esempio di codice di registrazione Ruby

```
require 'logger'
require 'json'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
```

```
extra_params = { "a" => "b", "b" => [3] }  
logger.info({ message: "extra parameters example" }.merge(extra_params).to_json)  
end
```

### Example Esempio di record di registro JSON

```
{  
  "timestamp": "2025-11-02T15:26:28Z",  
  "level": "INFO",  
  "message": "extra parameters example",  
  "requestId": "3dbd5759-65f6-45f8-8d7d-5bdc79a3bd01",  
  "a": "b",  
  "b": [  
    3  
  ]  
}
```

### Registrazione delle eccezioni in JSON

Il seguente frammento di codice mostra come le eccezioni di Ruby vengono acquisite nell'output del registro della funzione quando configuri il formato di registro come JSON. Nota che agli output di log generati utilizzando `logger.error` con un'eccezione viene assegnato il livello di log `ERROR`.

### Example Esempio di codice di registrazione Ruby

```
require 'logger'  
  
def lambda_handler(event:, context:)  
  logger = Logger.new($stdout)  
  begin  
    raise "exception"  
  rescue => e  
    logger.error(e)  
  end  
end
```

### Example Esempio di record di registro JSON

```
{  
  "timestamp": "2025-11-02T16:18:57Z",  
  "level": "ERROR",
```

```
{
  "message": "exception",
  "stackTrace": [
    " /var/task/lambda_function.rb:4:in `lambda_handler'"
  ],
  "errorType": "RuntimeError",
  "errorMessage": "exception",
  "requestId": "3f9d155c-0f09-46b7-bdf1-e91dab220855",
  "location": "/var/task/lambda_function.rb:lambda_handler:6"
}
```

## Utilizzo del filtro a livello di log con Ruby

Configurando il filtro a livello di log, puoi scegliere di inviare solo i log con un determinato livello di registrazione o inferiore a Logs. CloudWatch Per informazioni su come configurare il filtraggio a livello di log della funzione, consulta la pagina [the section called “Log-level filtraggio”](#).

Per AWS Lambda filtrare i log delle applicazioni in base al relativo livello di registro, la funzione deve utilizzare i log in formato JSON. Puoi farlo in due modi:

- Crea output di registro utilizzando la `Logger` libreria Ruby standard e configura la tua funzione per utilizzare la formattazione dei log JSON. AWS Lambda quindi filtra gli output del registro utilizzando la coppia `"level"` chiave-valore nell'oggetto JSON descritto in [the section called “Utilizzo di log JSON strutturati con Ruby”](#) Per informazioni su come configurare il formato di registro della funzione, consulta [the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”](#)
- Utilizzate un'altra libreria o metodo di registrazione per creare log strutturati JSON nel codice che includono una coppia `"level"` chiave-valore che definisce il livello dell'output del log.

Puoi anche utilizzare un'putsistruzione per generare un oggetto JSON contenente un identificatore a livello di log. L'putsistruzione seguente produce un output in formato JSON in cui il livello di registro è impostato su INFO. AWS Lambda invierà l'oggetto JSON a CloudWatch Logs se il livello di registrazione della funzione è impostato su INFO, DEBUG o TRACE.

```
puts '{"msg":"My log message", "level":"info"}'
```

Per consentire a Lambda di filtrare i log della funzione, è necessario includere anche una coppia chiave-valore `"timestamp"` nell'output log JSON. L'ora deve essere specificata in un formato di timestamp [RFC 3339](#) valido. Se non fornisci un timestamp valido, Lambda assegnerà al log il livello INFO e aggiungerà un timestamp per tuo conto.

## Utilizzo di una libreria di registrazione alternativa

Se è necessario utilizzare una versione personalizzata della `logger` libreria, è possibile includerla nel pacchetto di distribuzione o in un livello Lambda e impostare la variabile di `RUBYLIB` ambiente nella `directory` della libreria. `lib` Il runtime Lambda caricherà la tua versione anziché quella in bundle.

Se il codice utilizza già un'altra libreria di registrazione per produrre log strutturati JSON, non è necessario apportare alcuna modifica. AWS Lambda non codifica due volte i log già codificati in JSON. Anche se configuri la tua funzione per utilizzare il formato di log JSON, i tuoi output di registrazione vengono visualizzati nella CloudWatch struttura JSON definita.

## Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor di codice incorporato, i log sono contenuti nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, trovi l'output del registro nella sezione `Dettagli`.

## Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Aprire la pagina dei gruppi di [log](#) sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (`/aws/lambda/your-function-name`).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'[istanza della funzione](#). Un flusso di log viene visualizzato quando si aggiorna la funzione Lambda e quando vengono create istanze aggiuntive per gestire chiamate simultanee. Per trovare i log per una chiamata specifica, ti consigliamo di strumentare la tua funzione con. AWS X-Ray X-Ray registra i dettagli sulla richiesta e sul flusso di log nella traccia.

## Visualizzazione dei log utilizzando AWS Command Line Interface (AWS CLI)

AWS CLI È uno strumento open source che puoi utilizzare per interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della [AWS CLI versione 2](#).

È possibile utilizzare [AWS CLI](#) per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

## Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "LogResult":
    "U1RBUIQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
  "ExecutedVersion": "$LATEST"
}
```

## Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'base64 utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'cli-binary-format opzione è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
```

```
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8   Duration: 79.67 ms   Billed
Duration: 80 ms   Memory Size: 128 MB   Max Memory Used: 73 MB
```

L'utilità base64 è disponibile su Linux, macOS e [Ubuntu su Windows](#). Gli utenti macOS potrebbero dover utilizzare base64 -D.

#### Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza sed per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando get-log-events.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come get-logs.sh.

L'cli-binary-formatopzione è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

#### Example(solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il comando seguente per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

#### Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.



```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version: $LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf\tINFO\tENVIRONMENT VARIABLES{\r \r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf\tINFO\tEVENT{\r \r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75 MB\t\n",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
}
```

```
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

## Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o [configurare un periodo di conservazione](#) trascorso il quale i log vengono eliminati automaticamente.

## Utilizzo della libreria dei logger Ruby

La [libreria di registrazione](#) Ruby restituisce log semplificati e facilmente leggibili. Utilizza l'utilità di registrazione per generare informazioni dettagliate, messaggi e codici di errore relativi alla tua funzione.

```
# lambda_function.rb

require 'logger'

def handler(event:, context:)
  logger = Logger.new($stdout)
  logger.info('## ENVIRONMENT VARIABLES')
  logger.info(ENV.to_a)
  logger.info('## EVENT')
  logger.info(event)
  event.to_a
end
```

L'output di logger include il livello del log, il timestamp e l'ID della richiesta.

```
START RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Version: $LATEST
[INFO] 2020-01-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    ##
ENVIRONMENT VARIABLES

[INFO] 2020-01-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125
    environ({'AWS_LAMBDA_LOG_GROUP_NAME': '/aws/lambda/my-function',
'AWS_LAMBDA_LOG_STREAM_NAME': '2020/01/31/[$LATEST]1bbe51xmplb34a2788dbaa7433b0aa4d',
'AWS_LAMBDA_FUNCTION_NAME': 'my-function', ...})

[INFO] 2020-01-31T22:12:58.535Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    ## EVENT

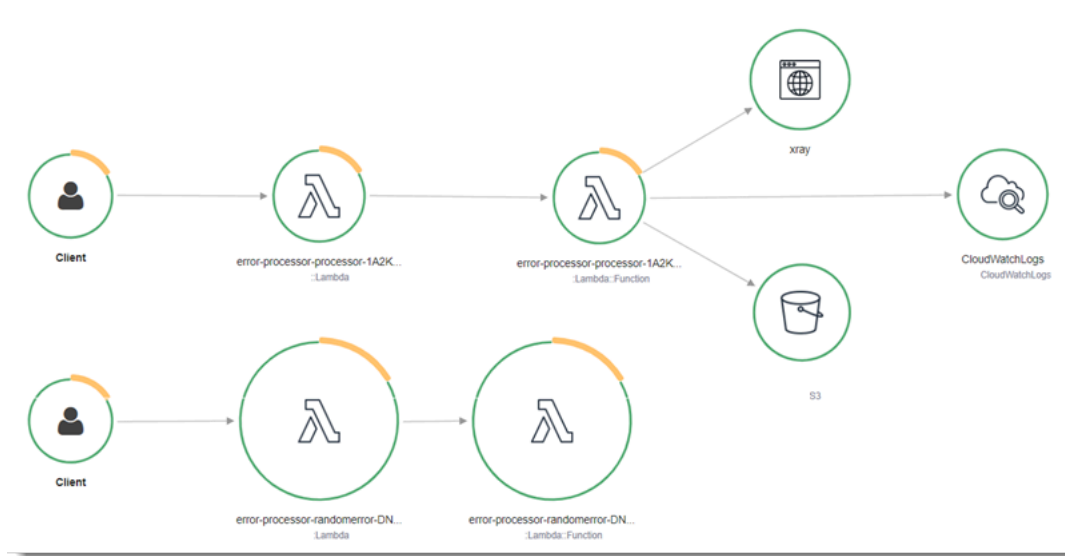
[INFO] 2020-01-31T22:12:58.535Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    {'key':
'value'}
```

```
END RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125
REPORT RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125  Duration: 2.75 ms   Billed
  Duration: 117 ms Memory Size: 128 MB Max Memory Used: 56 MB   Init Duration: 113.51 ms
XRAY TraceId: 1-5e34a66a-474xmpl7c2534a87870b4370   SegmentId: 073cxmpl3e442861
  Sampled: true
```

## Strumentazione del codice Ruby in AWS Lambda

Lambda si integra con AWS X-Ray per consentire di tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, dall'API di front-end allo storage e al database di back-end. Aggiungendo semplicemente la libreria X-Ray SDK alla configurazione di build, puoi registrare errori e latenza per qualsiasi chiamata effettuata dalla tua funzione a un servizio AWS.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico del [X-Ray servizio](#) mostra le informazioni sull'applicazione e su tutti i suoi componenti. Il seguente esempio mostra un'applicazione con due funzioni. La funzione principale elabora gli eventi e talvolta restituisce errori. La seconda funzione in alto elabora gli errori che appaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon CloudWatch Logs.



Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. In Strumenti di monitoraggio aggiuntivi scegli Modifica.

5. In CloudWatch Application Signals e AWS X-Ray, seleziona **Abilita** per le tracce di servizio Lambda.
6. Scegli **Save (Salva)**.

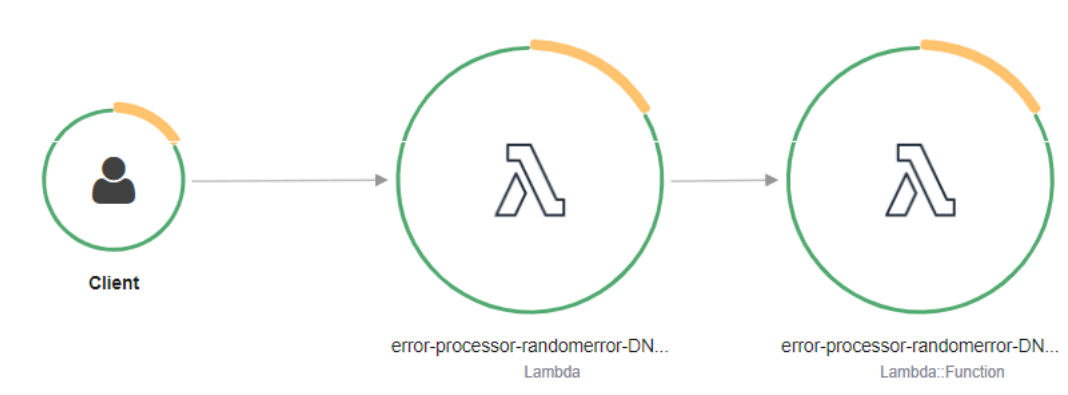
### Prezzi

Puoi utilizzare il X-Ray tracciamento gratuitamente ogni mese fino a un determinato limite come parte del piano AWS gratuito. Oltre tale soglia, vengono X-Ray addebitati i costi per l'archiviazione e il recupero delle tracce. Per ulteriori informazioni, consultare [Prezzi di AWS X-Ray](#).

La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al [ruolo di esecuzione](#) della funzione. Altrimenti, aggiungi la [AWSXRayDaemonWriteAccess](#) policy al ruolo di esecuzione.

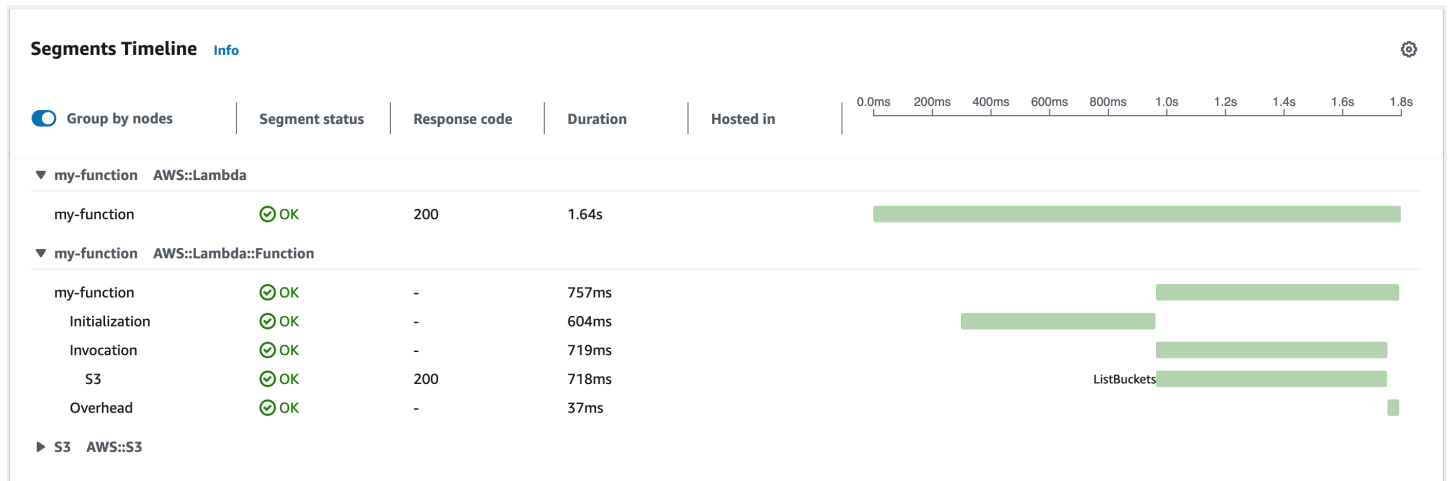
X-Ray non traccia tutte le richieste all'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento sia efficiente, fornendo comunque un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non puoi configurare la frequenza di X-Ray campionamento per le tue funzioni.

In X-Ray, una **traccia** registra le informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2

segmenti. Entrambi sono nominati `my-function`, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.

#### Note

AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti. Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regions, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia nel nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- **Inizializzazione** – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il [codice di inizializzazione](#). Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.
- **Chiamata**: rappresenta il tempo impiegato per eseguire il codice del gestore.

- **Overhead:** rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla struttura dei segmenti di funzioni vecchio e nuovo stile, consulta [the section called “Comprensione delle tracce X-Ray”](#).

È possibile strumentare il codice del gestore per registrare i metadati e tracciare le chiamate a valle. Per registrare i dettagli sulle chiamate effettuate dal tuo gestore ad altre risorse e servizi, usa l' X-Ray SDK per Ruby. Per ottenere l'SDK, aggiungere il pacchetto `aws-xray-sdk` alle dipendenze dell'applicazione.

Example [blank- /Gemfile ruby/function](#)

```
# Gemfile
source 'https://rubygems.org'

gem 'aws-xray-sdk', '0.11.4'
gem 'aws-sdk-lambda', '1.39.0'
gem 'test-unit', '3.3.5'
```

Per utilizzare i client AWS SDK, richiedi il `aws-xray-sdk/lambda` modulo dopo aver creato un client nel codice di inizializzazione.

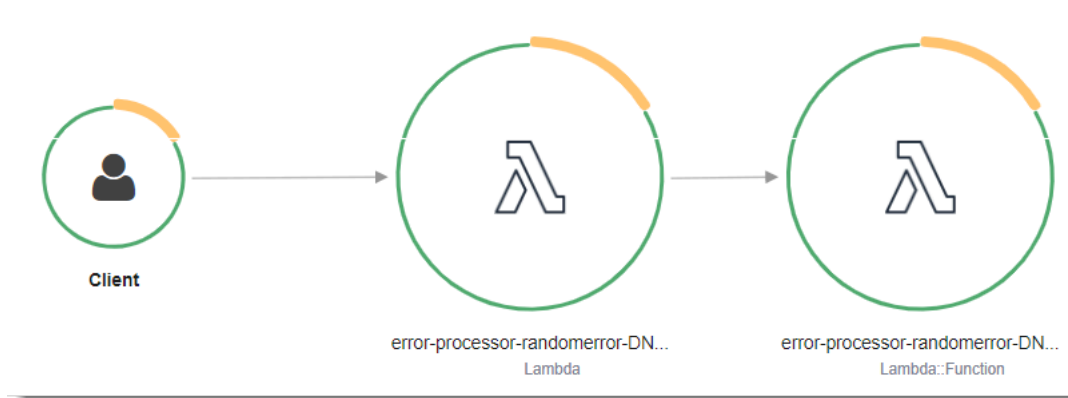
Example [blank- /lambda\\_function.rb ruby/function](#) — Tracciamento di un AWS Client SDK

```
# lambda_function.rb
require 'logger'
require 'json'
require 'aws-sdk-lambda'
$client = Aws::Lambda::Client.new()
$client.get_account_settings()

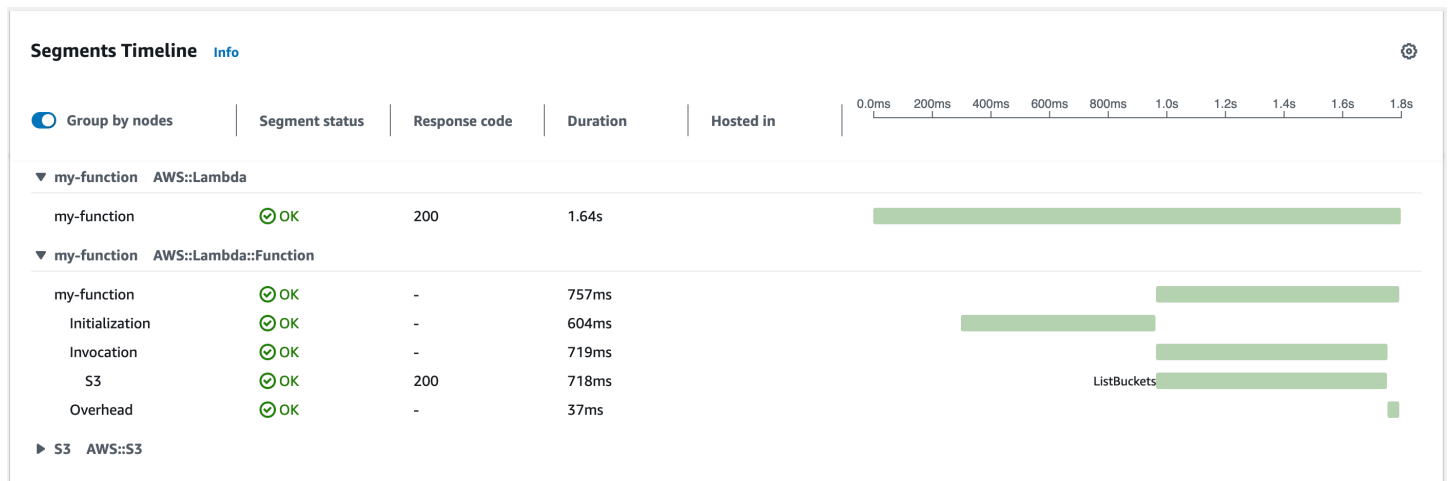
require 'aws-xray-sdk/lambda'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  ...
```

In X-Ray, una traccia registra le informazioni su una richiesta che viene elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.



### Note

AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti.

Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regioni, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia nel nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- **Inizializzazione** – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il [codice di inizializzazione](#). Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.
- **Chiamata**: rappresenta il tempo impiegato per eseguire il codice del gestore.
- **Overhead**: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla struttura dei segmenti di funzioni vecchio e nuovo stile, consulta [the section called “Comprensione delle tracce X-Ray”](#).

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta [The X-Ray SDK for Ruby](#) nella Developer Guide. AWS X-Ray

### Sections

- [Abilitazione del tracciamento attivo con l'API Lambda](#)
- [Attivazione della traccia attiva con CloudFormation](#)
- [Memorizzazione delle dipendenze di runtime in un layer](#)

## Abilitazione del tracciamento attivo con l'API Lambda

Per gestire la configurazione di tracciamento con AWS CLI or AWS SDK, utilizza le seguenti operazioni API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

Il seguente AWS CLI comando di esempio abilita la traccia attiva su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

## Attivazione della traccia attiva con CloudFormation

Per attivare la traccia su una `AWS::Lambda::Function` risorsa in un CloudFormation modello, utilizzate la `TracingConfig` proprietà.

Example [function-inline.yml](#) — Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Per una `AWS::Serverless::Function` risorsa AWS Serverless Application Model (AWS SAM), usa la `Tracing` proprietà.

Example [template.yml](#) — Configurazione di tracciamento

```
Resources:
```

```
function:
  Type: AWS::Serverless::Function
  Properties:
    Tracing: Active
    ...
```

## Memorizzazione delle dipendenze di runtime in un layer

Se si utilizza l' X-Ray SDK per configurare il codice funzione dei client AWS SDK, il pacchetto di distribuzione può diventare piuttosto ingombrante. Per evitare di caricare le dipendenze di runtime ogni volta che aggiorni il codice funzione, impacchetta l' X-Ray SDK in un layer Lambda. [???](#)

L'esempio seguente mostra una `AWS::Serverless::LayerVersion` risorsa che memorizza X-Ray l'SDK per Ruby.

Example [template.yml](#) — Livello delle dipendenze

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/.
      Tracing: Active
      Layers:
        - !Ref libs
        ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-ruby-lib
      Description: Dependencies for the blank-ruby sample app.
      ContentUri: lib/.
      CompatibleRuntimes:
        - ruby2.5
```

Con questa configurazione, si aggiorna il livello della libreria solo se si modificano le dipendenze di runtime. Poiché il pacchetto di implementazione della funzione contiene solo il codice, questo può contribuire a ridurre i tempi di caricamento.

La creazione di un livello per le dipendenze richiede modifiche alla compilazione per generare l'archivio dei livelli prima della distribuzione. Per un esempio funzionante, vedere l'applicazione di esempio [blank-ruby](#) .

## Compilazione di funzioni Lambda con Java

Puoi eseguire il codice Java in AWS Lambda. Lambda fornisce [Runtime](#) per Java che eseguono il tuo codice per elaborare gli eventi. Il codice viene eseguito in un ambiente Amazon Linux che include AWS le credenziali di un ruolo AWS Identity and Access Management (IAM) che gestisci.

Lambda supporta i seguenti runtime di Java.

| Nome     | Identificatore | Sistema operativo | Data di ritiro | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|----------|----------------|-------------------|----------------|---------------------------|-------------------------------|
| Java 2.5 | java25         | Amazon Linux 2023 | 30 giugno 2029 | 31 luglio 2029            | 31 agosto 2029                |
| Java 21  | java21         | Amazon Linux 2023 | 30 giugno 2029 | 31 luglio 2029            | 31 agosto 2029                |
| Java 17  | java17.al2023  | Amazon Linux 2023 | 30 giugno 2029 | 31 luglio 2029            | 31 agosto 2029                |
| Java 11  | java11.al2023  | Amazon Linux 2023 | 30 giugno 2029 | 31 luglio 2029            | 31 agosto 2029                |
| Java 8   | java8.al2023   | Amazon Linux 2023 | 30 giugno 2029 | 31 luglio 2029            | 31 agosto 2029                |
| Java 17  | java17         | Amazon Linux 2    | 30 giugno 2027 | 31 luglio 2027            | 31 agosto 2027                |
| Java 11  | java11         | Amazon Linux 2    | 30 giugno 2027 | 31 luglio 2027            | 31 agosto 2027                |
| Java 8   | java8.al2      | Amazon Linux 2    | 30 giugno 2027 | 31 luglio 2027            | 31 agosto 2027                |

AWS fornisce le seguenti librerie per le funzioni Java. Queste librerie sono disponibili tramite [Maven Central Repository](#).

- [com.amazonaws:aws-lambda-java-core](#) (obbligatorio): definisce le interfacce del metodo del gestore e l'oggetto contestuale che il runtime passa al gestore. Se definisci i propri tipi di input, questa è l'unica libreria necessaria.
- [com.amazonaws:aws-lambda-java-events](#): tipi di input per eventi da servizi che richiamano funzioni Lambda.
- [com.amazonaws:aws-lambda-java-log4j2](#): una libreria appender per Apache Log4j 2 che è possibile utilizzare per aggiungere l'ID della richiesta per la chiamata corrente ai [log della funzione](#).
- [AWS SDK per Java 2.0](#): l'AWS SDK ufficiale per il linguaggio di programmazione Java.
- [AWS Lambda Durable Execution SDK for Java](#): crea applicazioni affidabili in più fasi e flussi di lavoro AI che controllano l'avanzamento, ripristinano automaticamente gli errori e mettono in pausa l'esecuzione. Per ulteriori informazioni, consulta Lambda durable functions. [???](#)

Aggiungi queste librerie alla definizione di build come segue.

## Gradle

```
dependencies {  
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.2'  
    implementation 'com.amazonaws:aws-lambda-java-events:3.11.1'  
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.5.1'  
    implementation 'software.amazon.lambda.durable:aws-durable-execution-sdk-  
java:1.0.1'  
}
```

## Maven

```
<dependencies>  
  <dependency>  
    <groupId>com.amazonaws</groupId>  
    <artifactId>aws-lambda-java-core</artifactId>  
    <version>1.2.2</version>  
  </dependency>  
  <dependency>  
    <groupId>com.amazonaws</groupId>  
    <artifactId>aws-lambda-java-events</artifactId>  
    <version>3.11.1</version>  
  </dependency>  
  <dependency>  
    <groupId>com.amazonaws</groupId>
```

```
<artifactId>aws-lambda-java-log4j2</artifactId>
<version>1.5.1</version>
</dependency>
<dependency>
  <groupId>software.amazon.lambda.durable</groupId>
  <artifactId>aws-durable-execution-sdk-java</artifactId>
  <version>1.0.1</version>
</dependency>
</dependencies>
```

### Important

Non utilizzare componenti privati dell'API JDK, come campi, metodi o classi privati. Non-public I componenti dell'API possono cambiare o essere rimossi in qualsiasi aggiornamento, causando l'interruzione dell'applicazione.

Per informazioni sulla serializzazione personalizzata, vedere [the section called “Serializzazione personalizzata”](#).

Per creare una funzione Java

1. Aprire la [console Lambda](#).
2. Scegli Crea funzione.
3. Configura le impostazioni seguenti:
  - Nome della funzione: inserisci il nome della funzione.
  - Runtime: scegli Java 25.
4. Scegli Crea funzione.

La console crea una funzione Lambda con una classe del gestore denominata Hello. Poiché Java è un linguaggio compilato, non è possibile visualizzare o modificare il codice sorgente nella console Lambda, ma è possibile modificarne la configurazione, invocarla e configurare i trigger.

 Note

Per iniziare a sviluppare applicazioni nel tuo ambiente locale, implementa una delle applicazioni di [esempio](#) disponibili nell' GitHub archivio di questa guida.

La classe `Hello` ha una funzione denominata `handleRequest` che richiede un oggetto evento e un oggetto contesto. Questa è la [funzione del gestore](#) chiamata da Lambda quando la funzione viene richiamata. Il runtime della funzione Lambda riceve gli eventi di chiamata da e li passa al gestore. Nella configurazione della funzione il valore del gestore è `example.Hello::handleRequest`.

Per aggiornare il codice di funzione, crea un pacchetto di distribuzione costituito da un archivio `.zip` contenente il codice di funzione. Man mano che lo sviluppo della tua funzione procede vorrai memorizzare il tuo codice di funzione nel controllo del codice sorgente, aggiungere le librerie e automatizzare le distribuzioni. Inizia [creando un pacchetto di distribuzione](#) e aggiornando il codice dalla riga di comando.

Il runtime della funzione passa un oggetto contesto al gestore, oltre all'evento di chiamata. L'[oggetto contesto](#) contiene ulteriori informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. Altre informazioni sono disponibili con le variabili di ambiente.

La funzione Lambda è dotata di un gruppo di CloudWatch log Logs. Il runtime della funzione invia i dettagli su ogni chiamata a Logs. CloudWatch Si trasmette qualsiasi [log che la tua funzione emette](#) durante la chiamata. Se la funzione restituisce un errore, Lambda formatta l'errore e lo restituisce al chiamante.

## Argomenti

- [Definire l'handler della funzione Lambda in Java](#)
- [Distribuisce funzioni Lambda per Java con archivi di file `.zip` o JAR](#)
- [Distribuisce funzioni Java Lambda con immagini di container](#)
- [Utilizzo dei livelli per le funzioni Lambda in Java](#)
- [Personalizzare la serializzazione per le funzioni Lambda Java](#)
- [Personalizzare il comportamento di avvio del runtime Java per le funzioni Lambda](#)
- [Utilizzo dell'oggetto di contesto Lambda per recuperare le informazioni sulla funzione Java](#)
- [Registrare e monitorare funzioni Lambda in Java](#)
- [Strumentazione del codice Java in AWS Lambda](#)

- [Applicazioni di esempio Java per AWS Lambda](#)



## Definire l'handler della funzione Lambda in Java

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come lavorare con i gestori di funzioni Lambda in Java, incluse le opzioni per la configurazione del progetto, le convenzioni di denominazione e le best practice. Questa pagina include anche un esempio di funzione Java Lambda che acquisisce informazioni su un ordine, genera una ricevuta in un file di testo e lo inserisce in un bucket Amazon Simple Storage Service (Amazon S3). Per informazioni su come distribuire una funzione dopo averla scritta, consulta o [the section called “Implementazione di archivi di file .zip”](#) [the section called “Implementazione di immagini di container”](#)

### Sections

- [Configurazione del progetto Java handler](#)
- [Esempio di codice della funzione Java Lambda](#)
- [Definizioni di classe valide per i gestori Java](#)
- [Convenzioni di denominazione dei gestori](#)
- [Definizione e accesso all'oggetto evento di input](#)
- [Accesso e utilizzo dell'oggetto contestuale Lambda](#)
- [Utilizzo di AWS SDK per Java v2 nel tuo gestore](#)
- [Accesso alle variabili d'ambiente](#)
- [Utilizzo dello stato globale](#)
- [Best practice di codice per funzioni Lambda in Java](#)

### Configurazione del progetto Java handler

Quando si lavora con le funzioni Lambda in Java, il processo prevede la scrittura del codice, la compilazione e la distribuzione degli artefatti compilati in Lambda. È possibile inizializzare un progetto Java Lambda in vari modi. Ad esempio, puoi utilizzare strumenti come [Maven Archetype for Lambda functions](#), il comando AWS SAM CLI [sam init o persino una configurazione standard di progetto Java nel tuo IDE preferito](#), come IntelliJ IDEA o Visual Studio Code. In alternativa, è possibile creare manualmente la struttura di file richiesta.

Un tipico progetto di funzione Java Lambda segue questa struttura generale:

```
/project-root
# src
# main
# java
# example
#   OrderHandler.java (contains main handler)
#   <other_supporting_classes>
# build.gradle OR pom.xml
```

Puoi usare Maven o Gradle per creare il tuo progetto e gestire le dipendenze.

La logica principale del gestore per la funzione risiede in un file Java nella directory. `src/main/java/example` Nell'esempio di questa pagina, diamo un nome a questo file. `OrderHandler.java` Oltre a questo file, è possibile includere classi Java aggiuntive, se necessario. Quando distribuisce la tua funzione su Lambda, assicurati di specificare la classe Java che contiene il metodo di gestione principale che Lambda deve richiamare durante un'invocazione.

### Esempio di codice della funzione Java Lambda

Il seguente esempio di codice funzione Java 21 Lambda acquisisce informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3.

### Example Funzione Lambda **OrderHandler.java**

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.S3Exception;

import java.nio.charset.StandardCharsets;

/**
 * Lambda handler for processing orders and storing receipts in S3.
 */
public class OrderHandler implements RequestHandler<OrderHandler.Order, String> {

    private static final S3Client S3_CLIENT = S3Client.builder().build();
```

```

/**
 * Record to model the input event.
 */
public record Order(String orderId, double amount, String item) {}

@Override
public String handleRequest(Order event, Context context) {
    try {
        // Access environment variables
        String bucketName = System.getenv("RECEIPT_BUCKET");
        if (bucketName == null || bucketName.isEmpty()) {
            throw new IllegalArgumentException("RECEIPT_BUCKET environment variable
is not set");
        }

        // Create the receipt content and key destination
        String receiptContent = String.format("OrderID: %s\nAmount: $%.2f\nItem:
%s",
            event.orderId(), event.amount(), event.item());
        String key = "receipts/" + event.orderId() + ".txt";

        // Upload the receipt to S3
        uploadReceiptToS3(bucketName, key, receiptContent);

        context.getLogger().log("Successfully processed order " + event.orderId() +
            " and stored receipt in S3 bucket " + bucketName);
        return "Success";

    } catch (Exception e) {
        context.getLogger().log("Failed to process order: " + e.getMessage());
        throw new RuntimeException(e);
    }
}

private void uploadReceiptToS3(String bucketName, String key, String
receiptContent) {
    try {
        PutObjectRequest putObjectRequest = PutObjectRequest.builder()
            .bucket(bucketName)
            .key(key)
            .build();

        // Convert the receipt content to bytes and upload to S3

```

```
S3_CLIENT.putObject(putObjectRequest,
RequestBody.fromBytes(receiptContent.getBytes(StandardCharsets.UTF_8)));
    } catch (S3Exception e) {
        throw new RuntimeException("Failed to upload receipt to S3: " +
e.awsErrorDetails().errorMessage(), e);
    }
}
```

Questo file `OrderHandler.java` contiene le sezioni seguenti:

- `package example`: In Java, può essere qualsiasi cosa, ma deve corrispondere alla struttura delle directory del progetto. Qui, usiamo `package example` perché la struttura delle directory è `src/main/java/example`.
- `import` dichiarazioni: usate per importare le classi Java richieste dalla tua funzione Lambda.
- `public class OrderHandler ...`: definisce la tua classe Java e deve essere una definizione [di classe](#) valida.
- `private static final S3Client S3_CLIENT ...`: inizializza un client S3 al di fuori di qualsiasi metodo della classe. Ciò fa sì che Lambda esegua questo codice durante la [fase di inizializzazione](#).
- `public record Order ...`: definisce la forma dell'evento di input previsto in questo [record](#) Java personalizzato.
- `public String handleRequest(Order event, Context context)`: questo è il metodo dell'handler principale, che contiene la logica principale dell'applicazione.
- `private void uploadReceiptToS3(...)`: questo è un metodo helper a cui fa riferimento il metodo dell'handler principale `handleRequest`.

Esempio di file `build.gradle` e `pom.xml`

Il seguente file `build.gradle` o `pom.xml` accompagna questa funzione.

`build.gradle`

```
plugins {
    id 'java'
}

repositories {
```

```

    mavenCentral()
}

dependencies {
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.3'
    implementation 'software.amazon.awssdk:s3:2.28.29'
    implementation 'org.slf4j:slf4j-nop:2.0.16'
}

task buildZip(type: Zip) {
    from compileJava
    from processResources
    into('lib') {
        from configurations.runtimeClasspath
    }
}

java {
    sourceCompatibility = JavaVersion.VERSION_21
    targetCompatibility = JavaVersion.VERSION_21
}

build.dependsOn buildZip

```

## pom.xml

```

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>example-java</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>example-java-function</name>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>

```

```
<groupId>com.amazonaws</groupId>
<artifactId>aws-lambda-java-core</artifactId>
<version>1.2.3</version>
</dependency>
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>s3</artifactId>
  <version>2.28.29</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-nop</artifactId>
  <version>2.0.16</version>
</dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>3.5.2</version>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>3.4.1</version>
      <configuration>
        <createDependencyReducedPom>>false</createDependencyReducedPom>
        <filters>
          <filter>
            <artifact>*:*</artifact>
            <excludes>
              <exclude>META-INF/*</exclude>
              <exclude>META-INF/versions/**</exclude>
            </excludes>
          </filter>
        </filters>
      </configuration>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>shade</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

```

        </execution>
    </executions>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.13.0</version>
    <configuration>
        <release>21</release>
    </configuration>
</plugin>
</plugins>
</build>
</project>

```

Affinché questa funzione funzioni correttamente, il suo ruolo di [esecuzione](#) deve consentire l'azione. `s3:PutObject` Inoltre, assicuratevi di definire la variabile di ambiente `RECEIPT_BUCKET`. Dopo una chiamata riuscita, il bucket Amazon S3 dovrebbe contenere un file di ricevuta.

#### Note

Questa funzione potrebbe richiedere impostazioni di configurazione aggiuntive per funzionare correttamente senza scadere. Si consiglia di configurare 256 MB di memoria e un timeout di 10 secondi. La prima chiamata potrebbe richiedere più tempo a causa di un avvio a freddo. [???](#) Le chiamate successive dovrebbero essere eseguite molto più velocemente grazie al riutilizzo dell'ambiente di esecuzione.

## Definizioni di classe valide per i gestori Java

Per definire la tua classe, la libreria <https://github.com/aws/aws-lambda-java-libs/tree/master/aws-lambda-java-core> definisce due interfacce per i metodi handler. Usa le interfacce fornite per semplificare la configurazione del gestore e convalidare la firma del metodo in fase di compilazione.

- [com.amazonaws.services.lambda.runtime. RequestHandler](#)
- [com.amazonaws.services.lambda.runtime. RequestStreamHandler](#)

L'interfaccia `RequestHandler` è un tipo generico che accetta due parametri: il tipo di input e il tipo di output. Entrambi i tipi devono essere oggetti. In questo esempio, la nostra `OrderHandler` classe implementa `RequestHandler<OrderHandler.Order, String>`. Il tipo di input è il `Order` record che definiamo all'interno della classe e il tipo di output è `String`.

```
public class OrderHandler implements RequestHandler<OrderHandler.Order, String> {  
    ...  
}
```

Quando si utilizza questa interfaccia, il runtime Java deserializza l'evento nell'oggetto con il tipo di input e serializza l'output in testo. Utilizzare questa interfaccia quando la serializzazione integrata funziona con i tipi di input e output.

Per utilizzare la propria serializzazione, è possibile implementare l'interfaccia.

**RequestStreamHandler** Con questa interfaccia, Lambda passa al gestore un flusso di input e un flusso di output. Il gestore legge i byte dal flusso di input, scrive nel flusso di output e restituisce il valore `void`. Per un esempio di ciò utilizzando il runtime Java 21, vedere [HandlerStream.java](#).

Se state lavorando solo con tipi base e generici (ad esempio `StringInteger`, `List`, `oMap`) nella vostra funzione Java, non è necessario implementare un'interfaccia. Ad esempio, se la tua funzione accetta un `Map<String, String>` input e restituisce un `String`, la definizione della classe e la firma del gestore potrebbero assomigliare alle seguenti:

```
public class ExampleHandler {  
    public String handleRequest(Map<String, String> input, Context context) {  
        ...  
    }  
}
```

Inoltre, quando non si implementa un'interfaccia, l'[oggetto](#) `context` è opzionale. Ad esempio, la definizione della classe e la firma del gestore potrebbero essere le seguenti:

```
public class NoContextHandler {  
    public String handleRequest(Map<String, String> input) {  
        ...  
    }  
}
```



## Convenzioni di denominazione dei gestori

Per le funzioni Lambda in Java, se si implementa l'interfaccia `RequestHandler` o `RequestStreamHandler`, è necessario denominare il metodo del gestore principale `handleRequest`. Inoltre, includi il `@Override` tag sopra il tuo `handleRequest` metodo. Quando distribuisce la tua funzione su Lambda, specifica il gestore principale nella configurazione della tua funzione nel seguente formato:

- `<package>. <Class>` — Ad esempio, `example.OrderHandler`.

Per le funzioni Lambda in Java che non implementano l'`RequestStreamHandler` interfaccia `RequestHandler` o, puoi utilizzare qualsiasi nome per il gestore. Quando distribuisce la tua funzione su Lambda, specifica il gestore principale nella configurazione della tua funzione nel seguente formato:

- `<package>. <Class>:: <handler_method_name>` — Ad esempio, `example.Handler::mainHandler`.

## Definizione e accesso all'oggetto evento di input

JSON è il formato di input più comune e standard per le funzioni Lambda. In questo esempio, la funzione prevede un input simile a quanto segue:

```
{
  "orderId": "12345",
  "amount": 199.99,
  "item": "Wireless Headphones"
}
```

Quando si lavora con le funzioni Lambda in Java 17 o versioni successive, è possibile definire la forma dell'evento di input previsto come record Java. In questo esempio, definiamo un record all'interno della `OrderHandler` classe per rappresentare un `Order` oggetto:

```
public record Order(String orderId, double amount, String item) {}
```

Questo record corrisponde alla forma di input prevista. Dopo aver definito il record, puoi scrivere una firma del gestore che accetta un input JSON conforme alla definizione del record. Il runtime Java deserializza automaticamente questo JSON in un oggetto Java. È quindi possibile accedere ai campi dell'oggetto. Ad esempio, `event.orderId` recupera il valore di `orderId` dall'input originale.

 Note

I record Java sono una funzionalità solo dei runtime di Java 17 e versioni successive. In tutti i runtime di Java, è possibile utilizzare una classe per rappresentare i dati degli eventi. In questi casi, puoi usare una libreria come [jackson](#) per deserializzare gli input JSON.

## Altri tipi di eventi di input

Esistono molti possibili eventi di input per le funzioni Lambda in Java:

- `Integer`, `LongDouble`, e altri tipi numerici: l'evento è un numero senza formattazione aggiuntiva, ad esempio. `3.5` Il runtime Java converte il valore in un oggetto del tipo specificato.
- `String`: l'evento è una stringa JSON, incluse le virgolette, ad esempio `"My string"`. Il runtime converte il valore in un `String` oggetto senza virgolette.
- `List<Integer>`, `List<String>`, `List<Object>`, e tipi simili: l'evento è un array JSON. Il runtime lo deserializza in un oggetto del tipo o dell'interfaccia specificati.
- `InputStream`: l'evento è un tipo qualsiasi di JSON. Il runtime passa un flusso di byte del documento al gestore senza modifiche. Si deserializza l'output di input e scrittura in un flusso di output.
- Tipo di libreria: per gli eventi inviati da altri AWS servizi, usa i tipi nella libreria <https://github.com/aws/aws-lambda-java-libs/tree/main/aws-lambda-java-events>. Ad esempio, se la tua funzione Lambda viene richiamata da Amazon Simple Queue Service (SQS), utilizza l'oggetto come input. `SQSEvent`

## Accesso e utilizzo dell'oggetto contestuale Lambda

L'[oggetto contesto](#): contiene informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. In questo esempio, l'oggetto contesto è di tipo `com.amazonaws.services.lambda.runtime.Context` ed è il secondo argomento della funzione del gestore principale.

```
public String handleRequest(Order event, Context context) {  
    ...  
}
```

Se la tua classe implementa l' [RequestStreamHandler](#) interfaccia [RequestHandler](#) or, l'oggetto context è un argomento obbligatorio. In caso contrario, l'oggetto è facoltativo. Per ulteriori informazioni sulle firme valide accettate del gestore, consulta [the section called “Definizioni di classe valide per i gestori Java”](#).

Se si effettuano chiamate ad altri servizi utilizzando l' AWS SDK, l'oggetto context è richiesto in alcune aree chiave. Ad esempio, per produrre log delle funzioni per Amazon CloudWatch, puoi utilizzare il `context.getLogger()` metodo per ottenere un `LambdaLogger` oggetto per la registrazione. In questo esempio, possiamo usare il logger per registrare il log di un messaggio di errore se l'elaborazione fallisce per qualche motivo:

```
context.getLogger().log("Failed to process order: " + e.getMessage());
```

Oltre alla registrazione di log, puoi anche utilizzare l'oggetto contesto per il monitoraggio delle funzioni. Per ulteriori informazioni sulla copia di oggetti, consulta la sezione [the section called “Contesto”](#).

### Utilizzo di AWS SDK per Java v2 nel tuo gestore

Spesso, utilizzerai le funzioni Lambda per interagire o apportare aggiornamenti ad altre risorse. AWS Il modo più semplice per interfacciarsi con queste risorse è utilizzare l' AWS SDK per Java v2.

#### Note

L' AWS SDK per Java (v1) è in modalità manutenzione e terminerà il supporto il 31 dicembre 2025. Ti consigliamo di utilizzare solo l' AWS SDK per Java v2 in futuro.

Per aggiungere dipendenze SDK alla tua funzione, aggiungile nel tuo file `build.gradle` per Gradle o per Maven. `pom.xml` Ti consigliamo di aggiungere solo le librerie necessarie per la tua funzione. Nel codice di esempio precedente, abbiamo usato la libreria `software.amazon.awssdk.services.s3`. In Gradle, puoi aggiungere questa dipendenza aggiungendo la seguente riga nella sezione delle dipendenze del tuo: `build.gradle`

```
implementation 'software.amazon.awssdk:s3:2.28.29'
```

In Maven, aggiungi le seguenti righe nella sezione del tuo: `<dependencies>` `pom.xml`

```
<dependency>
```

```
<groupId>software.amazon.awssdk</groupId>
<artifactId>s3</artifactId>
<version>2.28.29</version>
</dependency>
```

### Note

Questa potrebbe non essere la versione più recente dell'SDK. Scegli la versione appropriata dell'SDK per la tua applicazione.

Quindi, importa le dipendenze direttamente nella tua classe Java:

```
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.S3Exception;
```

Il codice di esempio inizializza quindi un client Amazon S3 come segue:

```
private static final S3Client S3_CLIENT = S3Client.builder().build();
```

In questo esempio, abbiamo inizializzato il nostro client Amazon S3 all'esterno della funzione del gestore principale per evitare di doverlo inizializzare ogni volta che invochiamo la nostra funzione. Dopo aver inizializzato il client SDK, puoi utilizzarlo per interagire con altri servizi. AWS Il codice di esempio richiama l'API PutObject Amazon S3 nel modo seguente:

```
PutObjectRequest putObjectRequest = PutObjectRequest.builder()
    .bucket(bucketName)
    .key(key)
    .build();

// Convert the receipt content to bytes and upload to S3
S3_CLIENT.putObject(putObjectRequest,
    RequestBody.fromBytes(receiptContent.getBytes(StandardCharsets.UTF_8)));
```

## Accesso alle variabili d'ambiente

Nel codice del gestore, puoi fare riferimento a qualsiasi variabile di [ambiente](#) utilizzando il metodo `System.getenv()`. In questo esempio, facciamo riferimento alla variabile di `RECEIPT_BUCKET` ambiente definita utilizzando la seguente riga di codice:

```
String bucketName = System.getenv("RECEIPT_BUCKET");
if (bucketName == null || bucketName.isEmpty()) {
    throw new IllegalArgumentException("RECEIPT_BUCKET environment variable is not set");
}
```

## Utilizzo dello stato globale

Lambda esegue il codice statico e il costruttore della classe durante la [fase di inizializzazione](#) prima di richiamare la funzione per la prima volta. Le risorse create durante l'inizializzazione restano in memoria tra le invocazioni, in modo da evitare di doverle creare ogni volta che si invoca la funzione.

Nel codice di esempio, il codice di inizializzazione del client S3 non rientra nel metodo del gestore principale. Il runtime inizializza il client prima che la funzione gestisca il primo evento e il client rimane disponibile per il riutilizzo in tutte le chiamate.

## Best practice di codice per funzioni Lambda in Java

Segui le linee guida riportate nell'elenco seguente per utilizzare le best practice di codifica durante la creazione delle funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. Con questo approccio, è possibile creare una funzione più testabile a livello di unità.
- Controllare le dipendenze nel pacchetto di distribuzione della funzione. L'ambiente di esecuzione AWS Lambda contiene diverse librerie. Per abilitare l'ultimo set di funzionalità e aggiornamenti di sicurezza, Lambda aggiorna periodicamente queste librerie. Questi aggiornamenti potrebbero introdurre lievi modifiche al comportamento della funzione Lambda. Per mantenere il controllo completo delle dipendenze utilizzate dalla funzione, inserire tutte le dipendenze nel pacchetto di implementazione.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del [contesto di esecuzione](#). Preferire ad esempio l'utilizzo di framework di inserimento di dipendenze Java, come [Dagger](#) o [Guice](#), rispetto a framework più complessi come [Spring Framework](#).
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. Ciò riduce il tempo necessario per scaricare e decomprimere il pacchetto di distribuzione prima dell'invocazione. Per le funzioni create in Java, evita di caricare l'intera libreria AWS SDK come parte del pacchetto di distribuzione. Dipendi invece in modo selettivo dai moduli che raccolgono i componenti dell'SDK di cui hai bisogno (ad esempio, DynamoDB, moduli SDK

Amazon S3 e librerie di base Lambda sul sito web). <https://github.com/aws/aws-lambda-java-libs>  
GitHub

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory /tmp. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, consulta Reusing Connections with in. [Keep-Alive Node.js](#)

Utilizzare [le variabili di ambiente](#) per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su 0 per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionali alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha una dipendenza su queste API non pubbliche. Consulta il [riferimento all'API](#) per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente

gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta [Come posso rendere idempotente la mia funzione Lambda?](#).

- Evita di usare la cache DNS Java. Le funzioni Lambda memorizzano già nella cache le risposte DNS. Se viene utilizzata un'altra cache DNS, è possibile che si verifichino dei timeout di connessione.

La classe `java.util.logging.Logger` può abilitare indirettamente la cache DNS JVM. Per sovrascrivere le impostazioni predefinite, imposta [networkaddress.cache.ttl](#) su 0 prima dell'inizializzazione di `logger`. Esempio:

```
public class MyHandler {
    // first set TTL property
    static{
        java.security.Security.setProperty("networkaddress.cache.ttl" , "0");
    }
    // then instantiate logger
    var logger = org.apache.logging.log4j.LogManager.getLogger(MyHandler.class);
}
```

- Ridurre il tempo necessario a Lambda per decomprimere i pacchetti di distribuzione creati in Java inserendo i file `.jar` della dipendenza in una directory `/lib` separata. Questo è più veloce rispetto all'inserimento di tutto il codice della funzione in un singolo `jar` con un numero elevato di file. `.class` Per istruzioni, consulta [Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR](#).

## Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR

Il codice della tua AWS Lambda funzione è costituito da script o programmi compilati e dalle relative dipendenze. Utilizza un pacchetto di implementazione per distribuire il codice della funzione a Lambda. Lambda supporta due tipi di pacchetti di implementazione: immagini di container e archivi di file .zip.

Questa pagina descrive come creare il pacchetto di distribuzione come file .zip o file Jar e quindi utilizzare il pacchetto di distribuzione per distribuire il codice della funzione utilizzando (). AWS Lambda AWS Command Line Interface AWS CLI

### Important

Java 25 ha introdotto il supporto per le cache Ahead-of-Time (AOT). Consigliamo vivamente di non utilizzare le cache AOT quando si distribuiscono le funzioni come archivi di file .zip o JAR, poiché le cache potrebbero causare un comportamento imprevisto quando Lambda aggiorna il runtime gestito. Per ulteriori informazioni, consulta. [Ahead-of-Time \(AOT\) e cache CDS](#)

### Sections

- [Prerequisiti](#)
- [Strumenti e librerie](#)
- [Creazione di un pacchetto di distribuzione con Gradle](#)
- [Utilizzo dei livelli per le dipendenze](#)
- [Creazione di un pacchetto di distribuzione con Maven](#)
- [Caricamento di un pacchetto di implementazione con la console Lambda](#)
- [Caricamento di un pacchetto di distribuzione con AWS CLI](#)
- [Caricamento di un pacchetto di distribuzione con AWS SAM](#)

### Prerequisiti

AWS CLI È uno strumento open source che puoi utilizzare per interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della [AWS CLI versione 2](#).



## Strumenti e librerie

AWS fornisce le seguenti librerie per le funzioni Java. Queste librerie sono disponibili tramite [Maven Central Repository](#).

- [com.amazonaws:aws-lambda-java-core](#) (obbligatorio): definisce le interfacce del metodo del gestore e l'oggetto contestuale che il runtime passa al gestore. Se definisci i propri tipi di input, questa è l'unica libreria necessaria.
- [com.amazonaws:aws-lambda-java-events](#): tipi di input per eventi da servizi che richiamano funzioni Lambda.
- [com.amazonaws:aws-lambda-java-log4j2](#): una libreria appender per Apache Log4j 2 che è possibile utilizzare per aggiungere l'ID della richiesta per la chiamata corrente ai [log della funzione](#).
- [AWS SDK per Java 2.0](#): l'AWS SDK ufficiale per il linguaggio di programmazione Java.
- [AWS Lambda Durable Execution SDK for Java](#): crea applicazioni affidabili in più fasi e flussi di lavoro AI che controllano l'avanzamento, ripristinano automaticamente gli errori e mettono in pausa l'esecuzione. Per ulteriori informazioni, consulta [Lambda durable functions](#). ???

Aggiungi queste librerie alla definizione di build come segue.

### Gradle

```
dependencies {  
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.2'  
    implementation 'com.amazonaws:aws-lambda-java-events:3.11.1'  
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.5.1'  
    implementation 'software.amazon.lambda.durable:aws-durable-execution-sdk-  
java:1.0.1'  
}
```

### Maven

```
<dependencies>  
  <dependency>  
    <groupId>com.amazonaws</groupId>  
    <artifactId>aws-lambda-java-core</artifactId>  
    <version>1.2.2</version>  
  </dependency>  
  <dependency>  
    <groupId>com.amazonaws</groupId>
```

```
<artifactId>aws-lambda-java-events</artifactId>
<version>3.11.1</version>
</dependency>
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-lambda-java-log4j2</artifactId>
  <version>1.5.1</version>
</dependency>
<dependency>
  <groupId>software.amazon.lambda.durable</groupId>
  <artifactId>aws-durable-execution-sdk-java</artifactId>
  <version>1.0.1</version>
</dependency>
</dependencies>
```

Per creare un pacchetto di distribuzione, compila il codice della funzione e le dipendenze in un singolo file .zip o un file di archivio Java (JAR). Per Gradle, [utilizza il tipo di build Zip](#). Per Apache Maven, [utilizza il plugin Maven Shade](#). Per caricare il pacchetto di distribuzione, usa la console Lambda, l'API Lambda o (). AWS Serverless Application Model AWS SAM

#### Note

Per mantenere contenute le dimensioni del pacchetto di distribuzione, raggruppa le dipendenze della funzione in livelli. I livelli consentono di gestire le dipendenze in modo indipendente, possono essere utilizzate da più funzioni e possono essere condivisi con altri account. Per ulteriori informazioni, consulta [the section called "Livelli"](#).

## Creazione di un pacchetto di distribuzione con Gradle

Per creare un pacchetto di implementazione con il codice e le dipendenze della funzione in Gradle, utilizza il tipo di compilazione Zip. Ecco un esempio tratto da un [file build.gradle di esempio completo](#):

Example build.gradle: attività di compilazione

```
task buildZip(type: Zip) {
    into('lib') {
        from(jar)
        from(configurations.runtimeClasspath)
    }
}
```

```
}  
}
```

Questa configurazione di build crea un pacchetto di distribuzione nella directory `build/distributions`. All'interno dell'istruzione `into('lib')`, l'attività `jar` assembla un archivio `jar` contenente le classi principali in una cartella denominata `lib`. Inoltre, l'istruzione `configurations.runtimeClassPath` copia le librerie delle dipendenze dal classpath della build in una cartella denominata `lib`.

#### Example build.gradle: dipendenze

```
dependencies {  
    ...  
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.2'  
    implementation 'com.amazonaws:aws-lambda-java-events:3.11.1'  
    implementation 'org.apache.logging.log4j:log4j-api:2.17.1'  
    implementation 'org.apache.logging.log4j:log4j-core:2.17.1'  
    runtimeOnly 'org.apache.logging.log4j:log4j-slf4j18-impl:2.17.1'  
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.5.1'  
    ...  
}
```

Lambda carica i file JAR in ordine alfabetico Unicode. Se più file JAR nella directory `lib` contengono la stessa classe, viene utilizzato il primo. È possibile utilizzare il seguente script di shell per identificare le classi duplicate:

#### Example test-zip.sh

```
mkdir -p expanded  
unzip path/to/my/function.zip -d expanded  
find ./expanded/lib -name '*.jar' | xargs -n1 zipinfo -1 | grep '.*.class' | sort |  
    uniq -c | sort
```

#### Utilizzo dei livelli per le dipendenze

Puoi impacchettare le dipendenze della tua funzione in livelli per mantenere piccolo il tuo pacchetto di distribuzione e gestire le dipendenze in modo indipendente. Per ulteriori informazioni, consulta [the section called “Livelli”](#).

## Creazione di un pacchetto di distribuzione con Maven

Per creare un pacchetto di distribuzione con Maven, utilizzare il [plugin Maven Shade](#). Il plugin crea un file JAR che contiene il codice della funzione compilato e tutte le relative dipendenze.

Example pom.xml: configurazione del plugin

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <version>3.2.2</version>
  <configuration>
    <createDependencyReducedPom>>false</createDependencyReducedPom>
  </configuration>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>shade</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Per compilare il pacchetto di distribuzione, utilizzare il comando `mvn package`.

```
[INFO] Scanning for projects...
[INFO] -----< com.example:java-maven >-----
[INFO] Building java-maven-function 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
...
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ java-maven ---
[INFO] Building jar: target/java-maven-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-shade-plugin:3.2.2:shade (default) @ java-maven ---
[INFO] Including com.amazonaws:aws-lambda-java-core:jar:1.2.2 in the shaded jar.
[INFO] Including com.amazonaws:aws-lambda-java-events:jar:3.11.1 in the shaded jar.
[INFO] Including joda-time:joda-time:jar:2.6 in the shaded jar.
[INFO] Including com.google.code.gson:gson:jar:2.8.6 in the shaded jar.
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing target/java-maven-1.0-SNAPSHOT.jar with target/java-maven-1.0-SNAPSHOT-shaded.jar
[INFO] -----
```

```
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 8.321 s
[INFO] Finished at: 2020-03-03T09:07:19Z
[INFO] -----
```

Questo comando genera un file JAR nella directory target.

#### Note

Se stai lavorando con un [JAR multi-rilascio \(MRJAR\)](#), devi includere l'MRJAR (ossia il JAR shaded prodotto dal plug-in Maven Shade) nella directory `lib` e comprimerlo prima di caricare il pacchetto di implementazione in Lambda. Altrimenti, Lambda potrebbe non decomprimere correttamente il file JAR, facendo sì che il `MANIFEST.MF` file venga ignorato.

Se si utilizza la libreria appender (`aws-lambda-java-log4j2`), è necessario configurare anche un trasformatore per il plugin Maven Shade. La libreria del trasformatore combina le versioni di un file di cache presenti sia nella libreria appender che in Log4j.

Example pom.xml: configurazione del plugin con l'appenders Log4j 2

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <version>3.2.2</version>
  <configuration>
    <createDependencyReducedPom>>false</createDependencyReducedPom>
  </configuration>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>shade</goal>
      </goals>
      <configuration>
        <transformers>
          <transformer
implementation="com.github.edwgiz.maven_shade_plugin.log4j2_cache_transformer.PluginsCacheFile
          </transformer>
        </transformers>
      </configuration>
    </execution>
  </executions>
</plugin>
```

```
</execution>
</executions>
<dependencies>
  <dependency>
    <groupId>com.github.edwgiz</groupId>
    <artifactId>maven-shade-plugin.log4j2-cache-file-transformer</artifactId>
    <version>2.13.0</version>
  </dependency>
</dependencies>
</plugin>
```

## Caricamento di un pacchetto di implementazione con la console Lambda

Per creare una nuova funzione, devi prima creare la funzione nella console, quindi devi caricare il tuo file .zip o JAR. Per aggiornare una funzione esistente, apri la pagina relativa alla tua funzione, quindi segui la stessa procedura per aggiungere il file .zip o JAR aggiornato.

Se il file del pacchetto di implementazione ha dimensioni inferiori a 50 MB, è possibile creare o aggiornare una funzione caricando il file direttamente dal computer locale. Per i file .zip o JAR di dimensioni superiori a 50 MB, prima è necessario caricare il pacchetto in un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il Console di gestione AWS, consulta [Guida introduttiva ad Amazon S3](#). Per caricare file utilizzando il AWS CLI, consulta [Sposta oggetti](#) nella Guida per l'AWS CLI utente.

### Note

Non è possibile modificare il [tipo di pacchetto di implementazione](#) (.zip o immagine di container) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio di file .zip. È necessario creare una nuova funzione.

## Creazione di una nuova funzione (console)

1. Apri la [pagina Funzioni](#) della console Lambda e scegli Crea funzione.
2. Scegli Author from scratch (Crea da zero).
3. In Basic information (Informazioni di base) eseguire queste operazioni:
  - a. In Nome funzione, inserisci il nome della funzione.
  - b. Per Runtime, seleziona il runtime che desideri utilizzare.

- c. (Facoltativo) Per Architettura, scegli l'architettura del set di istruzioni per la funzione. L'architettura predefinita è x86\_64. Assicurati che il pacchetto di implementazione per la tua funzione sia compatibile con l'architettura del set di istruzioni scelta.
4. (Opzionale) In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito. Puoi creare un nuovo ruolo di esecuzione o utilizzare un ruolo esistente.
5. Scegli Crea funzione. Lambda crea una funzione di base "Hello world" utilizzando il runtime scelto.

#### Caricamento di un archivio .zip o JAR dal computer locale (console)

1. Nella [pagina Funzioni](#) della console Lambda, scegli la funzione per cui vuoi caricare il file .zip o JAR.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli File .zip o .jar.
5. Per caricare il file .zip o JAR, procedi come segue:
  - a. Seleziona Carica, quindi seleziona il tuo file .zip o JAR nel selettore di file.
  - b. Seleziona Apri.
  - c. Scegli Save (Salva).

#### Caricamento di un archivio .zip o JAR da un bucket Amazon S3 (console)

1. Nella [pagina Funzioni](#) della console Lambda, scegli la funzione per cui vuoi caricare un nuovo file .zip o JAR.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli Posizione Amazon S3.
5. Incolla l'URL del link Amazon S3 del tuo file .zip e scegli Salva.

#### Caricamento di un pacchetto di distribuzione con AWS CLI

È possibile utilizzare la [AWS CLI](#) per creare una nuova funzione o aggiornare una funzione esistente mediante un file .zip o JAR. Utilizza i comandi [create-function](#) e [update-function-code](#)

per implementare il tuo pacchetto .zip o JAR. Se il file ha dimensioni inferiori a 50 MB, è possibile caricare il pacchetto da una posizione nella macchina di compilazione locale. Per i file di dimensioni superiori, è necessario caricare il pacchetto .zip o JAR da un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando AWS CLI, consulta [Move objects](#) nella Guida per l'utente. AWS CLI

#### Note

Se carichi il tuo file .zip o JAR da un bucket Amazon S3 utilizzando AWS CLI, il bucket deve trovarsi nella stessa posizione della tua funzione. Regione AWS

Per creare una nuova funzione utilizzando un file .zip o JAR con AWS CLI, devi specificare quanto segue:

- Il nome della funzione (`--function-name`)
- runtime () `--runtime` della tua funzione
- L'Amazon Resource Name (ARN) del ruolo di [esecuzione della tua funzione](#) () `--role`
- Il nome del metodo del gestore nel codice della funzione (`--handler`)

È inoltre necessario specificare la posizione del file .zip o JAR. Se il file .zip o JAR si trova in una cartella nella macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda create-function --function-name myFunction \  
--runtime java25 --handler example.handler \  
--role arn:aws:iam::123456789012:role/service-role/my-lambda-role \  
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza l'opzione `--code` illustrata nel seguente comando di esempio. È necessario utilizzare il parametro `S3ObjectVersion` solo per gli oggetti con controllo delle versioni.

```
aws lambda create-function --function-name myFunction \  
--runtime java25 --handler example.handler \  
--role arn:aws:iam::123456789012:role/service-role/my-lambda-role \  
--code S3Bucket=amzn-s3-demo-  
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```



Per aggiornare una funzione esistente mediante la CLI, specifica il nome della funzione utilizzando il parametro `--function-name`. È inoltre necessario specificare la posizione del file `.zip` che desideri utilizzare per aggiornare il codice della funzione. Se il file `.zip` si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda update-function-code --function-name myFunction \  
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file `.zip` in un bucket Amazon S3, utilizza le opzioni `--s3-bucket` e `--s3-key` come illustrato nel seguente comando di esempio. È necessario utilizzare il parametro `--s3-object-version` solo per gli oggetti con controllo delle versioni.

```
aws lambda update-function-code --function-name myFunction \  
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject  
Version
```

## Caricamento di un pacchetto di distribuzione con AWS SAM

Puoi utilizzarlo AWS SAM per automatizzare le distribuzioni del codice funzionale, della configurazione e delle dipendenze. AWS SAM è un'estensione CloudFormation che fornisce una sintassi semplificata per la definizione di applicazioni serverless. Il modello di esempio seguente definisce una funzione con un pacchetto di distribuzione nella directory `build/distributions` utilizzata da Gradle:

### Example template.yml

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: 'AWS::Serverless-2016-10-31'  
Description: An AWS Lambda application that calls the Lambda API.  
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      CodeUri: build/distributions/java-basic.zip  
      Handler: example.Handler  
      Runtime: java25  
      Description: Java function  
      MemorySize: 512  
      Timeout: 10
```

```
# Function's execution role
Policies:
  - AWSLambdaBasicExecutionRole
  - AWSLambda_ReadOnlyAccess
  - AWSXrayWriteOnlyAccess
  - AWSLambdaVPCLambdaAccessExecutionRole
Tracing: Active
```

Per creare la funzione, utilizzare i comandi `package` e `deploy`. Questi comandi sono personalizzazioni per l' AWS CLI. Essi avvolgono altri comandi per caricare il pacchetto di distribuzione Amazon S3, riscrivere il modello con l'URI dell'oggetto e aggiornare il codice della funzione.

Lo script di esempio seguente esegue una compilazione Gradle e carica il pacchetto di distribuzione creato. Crea uno CloudFormation stack la prima volta che lo esegui. Se lo stack esiste già, lo script lo aggiorna.

#### Example `deploy.sh`

```
#!/bin/bash
set -eo pipefail
aws cloudformation package --template-file template.yml --s3-bucket MY_BUCKET --output-template-file out.yml
aws cloudformation deploy --template-file out.yml --stack-name java-basic --capabilities CAPABILITY_NAMED_IAM
```

Per un esempio pratico completo, consulta le seguenti applicazioni di esempio:

#### Applicazioni Lambda di esempio in Java

- [example-java](#): una funzione Java che dimostra come utilizzare Lambda per elaborare gli ordini. Questa funzione illustra come definire e deserializzare un oggetto evento di input personalizzato, utilizzare l' AWS SDK e registrare l'output.
- [java-basic](#): una raccolta di funzioni Java minimali con unit test e configurazione della registrazione dei log delle variabili.
- [java-events](#): una raccolta di funzioni Java che contengono codice skeleton per la gestione degli eventi di vari servizi, ad esempio Gateway Amazon API, Amazon SQS e Amazon Kinesis. Queste funzioni utilizzano la versione più recente della libreria [aws-lambda-java-events](#) (3.0.0 e versioni successive). Questi esempi non richiedono l'SDK come dipendenza. AWS

- [s3-java](#) – Una funzione Java che elabora gli eventi di notifica da Amazon S3 e utilizza la Java Class Library (JCL) per creare anteprime dai file di immagine caricati.
- [layer-java](#): una funzione Java che illustra come utilizzare un livello Lambda per impacchettare dipendenze separate dal codice della funzione principale.

## Distribuisci funzioni Java Lambda con immagini di container

Esistono tre modi per creare un'immagine di container per una funzione Lambda in Java:

- [Usando un AWS immagine base per Java](#)

[Le immagini di base AWS](#) sono precaricate con un runtime in linguaggio, un client di interfaccia di runtime per gestire l'interazione tra Lambda e il codice della funzione e un emulatore di interfaccia di runtime per i test locali.

- [Usando un AWS OS-only immagine di base](#)

[AWS OS-only le immagini di base](#) contengono una distribuzione Amazon Linux e l'emulatore dell'interfaccia [di runtime](#). Queste immagini vengono comunemente utilizzate per creare immagini contenitore per linguaggi compilati, come [Go](#) e [Rust](#), e per una lingua o una versione per la quale Lambda non fornisce un'immagine di base, come 19. Node.js Puoi anche utilizzare immagini di OS-only base per implementare un runtime [personalizzato](#). Per rendere l'immagine compatibile con Lambda, devi includere il [client di interfaccia di runtime per Java](#) nell'immagine.

- [Usare un'immagine non di AWS base](#)

È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere il [client di interfaccia di runtime per Java](#) nell'immagine.

### Tip

Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta [Utilizzo di compilazioni a più fasi](#) nella documentazione Docker. Per creare immagini di container efficienti, segui le [best practice per scrivere file Docker](#).

Questa pagina spiega come creare, testare e implementare le immagini di container per Lambda.

### Argomenti

- [AWS immagini di base per Java](#)
- [Usando un AWS immagine base per Java](#)
- [Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime](#)

## AWS immagini di base per Java

AWS fornisce le seguenti immagini di base per Java:

| Tag                  | Runtime  | Sistema operativo | Dockerfile                                           | Raggiunta obsolescenza |
|----------------------|----------|-------------------|------------------------------------------------------|------------------------|
| 25                   | Java 2.5 | Amazon Linux 2023 | <a href="#">Dockerfile per Java 25 in poi GitHub</a> | 30 giugno 2029         |
| 21                   | Java 21  | Amazon Linux 2023 | <a href="#">Dockerfile per Java 21 in poi GitHub</a> | 30 giugno 2029         |
| 17<br>giugno<br>2023 | Java 17  | Amazon Linux 2023 | <a href="#">Dockerfile per Java 17 in poi GitHub</a> | 30 giugno 2029         |
| 11<br>giugno<br>2023 | Java 11  | Amazon Linux 2023 | <a href="#">Dockerfile per Java 11 in poi GitHub</a> | 30 giugno 2029         |
| 8. nel<br>2023       | Java 8   | Amazon Linux 2023 | <a href="#">Dockerfile per Java 8 in poi GitHub</a>  | 30 giugno 2029         |
| 17                   | Java 17  | Amazon Linux 2    | <a href="#">Dockerfile per Java 17 in poi GitHub</a> | 30 giugno 2027         |
| 11                   | Java 11  | Amazon Linux 2    | <a href="#">Dockerfile per Java 11 in poi GitHub</a> | 30 giugno 2027         |
| 8.al2                | Java 8   | Amazon Linux 2    | <a href="#">Dockerfile per Java 8 in poi GitHub</a>  | 30 giugno 2027         |

Repository Amazon ECR: [gallery.ecr. aws/lambda/java](https://gallery.ecr.aws/lambda/java)

Le immagini di base Java 21 e versioni successive si basano sull'[immagine di container minima di Amazon Linux 2023](#). Le immagini di base precedenti utilizzano Amazon Linux 2. AL2023 offre diversi vantaggi rispetto ad Amazon Linux 2, tra cui un'impronta di implementazione ridotta e versioni aggiornate di librerie come `glibc`.

AL2023-based le immagini utilizzano `microdnf` (con collegamenti simbolici `dnf`) come gestore di pacchetti invece di `yum`, che è il gestore di pacchetti predefinito in Amazon Linux 2. `microdnf` è un'implementazione autonoma di `dnf`. Per un elenco dei pacchetti inclusi nelle AL2023-based immagini, consulta le colonne Minimal Container in [Confronto dei pacchetti installati su Amazon Linux 2023 Container Images](#). Per ulteriori informazioni sulle differenze tra AL2023 e Amazon Linux 2, consulta [Introduzione del runtime di Amazon Linux 2023 per AWS Lambda](#) nel blog AWS Compute.

#### Note

Per eseguire AL2023-based le immagini localmente, incluso con AWS Serverless Application Model (AWS SAM), devi utilizzare la versione Docker 20.10.10 o successiva.

Usando un'AWS immagine base per Java

#### Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Java (ad esempio, [Amazon Corretto](#))
- [Apache Maven](#) o [Gradle](#)
- [AWS CLI versione 2](#)
- [Docker](#) (versione minima 25.0.0)
- Il [plugin buildx](#) di Docker.

Creazione di un'immagine da un'immagine di base

#### Maven

1. Esegui il comando seguente per creare un progetto Maven utilizzando l'[archetipo per Lambda](#). I parametri seguenti sono obbligatori:
  - `service` — Il Servizio AWS client da utilizzare nella funzione Lambda. Per un elenco delle fonti disponibili, vedere [aws-sdk-java- on. v2/services](#) GitHub
  - `regione`: la Regione AWS posizione in cui desideri creare la funzione Lambda.
  - `groupId`: il namespace completo del pacchetto dell'applicazione.

- `artifactId`: il nome del progetto. Questo sarà il nome della directory per il progetto.

In Linux e macOS, esegui questo comando:

```
mvn -B archetype:generate \
  -DarchetypeGroupId=software.amazon.awssdk \
  -DarchetypeArtifactId=archetype-lambda -Dservice=s3 -Dregion=US_WEST_2 \
  -DgroupId=com.example.myapp \
  -DartifactId=myapp
```

In PowerShell, esegui questo comando:

```
mvn -B archetype:generate `
  "-DarchetypeGroupId=software.amazon.awssdk" `
  "-DarchetypeArtifactId=archetype-lambda" "-Dservice=s3" "-Dregion=US_WEST_2" `
  "-DgroupId=com.example.myapp" `
  "-DartifactId=myapp"
```

L'archetipo Maven per Lambda è preconfigurato per la compilazione con Java SE 8 e include una dipendenza da AWS SDK per Java. Se crei il tuo progetto con un archetipo diverso o utilizzando un altro metodo, devi [configurare il compilatore Java per Maven](#) e [dichiarare l'SDK come dipendenza](#).

2. Apri la directory `myapp/src/main/java/com/example/myapp` e cerca il file `App.java`. Questo è il codice per la funzione Lambda. A fini di test, puoi utilizzare il codice di esempio fornito o sostituirlo con il tuo codice personalizzato.
3. Torna alla directory principale del progetto e crea un nuovo Dockerfile con la seguente configurazione:
  - Imposta la proprietà FROM sull'[URI dell'immagine di base](#).
  - Imposta l'argomento CMD specificando il gestore della funzione Lambda.

Nota che l'esempio Dockerfile non include un'[istruzione USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione USER.

## Example Dockerfile

```
FROM public.ecr.aws/lambda/java:21

# Copy function code and runtime dependencies from Maven layout
COPY target/classes ${LAMBDA_TASK_ROOT}
COPY target/dependency/* ${LAMBDA_TASK_ROOT}/lib/

# Set the CMD to your handler (could also be done as a parameter override
# outside of the Dockerfile)
CMD [ "com.example.myapp.App::handleRequest" ]
```

4. Compila il progetto e raccogli le dipendenze di runtime.

```
mvn compile dependency:copy-dependencies -DincludeScope=runtime
```

5. Crea l'immagine Docker con il comando [docker build](#). L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il [tag](#) `test`. Per rendere l'immagine compatibile con Lambda, devi usare l'opzione `--provenance=false`.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test .
```

### Note

Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Per creare una funzione Lambda utilizzando l'architettura del set di istruzioni ARM64, accertati di modificare il comando per utilizzare l'opzione `--platform linux/arm64`.

## Gradle

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
cd example
```



2. Esegui il comando seguente per fare in modo che Gradle generi un nuovo progetto di applicazione Java nella directory `example` dell'ambiente. Per Seleziona script di creazione DSL, scegli 2: Groovy.

```
gradle init --type java-application
```

3. Apri la directory `/example/app/src/main/java/example` e cerca il file `App.java`. Questo è il codice per la funzione Lambda. A fini di test, puoi utilizzare il codice di esempio seguente o sostituirlo con il tuo codice personalizzato.

#### Example App.java

```
package com.example;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
public class App implements RequestHandler<Object, String> {
    public String handleRequest(Object input, Context context) {
        return "Hello world!";
    }
}
```

4. Apri il file `build.gradle`. Se stai utilizzando il codice della funzione di esempio del passaggio precedente, sostituisci il contenuto di `build.gradle` con il seguente. Se utilizzi un codice di funzione personalizzato, modifica il file `build.gradle` secondo necessità.

#### Example build.gradle (Groovy DSL)

```
plugins {
    id 'java'
}
group 'com.example'
version '1.0-SNAPSHOT'
sourceCompatibility = 1.8
repositories {
    mavenCentral()
}
dependencies {
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.1'
}
jar {
    manifest {
        attributes 'Main-Class': 'com.example.App'
    }
}
```

```
}  
}
```

5. Il comando `gradle init` del passaggio 2 ha anche generato un caso di test fittizio nella directory `app/test`. Ai fini di questo tutorial, salta l'esecuzione dei test eliminando la directory `/test`.
6. Compilare il progetto.

```
gradle build
```

7. Nella directory root del progetto (`/example`), crea un Dockerfile con la seguente configurazione:

- Imposta la proprietà FROM sull'[URI dell'immagine di base](#).
- Usa il comando COPY per copiare il codice della funzione e le dipendenze di runtime in una `{LAMBDA_TASK_ROOT}` variabile di [Lambda-defined ambiente](#).
- Imposta l'argomento CMD specificando il gestore della funzione Lambda.

Nota che l'esempio Dockerfile non include un'[istruzione USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione USER.

### Example Dockerfile

```
FROM public.ecr.aws/lambda/java:21  
  
# Copy function code and runtime dependencies from Gradle layout  
COPY app/build/classes/java/main ${LAMBDA_TASK_ROOT}  
  
# Set the CMD to your handler (could also be done as a parameter override  
# outside of the Dockerfile)  
CMD [ "com.example.App::handleRequest" ]
```

8. Crea l'immagine Docker con il comando [docker build](#). L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il [tag](#) `test`. Per rendere l'immagine compatibile con Lambda, devi usare l'opzione `--provenance=false`.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test .
```

#### Note

Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Per creare una funzione Lambda utilizzando l'architettura del set di istruzioni ARM64, accertati di modificare il comando per utilizzare l'opzione `--platform linux/arm64`.

### (Facoltativo) Test dell'immagine in locale

1. Avvia l'immagine Docker con il comando `docker run`. In questo esempio, `docker-image` è il nome dell'immagine e `test` è il tag.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

#### Note

Per creare l'immagine Docker per l'architettura del set di istruzioni ARM64, assicurati di utilizzare l'opzione `--platform linux/arm64` anziché `--platform linux/amd64`.

2. Da una nuova finestra di terminale, invia un evento all'endpoint locale.

### Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

## PowerShell

In PowerShell, esegui il seguente Invoke-WebRequest comando:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{} ' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType  
"application/json"
```

### 3. Ottieni l'ID del container.

```
docker ps
```

### 4. Utilizza il comando [docker kill](#) per arrestare il container. In questo comando, sostituisci 3766c4ab331c con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

## Implementazione dell'immagine

### Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

#### 1. Per autenticare la CLI Docker nel registro Amazon ECR, esegui il comando [get-login-password](#).

- Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
- Sostituiscilo 111122223333 con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

 Note

Il repository Amazon ECR deve essere nello stesso della funzione Regione AWS Lambda.

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando [docker tag](#) per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:
  - `docker-image:test` è il nome e [tag](#) dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
  - Sostituisci l'<ECRrepositoryUri> con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando [docker push](#) per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Crea un ruolo di esecuzione](#) per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puoi creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta [Autorizzazioni multiaccount Amazon ECR](#).

## 8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

## 9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nel repository Amazon ECR e quindi utilizzare il comando [update-function-code](#) per implementare l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per implementare la nuova immagine nella stessa funzione Lambda, devi utilizzare il comando [update-function-code](#) anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 11112223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

## Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

Se si utilizza un'immagine di [OS-only base](#) o un'immagine di base alternativa, è necessario includere il client dell'interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime estende l'[API Runtime](#), che gestisce l'interazione tra Lambda e il codice della funzione.

Installa il client di interfaccia di runtime per Java nel tuo Dockerfile o come dipendenza nel tuo progetto. Ad esempio, per installare il client di interfaccia di runtime utilizzando il gestore di pacchetti Maven, aggiungi quanto segue al file `pom.xml`:

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-lambda-java-runtime-interface-client</artifactId>
  <version>2.3.2</version>
</dependency>
```

Per i dettagli del pacchetto, consulta la pagina [Client di interfaccia di runtime AWS Lambda Java](#) nel Maven Central Repository. È inoltre possibile esaminare il codice sorgente del client dell'interfaccia di runtime nel [GitHub repository](#) AWS Lambda Java Support Libraries.

L'esempio seguente dimostra come creare un'immagine di container per Java utilizzando un'[immagine Amazon Corretto](#). Amazon Corretto è una distribuzione pronta per la produzione gratuita, con un ambiente multiplatforma di Open Java Development Kit (OpenJDK). Il progetto Maven include il client di interfaccia di runtime come dipendenza.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Java (ad esempio, [Amazon Corretto](#))
- [Apache Maven](#)
- [AWS CLI versione 2](#)
- [Docker](#) (versione minima 25.0.0)
- Il [plugin buildx](#) di Docker.

## Creazione di un'immagine da un'immagine di base alternativa

1. Crea un progetto Maven. I parametri seguenti sono obbligatori:

- groupId: il namespace completo del pacchetto dell'applicazione.
- artifactId: il nome del progetto. Questo sarà il nome della directory per il progetto.

## Linux/macOS

```
mvn -B archetype:generate \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DgroupId=example \
  -DartifactId=myapp \
```



```
-DinteractiveMode=false
```

## PowerShell

```
mvn -B archetype:generate `
  -DarchetypeArtifactId=maven-archetype-quickstart `
  -DgroupId=example `
  -DartifactId=myapp `
  -DinteractiveMode=false
```

2. Apri la directory del progetto.

```
cd myapp
```

3. Apri il file `pom.xml` e sostituisci il contenuto con quanto riportato di seguito. Questo file include [aws-lambda-java-runtime-interface-client](#) come dipendenza. In alternativa, è possibile installare il client di interfaccia di runtime nel Dockerfile. Tuttavia, l'approccio più semplice consiste nell'includere la libreria come dipendenza.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>example</groupId>
  <artifactId>hello-lambda</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>hello-lambda</name>
  <url>http://maven.apache.org</url>
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-runtime-interface-client</artifactId>
      <version>2.3.2</version>
    </dependency>
  </dependencies>
  <build>
```

```

<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-dependency-plugin</artifactId>
    <version>3.1.2</version>
    <executions>
      <execution>
        <id>copy-dependencies</id>
        <phase>package</phase>
        <goals>
          <goal>copy-dependencies</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
</plugins>
</build>
</project>

```

4. Apri la directory `myapp/src/main/java/com/example/myapp` e cerca il file `App.java`. Questo è il codice per la funzione Lambda. Sostituisci il codice con il seguente.

#### Example gestore della funzione

```

package example;

public class App {
    public static String sayHello() {
        return "Hello world!";
    }
}

```

5. Il comando `mvn -B archetype:generate` del passaggio 1 ha anche generato un test case fittizio nella directory `src/test`. Ai fini di questo tutorial, salta l'esecuzione dei test eliminando la directory `/test` generata interamente.
6. Torna alla directory principale del progetto e crea un nuovo Dockerfile. Il Dockerfile di esempio seguente utilizza un'[immagine Amazon Corretto](#). Amazon Corretto è una distribuzione multi-piattaforma gratuita e pronta per la produzione di OpenJDK.
  - Imposta la proprietà `FROM` sull'URI dell'immagine di base.
  - Imposta l'`ENTRYPOINT` sul modulo su cui desideri che il container Docker venga eseguito all'avvio. In questo caso, il modulo è il client di interfaccia di runtime.

- Imposta l'argomento CMD specificando il gestore della funzione Lambda.

Nota che l'esempio Dockerfile non include un'istruzione [USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente root quando non viene fornita alcuna istruzione USER.

### Example Dockerfile

```
FROM public.ecr.aws/amazoncorretto/amazoncorretto:21 as base

# Configure the build environment
FROM base as build
RUN yum install -y maven
WORKDIR /src

# Cache and copy dependencies
ADD pom.xml .
RUN mvn dependency:go-offline dependency:copy-dependencies

# Compile the function
ADD . .
RUN mvn package

# Copy the function artifact and dependencies onto a clean base
FROM base
WORKDIR /function

COPY --from=build /src/target/dependency/*.jar ./
COPY --from=build /src/target/*.jar ./

# Set runtime interface client as default command for the container runtime
ENTRYPOINT [ "/usr/bin/java", "-cp", ".*",
"com.amazonaws.services.lambda.runtime.api.client.AWSLambda" ]
# Pass the name of the function handler as an argument to the runtime
CMD [ "example.App::sayHello" ]
```

7. Crea l'immagine Docker con il comando [docker build](#). L'esempio seguente assegna un nome all'immagine docker-image e le assegna il [tag](#) test. Per rendere l'immagine compatibile con Lambda, devi usare l'opzione `--provenance=false`.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test .
```

### Note

Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Per creare una funzione Lambda utilizzando l'architettura del set di istruzioni ARM64, accertati di modificare il comando per utilizzare l'opzione `--platform linux/arm64`.

## (Facoltativo) Test dell'immagine in locale

Usa il [simulatore dell'interfaccia di runtime](#) per testare localmente l'immagine. Puoi [creare l'emulatore nella tua immagine](#) o seguire la procedura riportata e installarlo sul tuo computer locale.

## Installazione ed esecuzione dell'emulatore di interfaccia di runtime sul computer locale

1. Dalla directory del progetto, esegui il comando seguente per scaricare l'emulatore di interfaccia di runtime (architettura x86-64) GitHub e installarlo sul tuo computer locale.

### Linux/macOS

```
mkdir -p ~/.aws-lambda-rie && \
  curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-
runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
  chmod +x ~/.aws-lambda-rie/aws-lambda-rie
```

Per installare l'emulatore arm64, sostituite l'URL del GitHub repository nel comando precedente con il seguente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

### PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"
```

```
if (-not (Test-Path $dirPath)) {
    New-Item -Path $dirPath -ItemType Directory
}

$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/
releases/latest/download/aws-lambda-rie"
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Per installare l'emulatore arm64, sostituisci `$downloadLink` con quanto segue:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Avvia l'immagine Docker con il comando `docker run`. Tenere presente quanto segue:

- `docker-image` è il nome dell'immagine e `test` è il tag.
- `/usr/bin/java -cp './*' com.amazonaws.services.lambda.runtime.api.client.AWSLambda example.App::sayHello` è l'ENTRYPOINT seguito dal CMD del Dockerfile.

## Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
9000:8080 \
    --entrypoint /aws-lambda/aws-lambda-rie \
    docker-image:test \
    /usr/bin/java -cp './*'
com.amazonaws.services.lambda.runtime.api.client.AWSLambda
example.App::sayHello
```

## PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
9000:8080 `
--entrypoint /aws-lambda/aws-lambda-rie `
docker-image:test `
    /usr/bin/java -cp './*'
com.amazonaws.services.lambda.runtime.api.client.AWSLambda
example.App::sayHello
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

#### Note

Per creare l'immagine Docker per l'architettura del set di istruzioni ARM64, assicurati di utilizzare l'opzione `--platform linux/arm64` anziché `--platform linux/amd64`.

### 3. Pubblica un evento nell'endpoint locale.

#### Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

#### PowerShell

In PowerShell, esegui il seguente comando: `Invoke-WebRequest`

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType  
"application/json"
```

#### 4. Ottieni l'ID del container.

```
docker ps
```

#### 5. Utilizza il comando [docker kill](#) per arrestare il container. In questo comando, sostituisci 3766c4ab331c con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

### Implementazione dell'immagine

#### Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

#### 1. Per autenticare la CLI Docker nel registro Amazon ECR, esegui il comando [get-login-password](#).

- Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
- Sostituiscilo 111122223333 con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

#### 2. Crea un repository in Amazon ECR utilizzando il comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

#### Note

Il repository Amazon ECR deve essere nello stesso della funzione Regione AWS Lambda.

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{  
  "repository": {
```

```

    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}

```

3. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando [docker tag](#) per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:
  - `docker-image:test` è il nome e [tag](#) dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
  - Sostituisci l'<ECRrepositoryUri> con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando [docker push](#) per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Crea un ruolo di esecuzione](#) per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.



7. Creazione della funzione Lambda Per ImageUri, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puoi creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta [Autorizzazioni multiaccount Amazon ECR](#).

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nel repository Amazon ECR e quindi utilizzare il comando [update-function-code](#) per implementare l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per implementare la nuova immagine nella stessa funzione Lambda, devi utilizzare il comando [update-function-code](#) anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio

seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \  
  --function-name hello-world \  
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --publish
```

## Utilizzo dei livelli per le funzioni Lambda in Java

Usa i [livelli Lambda](#) per impacchettare codice e dipendenze che desideri riutilizzare in più funzioni. I livelli di solito contengono dipendenze dalla libreria, un [runtime personalizzato](#) o file di configurazione. La creazione di un livello prevede tre passaggi generali:

1. Crea un pacchetto per il contenuto del livello. Ciò significa creare un archivio di file con estensione .zip che contiene le dipendenze che desideri utilizzare nelle funzioni.
2. Crea il livello in Lambda.
3. Aggiungi il livello alle tue funzioni.

### Argomenti

- [Crea un pacchetto per il contenuto del livello](#)
- [Crea il layer in Lambda](#)
- [Aggiungete il layer alla vostra funzione](#)

### Crea un pacchetto per il contenuto del livello

Per creare un layer, raggruppa i pacchetti in un archivio di file.zip che soddisfi i seguenti requisiti:

- Assicurati che la versione Java a cui fanno riferimento Maven o Gradle sia la stessa della versione Java della funzione che intendi distribuire. Ad esempio, per una funzione Java 25, il `mvn -v` comando dovrebbe elencare Java 25 nell'output.
- Le dipendenze devono essere archiviate nella `java/lib` directory, alla radice del file.zip. Per ulteriori informazioni, consulta [Percorsi dei livelli per ciascun runtime Lambda](#).
- I pacchetti del livello devono essere compatibili con Linux. Le funzioni Lambda vengono eseguite su Amazon Linux.

Puoi creare livelli che contengono librerie Java di terze parti o moduli e pacchetti Java personalizzati. La procedura seguente utilizza Maven. Puoi anche usare Gradle per impacchettare il contenuto del tuo layer.

### Per creare un livello utilizzando le dipendenze di Maven

1. Crea un progetto Apache Maven con un `pom.xml` file che definisce le tue dipendenze.

L'esempio seguente include [Jackson Databind](#) per l'elaborazione JSON. La <build> sezione utilizza il [maven-dependency-plugin](#) per creare file JAR separati per ogni dipendenza invece di raggrupparli in un unico uber-jar. Se vuoi creare un uber-jar, usa il [maven-shade-plugin](#)

#### Example pom.xml

```
<dependencies>
  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.17.0</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.13.0</version>
      <configuration>
        <source>21</source>
        <target>21</target>
        <release>21</release>
      </configuration>
    </plugin>

    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-dependency-plugin</artifactId>
      <version>3.6.1</version>
      <executions>
        <execution>
          <id>copy-dependencies</id>
          <phase>package</phase>
          <goals>
            <goal>copy-dependencies</goal>
          </goals>
          <configuration>
            <outputDirectory>${project.build.directory}/lib</
outputDirectory>
          </configuration>
```

```
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

2. Compilare il progetto. Questo comando crea tutti i file JAR di dipendenza nella directory. `target/lib/`

```
mvn clean package
```

3. Crea la struttura di directory richiesta per il tuo livello:

```
mkdir -p java/lib
```

4. Copia i file JAR delle dipendenze nella `java/lib` directory:

```
cp target/lib/*.jar java/lib/
```

5. Comprimi il contenuto del livello:

Linux/macOS

```
zip -r layer.zip java/
```

PowerShell

```
Compress-Archive -Path .\java -DestinationPath .\layer.zip
```

La struttura delle cartelle del file.zip dovrebbe essere simile alla seguente:

```
java/
### lib/
### jackson-databind-2.17.0.jar
### jackson-core-2.17.0.jar
### jackson-annotations-2.17.0.jar
```

 Note

Assicurati che il tuo file.zip includa la `java` directory a livello principale al `lib` suo interno. Questa struttura assicura che Lambda possa localizzare e importare le tue librerie. Ogni dipendenza viene conservata come file JAR separato anziché raggruppata in un uber-jar.

## Crea il layer in Lambda

È possibile pubblicare il layer utilizzando la console AWS CLI o la console Lambda.

### AWS CLI

Esegui il [publish-layer-version](#) AWS CLI comando per creare il livello Lambda:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip  
--compatible-runtimes java25
```

Il parametro di [runtime compatibile](#) è facoltativo. Quando specificato, Lambda utilizza questo parametro per filtrare i livelli nella console Lambda.

### Console

#### Creazione di un livello (console)

1. Apri la [pagina Layers](#) (Livelli) nella console Lambda.
2. Scegli Create layer (Crea livello).
3. Scegli Carica un file.zip, quindi carica l'archivio.zip che hai creato in precedenza.
4. (Facoltativo) Per i runtime compatibili, scegliete il runtime Java che corrisponde alla versione Java utilizzata per creare il layer.
5. Scegli Create (Crea).

## Aggiungete il layer alla vostra funzione

### AWS CLI

Per collegare il layer alla tua funzione, esegui il [update-function-configuration](#) AWS CLI comando. Per il `--layers` parametro, utilizzate il livello ARN. L'ARN deve specificare la versione (ad

esempio, `arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`). Per ulteriori informazioni, consulta [Livelli e versioni di livelli](#).

```
aws lambda update-function-configuration --function-name my-function --cli-binary-format raw-in-base64-out --layers "arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1"
```

L'opzione `--cli-binary-format raw-in-base64-out` è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

## Console

### Aggiunta di un livello a una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione.
3. Scorri verso il basso fino alla sezione Livelli, quindi scegli Aggiungi un livello.
4. In Scegli un livello, seleziona Livelli personalizzati, quindi scegli il tuo livello.

#### Note

Se non hai aggiunto un [runtime compatibile](#) quando hai creato il layer, il tuo layer non verrà elencato qui. È possibile specificare invece il livello ARN.

5. Scegliere Aggiungi.

## Personalizzare la serializzazione per le funzioni Lambda Java

I [runtime gestiti da Java](#) di Lambda supportano la serializzazione personalizzata per gli eventi JSON. La serializzazione personalizzata può semplificare il codice e potenzialmente migliorare le prestazioni.

### Argomenti

- [Quando usare la serializzazione personalizzata](#)
- [Implementazione della serializzazione personalizzata](#)
- [Test della serializzazione personalizzata](#)

### Quando usare la serializzazione personalizzata

Quando viene richiamata la funzione Lambda, i dati degli eventi di input devono essere deserializzati in un oggetto Java e l'output della funzione deve essere nuovamente serializzato in un formato che può essere restituito come risposta della funzione. I runtime gestiti Lambda Java forniscono funzionalità di serializzazione e deserializzazione predefinite che funzionano bene per la gestione dei payload degli eventi da vari AWS servizi, come Amazon API Gateway e Amazon Simple Queue Service (Amazon SQS). Per utilizzare questi eventi di integrazione dei servizi nella tua funzione, aggiungi la dipendenza [aws-java-lambda-events](#) al tuo progetto. Questa AWS libreria contiene oggetti Java che rappresentano questi eventi di integrazione dei servizi.

Puoi anche usare i tuoi oggetti per rappresentare l'evento JSON che passi alla tua funzione Lambda. Il runtime gestito prova a serializzare il JSON su una nuova istanza dell'oggetto con il suo comportamento predefinito. Se il serializzatore predefinito non ha il comportamento desiderato per il tuo caso d'uso, usa la serializzazione personalizzata.

Supponiamo, ad esempio, che l'handler di funzioni si aspetti una classe `Vehicle` come input, con la seguente struttura:

```
public class Vehicle {  
    private String vehicleType;  
    private long vehicleId;  
}
```

Tuttavia, il payload dell'evento JSON ha il seguente aspetto:

```
{
```



```
"vehicle-type": "car",  
"vehicleID": 123  
}
```

In questo scenario, la serializzazione predefinita nel runtime gestito prevede che i nomi delle proprietà JSON corrispondano ai nomi delle proprietà della classe Java con notazione a cammello (`vehicleType`, `vehicleId`). Poiché i nomi delle proprietà nell'evento JSON non sono in notazione a cammello (`vehicle-type`, `vehicleID`), è necessario utilizzare la serializzazione personalizzata.

### Implementazione della serializzazione personalizzata

Utilizza un'interfaccia [del fornitore di servizi](#) per caricare un serializzatore a tua scelta anziché la logica di serializzazione predefinita del runtime gestito. È possibile serializzare i payload di eventi JSON direttamente in oggetti Java, utilizzando l'interfaccia `RequestHandler` standard.

Per utilizzare la serializzazione personalizzata nella funzione Java per Lambda

1. Aggiungi la libreria [aws-lambda-java-core](#) come dipendenza. Questa libreria include l'[CustomPojoSerializer](#) interfaccia, insieme ad altre definizioni di interfaccia per lavorare con Java in Lambda.
2. Crea un file denominato `com.amazonaws.services.lambda.runtime.CustomPojoSerializer` nella directory `src/main/resources/META-INF/services/` del progetto.
3. In questo file, specifica il nome completo dell'implementazione del serializzatore personalizzato, che deve implementare l'interfaccia `CustomPojoSerializer`. Esempio:

```
com.mycompany.vehicles.CustomLambdaSerializer
```

4. Implementa l'interfaccia `CustomPojoSerializer` per fornire la tua logica di serializzazione personalizzata.
5. Usa l'interfaccia `RequestHandler` standard nella tua funzione Lambda. Il runtime gestito utilizzerà il serializzatore personalizzato.

Per altri esempi su come implementare la serializzazione personalizzata utilizzando librerie popolari come FastJSON, Gson, Moshi e jackson-jr, vedi l'esempio di serializzazione personalizzata nel repository. <https://github.com/aws/aws-lambda-java-libs/tree/main/samples/custom-serialization> AWS GitHub

## Test della serializzazione personalizzata

Verifica la tua funzione per assicurarti che la logica di serializzazione e deserializzazione funzioni come previsto. Puoi usare l'interfaccia a riga di AWS Serverless Application Model comando () per emulare l'invocazione del tuo payload Lambda. AWS SAMCLI Questo può aiutarti a testare e iterare rapidamente la tua funzione man mano che introduci un serializzatore personalizzato.

1. Crea un file con il payload dell'evento JSON con cui desideri richiamare la tua funzione, quindi chiama il. AWS SAMCLI
2. Esegui il comando [sam local invoke](#) per richiamare la tua funzione in locale. Esempio:

```
sam local invoke -e src/test/resources/event.json
```

Per ulteriori informazioni, consulta Invocare [localmente le funzioni Lambda con. AWS SAM](#)

## Personalizzare il comportamento di avvio del runtime Java per le funzioni Lambda

Questa pagina descrive le impostazioni specifiche delle funzioni Java in AWS Lambda. È possibile utilizzare queste impostazioni per personalizzare il comportamento di avvio del runtime Java. Ciò può ridurre la latenza complessiva della funzione e migliorare le prestazioni complessive delle funzioni senza dover modificare il codice.

### Sections

- [Comprendere la variabile di ambiente JAVA\\_TOOL\\_OPTIONS](#)
- [Patch Log4j per Log4Shell](#)
- [Ahead-of-Time \(AOT\) e cache CDS](#)

### Comprendere la variabile di ambiente JAVA\_TOOL\_OPTIONS

In Java, Lambda supporta la variabile di ambiente `JAVA_TOOL_OPTIONS` per impostare variabili della linea di comando aggiuntive in Lambda. È possibile utilizzare questa variabile di ambiente in vari modi, ad esempio per personalizzare le impostazioni di compilazione a più livelli. L'esempio seguente illustra come utilizzare la variabile di ambiente `JAVA_TOOL_OPTIONS` per questo caso d'uso.

Esempio: personalizzazione delle impostazioni di compilazione a più livelli

La compilazione a livelli è una funzionalità della macchina virtuale Java (JVM). È possibile utilizzare impostazioni di compilazione a più livelli specifiche per sfruttare al meglio i compilatori just-in-time (JIT) della JVM. In genere, il compilatore C1 è ottimizzato per tempi di avvio rapidi. Il compilatore C2 è ottimizzato per le migliori prestazioni complessive, ma di contro utilizza più memoria e impiega più tempo per raggiungerle.

Esistono 5 diversi livelli di compilazione a più livelli. Al livello 0, la JVM interpreta il bytecode Java. Al livello 4, la JVM utilizza il compilatore C2 per analizzare i dati di profilazione raccolti durante l'avvio dell'applicazione. Nel tempo, monitora l'utilizzo del codice per identificare le migliori ottimizzazioni.

La personalizzazione del livello di compilazione a più livelli può aiutarti a ottimizzare le prestazioni delle funzioni Java. Per le piccole funzioni che vengono eseguite rapidamente, l'impostazione della compilazione a più livelli al livello 1 può aiutare a migliorare le prestazioni di avvio a freddo facendo in modo che la JVM utilizzi il compilatore C1. Questa impostazione produce rapidamente codice nativo ottimizzato ma non genera dati di profilazione e non utilizza mai il compilatore C2. Per funzioni più ampie e ad alta intensità di calcolo, l'impostazione della compilazione su più livelli al livello 4 massimizza le prestazioni complessive a scapito del consumo aggiuntivo di memoria e dell'ulteriore

lavoro di ottimizzazione durante le prime chiamate dopo il provisioning di ciascun ambiente di esecuzione Lambda.

Per i runtime di Java 11 e versioni precedenti, Lambda utilizza le impostazioni di compilazione a più livelli JVM predefinite. Per Java 17 e Java 21, Lambda configura la JVM per interrompere la compilazione a più livelli al livello 1 per impostazione predefinita. A partire da Java 25, Lambda interrompe ancora la compilazione a più livelli al livello 1 per impostazione predefinita, tranne quando si utilizza la concorrenza SnapStart o Provisioned, nel qual caso vengono utilizzate le impostazioni JVM predefinite.

Ciò migliora le prestazioni SnapStart e la concorrenza con provisioning senza incorrere in una penalità di avvio a freddo, poiché in questi casi la compilazione a più livelli viene eseguita al di fuori del percorso di richiamo. Per massimizzare questo vantaggio, è possibile utilizzare il priming, ossia l'esecuzione di percorsi di codice durante l'inizializzazione della funzione, per attivare JIT prima di scattare l'istantanea o quando gli ambienti di esecuzione Provisioned Concurrency sono predisposti. SnapStart Per ulteriori informazioni, consulta il post del blog [Ottimizzazione delle prestazioni di avvio a freddo di Lambda utilizzando strategie di priming avanzate con AWS SnapStart](#)

Personalizzazione delle impostazioni di compilazione a più livelli (console)

1. Apri la [pagina Funzioni](#) della console Lambda.
2. Scegli una funzione Java per la quale desideri personalizzare la compilazione a più livelli.
3. Scegli la scheda Configurazione, quindi scegli Variabili di ambiente nel menu a sinistra.
4. Scegli Modifica.
5. Scegli Add environment variable (Aggiungi variabile d'ambiente).
6. Per la chiave, inserisci `JAVA_TOOL_OPTIONS`. Per il valore, inserisci `-XX:+TieredCompilation -XX:TieredStopAtLevel=1`.

## Edit environment variables

### Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#) 

Key

JAVA\_TOOL\_OPTIONS

Value

-XX:+TieredCompilation -XX:TieredStopAtNode=1

Remove

Add environment variable

► Encryption configuration

Cancel

Save

7. Scegli Save (Salva).

### Note

Puoi anche usare Lambda SnapStart per mitigare i problemi di avvio a freddo. SnapStart utilizza istantanee memorizzate nella cache dell'ambiente di esecuzione per migliorare significativamente le prestazioni di avvio. Per ulteriori informazioni su SnapStart funzionalità, limitazioni e regioni supportate, vedere. [Miglioramento delle prestazioni di avvio con Lambda SnapStart](#)

Esempio: personalizzazione del comportamento del GC utilizzando JAVA\_TOOL\_OPTIONS

I runtime di Java 11 utilizzano [Serial](#) Garbage Collector (GC) per la rimozione di oggetti inutili (garbage collection). Per impostazione predefinita, anche i runtime di Java 17 utilizzano Serial GC. Tuttavia, con Java 17 puoi anche utilizzare la variabile di ambiente JAVA\_TOOL\_OPTIONS per modificare il GC predefinito. Puoi scegliere tra Parallel GC e [Shenandoah GC](#).

Ad esempio, se il carico di lavoro utilizza più memoria e più CPU, prendi in considerazione l'utilizzo di Parallel GC per ottenere prestazioni migliori. Puoi farlo aggiungendo quanto segue al valore della tua variabile di ambiente `JAVA_TOOL_OPTIONS`:

```
-XX:+UseParallelGC
```

Se il carico di lavoro ha molti oggetti di breve durata, è possibile trarre vantaggio da un minore consumo di memoria abilitando la modalità generazionale del garbage collector di Shenandoah introdotto in Java 25. Per fare ciò, aggiungete quanto segue al valore della variabile di ambiente: `JAVA_TOOL_OPTIONS`

```
-XX:+UseShenandoahGC -XX:ShenandoahGCMode=generational
```

### Patch Log4j per Log4Shell

I runtime Lambda per Java 8, 11, 17 e 21 includono una patch per mitigare la vulnerabilità Log4Shell () CVE-2021-44228 in Log4j, un popolare framework di registrazione Java. Questa patch comporta un sovraccarico delle prestazioni di avvio a freddo. Se stai utilizzando una versione patchata di Log4j (versione 2.17.0 o successiva), puoi disabilitare questa patch per migliorare le prestazioni di avvio a freddo. Per disattivare la patch, imposta la variabile di ambiente su `AWS_LAMBDA_DISABLE_CVE_2021_44228_PROTECTION true`

A partire da Java 25, i runtime Lambda non includono più la patch Log4Shell. È necessario verificare di utilizzare Log4j versione 2.17.0 o successiva.

### Ahead-of-Time (AOT) e cache CDS

A partire da Java 25, il runtime Lambda include una cache Ahead-of-Time (AOT) per il Java runtime interface client (RIC), un componente di runtime che esegue attivamente il polling degli eventi dall'API Lambda Runtime. Ciò migliora le prestazioni di avvio a freddo.

Le cache AOT sono specifiche per una build JVM. Quando Lambda aggiorna il runtime gestito, aggiorna anche la cache AOT per il RIC. Tuttavia, se distribuisce le tue cache AOT, queste potrebbero essere invalidate o provocare un comportamento imprevisto a seguito di un aggiornamento del runtime. Pertanto, consigliamo vivamente di non utilizzare le cache AOT quando si utilizzano runtime gestiti. Per utilizzare le cache AOT, è necessario distribuire le funzioni utilizzando immagini di contenitori.

Le cache AOT non possono essere utilizzate con le cache Class Data Sharing (CDS). Se distribuisce cache CDS nella tua funzione Lambda, Lambda disabilita la cache AOT.

## Utilizzo dell'oggetto di contesto Lambda per recuperare le informazioni sulla funzione Java

Quando Lambda esegue la funzione, passa un oggetto Context al [gestore](#). Questo oggetto fornisce i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

### Metodi del contesto

- `getRemainingTimeInMillis()`: restituisce il numero di millisecondi rimasti prima del timeout dell'esecuzione.
- `getFunctionName()`: restituisce il nome della funzione Lambda.
- `getFunctionVersion()`: restituisce la [versione](#) della funzione.
- `getInvokedFunctionArn()`: restituisce l'ARN (Amazon Resource Name) utilizzato per invocare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `getMemoryLimitInMB()`: restituisce la quantità di memoria allocata per la funzione.
- `getAwsRequestId()`: restituisce l'identificatore della richiesta di invocazione.
- `getLogGroupName()`: restituisce il gruppo di log per la funzione.
- `getLogStreamName()`: restituisce il flusso di log per l'istanza della funzione.
- `getIdentity()`: (app per dispositivi mobili) restituisce informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
- `getClientContext()`: (app per dispositivi mobili) restituisce il contesto client fornito a Lambda dall'applicazione client.
- `getLogger()`: restituisce l'[oggetto logger](#) per la funzione.

Nell'esempio seguente viene illustrata una funzione che utilizza l'oggetto contesto per accedere al logger Lambda.

### Example [Handler.java](#)

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import java.util.Map;
```

```
// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String,String>, Void>{

    @Override
    public Void handleRequest(Map<String,String> event, Context context)
    {
        LambdaLogger logger = context.getLogger();
        logger.log("EVENT TYPE: " + event.getClass());
        return null;
    }
}
```

La funzione registra il tipo di classe dell'evento in entrata prima di restituire `null`.

#### Example Output log

```
EVENT TYPE: class java.util.LinkedHashMap
```

L'interfaccia per l'oggetto contesto è disponibile nella libreria [aws-lambda-java-core](#). È possibile implementare questa interfaccia per creare una classe di contesto per il test. L'esempio seguente mostra una classe di contesto che restituisce valori fittizi per la maggior parte delle proprietà e un logger di test funzionante.

Example [src/test/java/example/TestContext.java](#)

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.CognitoIdentity;
import com.amazonaws.services.lambda.runtime.ClientContext;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

public class TestContext implements Context{

    public TestContext() {}
    public String getAwsRequestId(){
        return new String("495b12a8-xmpl-4eca-8168-160484189f99");
    }
    public String getLogGroupName(){
        return new String("/aws/lambda/my-function");
    }
    public String getLogStreamName(){
```



```
    return new String("2020/02/26/[$LATEST]704f8dxmlpla04097b9134246b8438f1a");
}
public String getFunctionName(){
    return new String("my-function");
}
public String getFunctionVersion(){
    return new String("$LATEST");
}
public String getInvokedFunctionArn(){
    return new String("arn:aws:lambda:us-east-2:123456789012:function:my-function");
}
public CognitoIdentity getIdentity(){
    return null;
}
public ClientContext getClientContext(){
    return null;
}
public int getRemainingTimeInMillis(){
    return 300000;
}
public int getMemoryLimitInMB(){
    return 512;
}
public LambdaLogger getLogger(){
    return new TestLogger();
}
}
```

Per ulteriori informazioni sulla registrazione, consulta [Registrazione e monitoraggio funzioni Lambda in Java](#).

## Contesto nelle applicazioni di esempio

Il repository GitHub per questa guida include applicazioni di esempio che illustrano l'uso dell'oggetto contesto. Ogni applicazione di esempio include script per semplificare la distribuzione e la pulizia, un modello AWS Serverless Application Model (AWS SAM) e risorse di supporto.

## Applicazioni Lambda di esempio in Java

- [example-java](#) — Una funzione Java che dimostra come utilizzare Lambda per elaborare gli ordini. Questa funzione illustra come definire e deserializzare un oggetto evento di input personalizzato, utilizzare l'SDK e registrare l'output. AWS

- [java-basic](#): una raccolta di funzioni Java minimali con unit test e configurazione della registrazione dei log delle variabili.
- [java-events](#): una raccolta di funzioni Java che contengono codice skeleton per la gestione degli eventi di vari servizi, ad esempio Gateway Amazon API, Amazon SQS e Amazon Kinesis. Queste funzioni utilizzano la versione più recente della libreria [aws-lambda-java-events](#) (3.0.0 e versioni successive). Questi esempi non richiedono SDK AWS come dipendenza.
- [s3-java](#) – Una funzione Java che elabora gli eventi di notifica da Amazon S3 e utilizza la Java Class Library (JCL) per creare anteprime dai file di immagine caricati.
- [layer-java](#) — Una funzione Java che illustra come utilizzare un livello Lambda per impacchettare dipendenze separate dal codice della funzione principale.

## Registrare e monitorare funzioni Lambda in Java

AWS Lambda monitora automaticamente le funzioni Lambda e invia le voci di registro ad Amazon. CloudWatch La tua funzione Lambda include un gruppo di CloudWatch log Logs e un flusso di log per ogni istanza della tua funzione. L'ambiente di runtime di Lambda invia al flusso di log i dettagli su ogni invocazione e altri output dal codice della funzione. Per ulteriori informazioni sui CloudWatch log, consulta [Invio dei log delle funzioni Lambda ai CloudWatch log](#).

Per generare output di log dal codice della funzione, puoi utilizzare i metodi di [java.lang.System](#) o qualsiasi modulo di registrazione che scriva in stdout o stderr.

### Sections

- [Creazione di una funzione che restituisce i registri](#)
- [Utilizzo dei controlli di registrazione avanzati di Lambda con Java](#)
- [Implementazione della registrazione avanzata con Log4j2 e SLF4J](#)
- [Utilizzo di altri strumenti di registrazione e librerie](#)
- [Utilizzo di Powertools per AWS Lambda \(Java\) e AWS SAM per la registrazione strutturata](#)
- [Visualizzazione dei log nella console Lambda](#)
- [Visualizzazione dei log nella console CloudWatch](#)
- [Visualizzazione dei log utilizzando AWS Command Line Interface \(AWS CLI\)](#)
- [Eliminazione dei log](#)
- [Codice di registrazione dei log di esempio](#)

### Creazione di una funzione che restituisce i registri

Per i log di output del codice della funzione, puoi usare i metodi di [java.lang.System](#) o qualsiasi modulo di registrazione che scriva in stdout o stderr. La libreria [aws-lambda-java-core](#) fornisce una classe di logger denominata `LambdaLogger` a cui è possibile accedere dall'oggetto contestuale. La classe di logger supporta i log multilinea.

Nell'esempio seguente viene utilizzato il logger `LambdaLogger` fornito dall'oggetto contestuale.

### Example Handler.java

```
// Handler value: example.Handler
public class Handler implements RequestHandler<Object, String>{
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
```

```

@Override
public String handleRequest(Object event, Context context)
{
    LambdaLogger logger = context.getLogger();
    String response = new String("SUCCESS");
    // log execution details
    logger.log("ENVIRONMENT VARIABLES: " + gson.toJson(System.getenv()));
    logger.log("CONTEXT: " + gson.toJson(context));
    // process event
    logger.log("EVENT: " + gson.toJson(event));
    return response;
}
}

```

## Example Formato dei log

```

START RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Version: $LATEST
ENVIRONMENT VARIABLES:
{
    "_HANDLER": "example.Handler",
    "AWS_EXECUTION_ENV": "AWS_Lambda_java8",
    "AWS_LAMBDA_FUNCTION_MEMORY_SIZE": "512",
    ...
}
CONTEXT:
{
    "memoryLimit": 512,
    "awsRequestId": "6bc28136-xmpl-4365-b021-0ce6b2e64ab0",
    "functionName": "java-console",
    ...
}
EVENT:
{
    "records": [
        {
            "messageId": "19dd0b57-xmpl-4ac1-bd88-01bbb068cb78",
            "receiptHandle": "MessageReceiptHandle",
            "body": "Hello from SQS!",
            ...
        }
    ]
}
END RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0

```

```
REPORT RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Duration: 198.50 ms Billed
Duration: 724 ms Memory Size: 512 MB Max Memory Used: 90 MB Init Duration: 524.75 ms
```

Il runtime di Java registra START, END e REPORT per ogni chiamata. La riga del report fornisce i seguenti dettagli:

### Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per la chiamata.
- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.
- Quantità max utilizzata – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- Durata Init – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- XRAY TraceId: per le richieste tracciate, l'ID di traccia. [AWS X-Ray](#)
- SegmentId— Per le richieste tracciate, l'ID del X-Ray segmento.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

### Utilizzo dei controlli di registrazione avanzati di Lambda con Java

Per darti un maggiore controllo su come i log delle tue funzioni vengono acquisiti, elaborati e consumati, puoi configurare le seguenti opzioni di registrazione per i runtime Java supportati:

- Formato di registro: seleziona tra testo normale e formato JSON strutturato per i log della tua funzione
- Livello di registro: per i log in formato JSON, scegli il livello di dettaglio dei log a cui Lambda invia CloudWatch, ad esempio ERROR, DEBUG o INFO
- Gruppo di log: scegli il gruppo di CloudWatch log a cui la tua funzione invia i log

Per ulteriori informazioni su queste opzioni di registrazione e istruzioni su come configurare la funzione per utilizzarle, consulta la pagina [the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”](#).

Per utilizzare le opzioni del formato di log e del livello di log con le funzioni Lambda in Java, consulta le istruzioni nelle sezioni seguenti.

### Utilizzo del formato di log JSON strutturato con Java

Se selezioni JSON come formato di registro della tua funzione, Lambda invia l'output dei log utilizzando la `LambdaLogger` classe come JSON strutturato. Ogni oggetto di log JSON contiene almeno quattro coppie chiave-valore con le seguenti chiavi:

- `"timestamp"`: l'ora in cui è stato generato il messaggio di log
- `"level"`: il livello di log assegnato al messaggio
- `"message"`: il contenuto del messaggio di log
- `"AWSrequestId"`: l'ID di richiesta univoco dell'invocazione alla funzione

A seconda del metodo di registrazione di log utilizzato, gli output di log della funzione acquisiti in formato JSON possono contenere anche coppie di chiave-valore aggiuntive.

Per assegnare un livello ai log creati utilizzando il logger `LambdaLogger`, è necessario fornire un argomento `LogLevel` nel comando di registrazione, come mostrato nell'esempio seguente.

### Example Codice di registrazione di Java

```
LambdaLogger logger = context.getLogger();  
logger.log("This is a debug log", LogLevel.DEBUG);
```

L'output del log ottenuto da questo codice di esempio verrà acquisito in CloudWatch Logs come segue:

### Example Record di log JSON

```
{  
  "timestamp": "2023-11-01T00:21:51.358Z",  
  "level": "DEBUG",  
  "message": "This is a debug log",  
  "AWSrequestId": "93f25699-2cbf-4976-8f94-336a0aa98c6f"  
}
```

Se non assegni un livello all'output del log, Lambda gli assegna automaticamente il livello INFO.

Se il codice utilizza già un'altra libreria di registrazione per produrre log strutturati JSON, non è necessario apportare alcuna modifica. Lambda non codifica due volte i log già codificati in JSON. Anche se configuri la funzione per utilizzare il formato di log JSON, i tuoi output di registrazione vengono visualizzati nella struttura JSON definita. CloudWatch

## Utilizzo del filtraggio a livello di log con Java

Per AWS Lambda filtrare i log dell'applicazione in base al relativo livello di registro, la funzione deve utilizzare i log in formato JSON. Puoi farlo in due modi:

- Crea output log utilizzando il `LambdaLogger` standard e configura la tua funzione per utilizzare la formattazione dei log JSON. Lambda filtra quindi gli output del log utilizzando la coppia chiave-valore «level» nell'oggetto JSON descritto in [the section called “Utilizzo del formato di log JSON strutturato con Java”](#). Per sapere come configurare il formato di log della tua funzione, consulta [the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”](#)
- Utilizzate un'altra libreria o metodo di registrazione per creare log strutturati JSON nel codice che includono una coppia chiave-valore «level» che definisce il livello dell'output del log. Puoi utilizzare qualunque libreria di registrazione che sia in grado di scrivere log JSON in `stdout` o `stderr`. Ad esempio, puoi utilizzare Powertools for AWS Lambda o il pacchetto Log4j2 per generare output di log strutturati JSON dal tuo codice. Per ulteriori informazioni, consulta le pagine [the section called “Utilizzo di Powertools per AWS Lambda \(Java\) e AWS SAM per la registrazione strutturata”](#) e [the section called “Implementazione della registrazione avanzata con Log4j2 e SLF4J”](#).

Quando configuri la tua funzione per utilizzare il filtro a livello di log, devi selezionare una delle seguenti opzioni per il livello di log che desideri che Lambda invii a Logs: CloudWatch

| Livello di log            | Utilizzo standard                                                                             |
|---------------------------|-----------------------------------------------------------------------------------------------|
| TRACE (dettaglio massimo) | Le informazioni più dettagliate utilizzate per tracciare il percorso di esecuzione del codice |
| DEBUG                     | Informazioni dettagliate per il debug del sistema                                             |
| INFO                      | Messaggi che registrano il normale funzionamento della funzione                               |

| Livello di log           | Utilizzo standard                                                                                   |
|--------------------------|-----------------------------------------------------------------------------------------------------|
| WARN                     | Messaggi relativi a potenziali errori che possono portare a comportamenti imprevisti se non risolti |
| ERRORE                   | Messaggi relativi a problemi che impediscono al codice di funzionare come previsto                  |
| FATAL (dettaglio minimo) | Messaggi relativi a errori gravi che causano l'interruzione del funzionamento dell'applicazione     |

Per consentire a Lambda di filtrare i log della funzione, è necessario includere anche una coppia chiave-valore "timestamp" nell'output log JSON. L'ora deve essere specificata in un formato di timestamp [RFC 3339](#) valido. Se non fornisci un timestamp valido, Lambda assegna al log il livello INFO e aggiunge un timestamp per te.

Lambda invia i log del livello selezionato e inferiore a. CloudWatch Ad esempio, se si configura un livello di registro WARN, Lambda invia i log corrispondenti ai livelli WARN, ERROR e FATAL.

Implementazione della registrazione avanzata con Log4j2 e SLF4J

#### Note

AWS Lambda non include Log4j2 nei runtime gestiti o nelle immagini dei contenitori di base. Questi non sono quindi interessati dai problemi descritti in CVE-2021-44228, e. CVE-2021-45046 CVE-2021-45105

Nei casi in cui una funzione del cliente include una versione di Log4j2 interessata, abbiamo applicato una modifica ai runtime [gestiti da Lambda Java](#) e alle immagini dei contenitori di [base](#) che aiuta a mitigare i problemi relativi a, e. CVE-2021-44228 CVE-2021-45046 CVE-2021-45105 Come risultato di questa modifica, i clienti che utilizzano Log4J2 potrebbero vedere una voce di log aggiuntiva, simile a "Transforming org/apache/logging/log4j/core/lookup/JndiLookup (java.net.URLClassLoader@...)". Tutte le stringhe di log che fanno riferimento al mapper jndi nell'output di Log4J2 vengono sostituite da "». Patched JndiLookup::lookup()

Indipendentemente da questa modifica, incoraggiamo fortemente tutti i clienti le cui funzioni includono Log4j2 ad aggiornare l'ultima versione. In particolare, i clienti che utilizzano la



libreria `aws-lambda-java-log4j2` devono eseguire l'aggiornamento alla versione 1.5.0 (o successiva) e ridistribuire le funzioni. Questa versione aggiorna le dipendenze dell'utilità `Log4j2` sottostanti alla versione 2.17.0 (o successiva). Il file binario `aws-lambda-java-log4j2` è disponibile nel [repository Maven](#) e il codice sorgente è disponibile in [Github](#).

Infine, tieni presente che tutte le librerie relative ad `aws-lambda-java-log4j` (v1.0.0 o 1.0.1) non devono essere utilizzate in nessuna circostanza. Queste librerie sono correlate alla versione 1.x di `log4j`, che ha raggiunto il fine vita nel 2015. Le librerie non sono supportate, non sono gestite, non sono corredate di patch e presentano vulnerabilità di sicurezza note.

Per personalizzare l'output del registro, supportare la registrazione durante i test unitari e registrare le chiamate AWS SDK, usate Apache `Log4j2` con `SLF4J`. `Log4j` è una libreria di registrazione per programmi Java che puoi utilizzare per configurare i livelli di registro e utilizzare le librerie di appender. `SLF4J` è una libreria di facciata che puoi usare per cambiare la libreria che usi senza cambiare il codice funzione.

Per aggiungere l'ID richiesta ai log della funzione, usa l'appender nella libreria [aws-lambda-java-log4j2](#).

Example [src/main/resources/log4j2.xml](#) — Configurazione dell'appendice

```
<Configuration>
  <Appenders>
    <Lambda name="Lambda" format="{env:AWS_LAMBDA_LOG_FORMAT:-TEXT}">
      <LambdaTextFormat>
        <PatternLayout>
          <pattern>%d{yyyy-MM-dd HH:mm:ss} %X{AWSRequestId} %-5p %c{1} - %m%n </
pattern>
        </PatternLayout>
      </LambdaTextFormat>
      <LambdaJSONFormat>
        <JsonTemplateLayout eventTemplateUri="classpath:LambdaLayout.json" />
      </LambdaJSONFormat>
    </Lambda>
  </Appenders>
  <Loggers>
    <Root level="{env:AWS_LAMBDA_LOG_LEVEL:-INFO}">
      <AppenderRef ref="Lambda"/>
    </Root>
    <Logger name="software.amazon.awssdk" level="WARN" />
    <Logger name="software.amazon.awssdk.request" level="DEBUG" />
  </Loggers>
</Configuration>
```

```
</Loggers>
</Configuration>
```

Puoi decidere di configurare gli output dei log di Log4j2 in testo normale o JSON specificando un layout sotto i tag `<LambdaTextFormat>` e `<LambdaJSONFormat>`.

Con questa configurazione, in modalità di testo, ogni riga è preceduta da data, ora, ID richiesta, livello di log e nome della classe. In modalità JSON, viene utilizzato `<JsonTemplateLayout>` con una configurazione fornita insieme alla libreria `aws-lambda-java-log4j2`.

SLF4J è una libreria di tipo Facade per la registrazione in codice Java. Nel codice funzione, si utilizza la factory del logger SLF4J per recuperare un logger con metodi per i livelli di log quali `info()` e `warn()`. Nella configurazione di compilazione, includere la libreria di registrazione e l'adattatore SLF4J nel classpath. Modificando le librerie nella configurazione di compilazione, è possibile modificare il tipo di logger senza modificare il codice della funzione. SLF4J è richiesto per acquisire i log da SDK per Java.

Nell'esempio di codice seguente, la classe del gestore utilizza SLF4J per recuperare un logger.

Example [src/main/java/example/HandlerS3.java](#) — Registrazione con SLF4J

```
package example;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;

import static org.apache.logging.log4j.CloseableThreadContext.put;

public class HandlerS3 implements RequestHandler<S3Event, String>{
    private static final Logger logger = LoggerFactory.getLogger(HandlerS3.class);

    @Override
    public String handleRequest(S3Event event, Context context) {
        for(var record : event.getRecords()) {
            try (var loggingCtx = put("awsRegion", record.getAwsRegion())) {
                loggingCtx.put("eventName", record.getEventName());
                loggingCtx.put("bucket", record.getS3().getBucket().getName());
```

```
        loggingCtx.put("key", record.getS3().getObject().getKey());

        logger.info("Handling s3 event");
    }
}

return "Ok";
}
```

Questo codice genera output log simili al seguente:

### Example Formato dei log

```
{
  "timestamp": "2023-11-15T16:56:00.815Z",
  "level": "INFO",
  "message": "Handling s3 event",
  "logger": "example.HandlerS3",
  "AWSRequestId": "0bcd576-3936-4e5a-9dcd-db9477b77f97",
  "awsRegion": "eu-south-1",
  "bucket": "java-logging-test-input-bucket",
  "eventName": "ObjectCreated:Put",
  "key": "test-folder/"
}
```

La configurazione di compilazione prende le dipendenze di runtime sull'appender Lambda e sull'adattatore SLF4J e le dipendenze di implementazione su Log4j2.

### Example build.gradle – Registrazione delle dipendenze

```
dependencies {
    ...
    'com.amazonaws:aws-lambda-java-log4j2:[1.6.0,)',
    'com.amazonaws:aws-lambda-java-events:[3.11.3,)',
    'org.apache.logging.log4j:log4j-layout-template-json:[2.17.1,)',
    'org.apache.logging.log4j:log4j-slf4j2-impl:[2.19.0,)',
    ...
}
```

Quando esegui localmente il codice per i test, l'oggetto contestuale con il logger Lambda non è disponibile e non esiste alcun ID richiesta che possa essere utilizzato dall'appendere Lambda. Per configurazioni di test di esempio, consulta le applicazioni di esempio nella sezione successiva.

## Utilizzo di altri strumenti di registrazione e librerie

[Powertools for AWS Lambda \(Java\)](#) è un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori. L'[utilità di registrazione](#) fornisce un logger ottimizzato per Lambda che include informazioni aggiuntive sul contesto delle funzioni in tutte le funzioni con output strutturato come JSON. Utilizza l'utility per eseguire le seguenti operazioni:

- Acquisizione di campi essenziali dal contesto Lambda, avvii a freddo e output di registrazione della struttura come JSON
- Registrazione degli eventi di chiamata Lambda quando richiesto (disabilitata per impostazione predefinita)
- Stampa di tutti i log solo per una percentuale di chiamate tramite campionamento dei log (disabilitata per impostazione predefinita)
- Aggiunta di chiavi supplementari al log strutturato in qualsiasi momento
- Utilizzo di un formattatore di log personalizzato (Bring Your Own Formatter) per generare i log in una struttura compatibile con Logging RFC dell'organizzazione

## Utilizzo di Powertools per AWS Lambda (Java) e AWS SAM per la registrazione strutturata

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Java Hello World di esempio con [moduli](#) Powertools for AWS Lambda (Java) integrati utilizzando AWS SAM. Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a CloudWatch X-Ray. La funzione restituisce un messaggio `hello world`.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Java 11
- [AWS CLI versione 2](#)

- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI. AWS SAM](#)

Distribuisci un esempio AWS SAM applicazione

1. Inizializza l'applicazione utilizzando il modello Java Hello World.

```
sam init --app-template hello-world-powertools-java --name sam-app --package-type Zip --runtime java11 --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi Enter.

 Note

Perché HelloWorldFunction potrebbe non avere un'autorizzazione definita, va bene? , assicurati di entrarey.

5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query 'Stacks[0].Outputs[?OutputKey=='HelloWorldApi'].OutputValue' --output text
```

6. Richiama l'endpoint dell'API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere i log per la funzione, esegui [sam logs](#). Per ulteriori informazioni, consulta l'argomento relativo all'[utilizzo dei log](#) nella Guida per sviluppatori AWS Serverless Application Model .

```
sam logs --stack-name sam-app
```

L'output del log ha la struttura seguente:

```
2025/09/03/[$LATEST]851411a899b545eea2cffebe4cfbec81 2023-02-03T09:24:34.095000
  INIT_START Runtime Version: java:11.v15    Runtime Version ARN: arn:aws:lambda:eu-
central-1::runtime:0a25e3e7a1cc9ce404bc435eeb2ad358d8fa64338e618d0c224fe509403583ca
2025/09/03/[$LATEST]851411a899b545eea2cffebe4cfbec81 2023-02-03T09:24:34.114000
  Picked up JAVA_TOOL_OPTIONS: -XX:+TieredCompilation -XX:TieredStopAtLevel=1
2025/09/03/[$LATEST]851411a899b545eea2cffebe4cfbec81 2023-02-03T09:24:34.793000
  Transforming org/apache/logging/log4j/core/lookup/JndiLookup
(lambdainternal.CustomerClassLoader@1a6c5a9e)
2025/09/03/[$LATEST]851411a899b545eea2cffebe4cfbec81 2023-02-03T09:24:35.252000
  START RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765 Version: $LATEST
2025/09/03/[$LATEST]851411a899b545eea2cffebe4cfbec81 2023-02-03T09:24:36.531000 {
  "_aws": {
    "Timestamp": 1675416276051,
    "CloudWatchMetrics": [
      {
        "Namespace": "sam-app-powerools-java",
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
        ],
        "Dimensions": [
          [
            "Service",
            "FunctionName"
          ]
        ]
      }
    ],
    "function_request_id": "7fcf1548-d2d4-41cd-a9a8-6ae47c51f765",
    "traceId":
    "Root=1-63dcd2d1-25f90b9d1c753a783547f4dd;Parent=e29684c1be352ce4;Sampled=1",
```

```

    "FunctionName": "sam-app-HelloWorldFunction-y9Iu1FLJJBGD",
    "functionVersion": "$LATEST",
    "ColdStart": 1.0,
    "Service": "service_undefined",
    "logStreamId": "2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81",
    "executionEnvironment": "AWS_Lambda_java11"
}
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.974000 Feb
 03, 2023 9:24:36 AM com.amazonaws.xray.AWSXRayRecorder <init>
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.993000 Feb
 03, 2023 9:24:36 AM com.amazonaws.xray.config.DaemonConfiguration <init>
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.993000
INFO: Environment variable AWS_XRAY_DAEMON_ADDRESS is set. Emitting to daemon on
address XXXX.XXXX.XXXX.XXXX:2000.
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:37.331000
09:24:37.294 [main] INFO helloworld.App - {"version":null,"resource":"/
hello","path":"/hello/","httpMethod":"GET","headers":{"Accept":"*/
*","CloudFront-Forwarded-Proto":"https","CloudFront-Is-Desktop-
Viewer":"true","CloudFront-Is-Mobile-Viewer":"false","CloudFront-Is-
SmartTV-Viewer":"false","CloudFront-Is-Tablet-Viewer":"false","CloudFront-
Viewer-ASN":"16509","CloudFront-Viewer-Country":"IE","Host":"XXXX.execute-
api.eu-central-1.amazonaws.com","User-Agent":"curl/7.86.0","Via":"2.0
f0300a9921a99446a44423d996042050.cloudfront.net (CloudFront)","X-Amz-
Cf-Id":"t9W5ByT11HaY33NM8YioKECn_4eMpNsOMPfEVRczD7T1RdhbtivV1Q==","X-
Amzn-Trace-Id":"Root=1-63dcd2d1-25f90b9d1c753a783547f4dd","X-Forwarded-
For":"XX.XXX.XXX.XX, XX.XXX.XXX.XX","X-Forwarded-Port":"443","X-
Forwarded-Proto":"https"},"multiValueHeaders":{"Accept":["*/
*"],"CloudFront-Forwarded-Proto":["https"],"CloudFront-Is-Desktop-Viewer":
["true"],"CloudFront-Is-Mobile-Viewer":["false"],"CloudFront-Is-SmartTV-
Viewer":["false"],"CloudFront-Is-Tablet-Viewer":["false"],"CloudFront-Viewer-
ASN":["16509"],"CloudFront-Viewer-Country":["IE"],"Host":["XXXX.execute-
api.eu-central-1.amazonaws.com"],"User-Agent":["curl/7.86.0"],"Via":["2.0
f0300a9921a99446a44423d996042050.cloudfront.net (CloudFront)","X-Amz-
Cf-Id":["t9W5ByT11HaY33NM8YioKECn_4eMpNsOMPfEVRczD7T1RdhbtivV1Q=="],"X-
Amzn-Trace-Id":["Root=1-63dcd2d1-25f90b9d1c753a783547f4dd"],"X-Forwarded-
For":["XXX, XXX"],"X-Forwarded-Port":["443"],"X-Forwarded-Proto":
["https"]},"queryStringParameters":null,"multiValueQueryStringParameters":null,"pathParamet
{"accountId":"XXX","stage":"Prod","resourceId":"at73a1","requestId":"ba09ecd2-
acf3-40f6-89af-fad32df67597","operationName":null,"identity":
{"cognitoIdentityPoolId":null,"accountId":null,"cognitoIdentityId":null,"caller":null,"apiK
hello","httpMethod":"GET","apiId":"XXX","path":"/Prod/
hello/","authorizer":null},"body":null,"isBase64Encoded":false}

```

```

2025/09/03/[$LATEST]851411a899b545eea2cffebe4cfbec81 2023-02-03T09:24:37.351000
09:24:37.351 [main] INFO helloworld.App - Retrieving https://
checkip.amazonaws.com
2025/09/03/[$LATEST]851411a899b545eea2cffebe4cfbec81 2023-02-03T09:24:39.313000 {
  "function_request_id": "7fcf1548-d2d4-41cd-a9a8-6ae47c51f765",
  "traceId":
"Root=1-63dcd2d1-25f90b9d1c753a783547f4dd;Parent=e29684c1be352ce4;Sampled=1",
  "xray_trace_id": "1-63dcd2d1-25f90b9d1c753a783547f4dd",
  "functionVersion": "$LATEST",
  "Service": "service_undefined",
  "logStreamId": "2025/09/03/[$LATEST]851411a899b545eea2cffebe4cfbec81",
  "executionEnvironment": "AWS_Lambda_java11"
}
2025/09/03/[$LATEST]851411a899b545eea2cffebe4cfbec81 2023-02-03T09:24:39.371000 END
RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765
2025/09/03/[$LATEST]851411a899b545eea2cffebe4cfbec81 2023-02-03T09:24:39.371000
REPORT RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765    Duration: 4118.98 ms
Billed Duration: 5275 ms    Memory Size: 512 MB    Max Memory Used: 152 MB    Init
Duration: 1155.47 ms
XRAY TraceId: 1-63dcd2d1-25f90b9d1c753a783547f4dd    SegmentId: 3a028fee19b895cb
Sampled: true

```

8. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

## Gestione della conservazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, elimina il gruppo di log o configura un periodo di conservazione dopo il quale i log CloudWatch vengono eliminati automaticamente. Per impostare la conservazione dei log, aggiungi quanto segue al tuo AWS SAM modello:

```

Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup

```



```
Properties:
  LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
  RetentionInDays: 7
```

## Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor di codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, trovi l'output del registro nella sezione Dettagli.

## Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Aprire la pagina dei gruppi di [log](#) sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (`/aws/lambda/your-function-name`).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'[istanza della funzione](#). Un flusso di log viene visualizzato quando si aggiorna la funzione Lambda e quando vengono create istanze aggiuntive per gestire chiamate simultanee. Per trovare i log per una chiamata specifica, ti consigliamo di strumentare la tua funzione con AWS X-Ray. X-Ray registra i dettagli sulla richiesta e sul flusso di log nella traccia.

## Visualizzazione dei log utilizzando AWS Command Line Interface (AWS CLI)

AWS CLI È uno strumento open source che puoi utilizzare per interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della [AWS CLI versione 2](#).

È possibile utilizzare [AWS CLI](#) per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

## Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "LogResult":
    "U1RBULQgUmVxdWVzdElk0iA4N2QwNDRI0C1mMTU0LTExZTgt0GNkYS0yOTc0YzVlNGZiMjEgVmVyc2l1vb...",
  "ExecutedVersion": "$LATEST"
}
```

## Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'base64 utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'`cli-binary-format` opzione è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ22luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
Duration: 80 ms      Memory Size: 128 MB      Max Memory Used: 73 MB
```

L'utilità `base64` è disponibile su Linux, macOS e [Ubuntu su Windows](#). Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

## Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza sed per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'opzione `cli-binary-format` è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

## Example(solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il comando seguente per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

## Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
```

```

    "ExecutedVersion": "$LATEST"
  }
  {
    "events": [
      {
        "timestamp": 1559763003171,
        "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
        "ingestionTime": 1559763003309
      },
      {
        "timestamp": 1559763003173,
        "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\n",
\r ...",
        "ingestionTime": 1559763018353
      },
      {
        "timestamp": 1559763003173,
        "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
        "ingestionTime": 1559763018353
      },
      {
        "timestamp": 1559763003218,
        "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
        "ingestionTime": 1559763018353
      },
      {
        "timestamp": 1559763003218,
        "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
MB\t\n",
        "ingestionTime": 1559763018353
      }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
  }
}

```

## Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o [configurare un periodo di conservazione](#) trascorso il quale i log vengono eliminati automaticamente.

### Codice di registrazione dei log di esempio

L' GitHub archivio di questa guida include applicazioni di esempio che dimostrano l'uso di varie configurazioni di registrazione. Ogni applicazione di esempio include script per semplificare l'implementazione e la pulizia, un AWS SAM modello e risorse di supporto.

### Applicazioni Lambda di esempio in Java

- [example-java](#): una funzione Java che dimostra come utilizzare Lambda per elaborare gli ordini. Questa funzione illustra come definire e deserializzare un oggetto evento di input personalizzato, utilizzare l'SDK e registrare l' AWS output.
- [java-basic](#): una raccolta di funzioni Java minimali con unit test e configurazione della registrazione dei log delle variabili.
- [java-events](#): una raccolta di funzioni Java che contengono codice skeleton per la gestione degli eventi di vari servizi, ad esempio Gateway Amazon API, Amazon SQS e Amazon Kinesis. Queste funzioni utilizzano la versione più recente della libreria [aws-lambda-java-events](#) (3.0.0 e versioni successive). Questi esempi non richiedono l'SDK come dipendenza. AWS
- [s3-java](#) – Una funzione Java che elabora gli eventi di notifica da Amazon S3 e utilizza la Java Class Library (JCL) per creare anteprime dai file di immagine caricati.
- [layer-java](#): una funzione Java che illustra come utilizzare un livello Lambda per impacchettare dipendenze separate dal codice della funzione principale.

L'applicazione `java-basic` di esempio mostra una configurazione di registrazione minima che supporta i test di registrazione. Il codice del gestore utilizza il logger `LambdaLogger` fornito dall'oggetto contestuale. Per i test, l'applicazione utilizza una classe `TestLogger` personalizzata che implementa l'interfaccia `LambdaLogger` con un logger `Log4j2`. Utilizza `SLF4J` come facciata per la compatibilità con l'SDK. AWS Le librerie di logging sono escluse dall'output di compilazione per mantenere il pacchetto di implementazione di dimensioni ridotte.

## Strumentazione del codice Java in AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzarlo X-Ray per tracciare una richiesta mentre attraversa le risorse dell'applicazione, che possono includere funzioni Lambda e altri servizi. AWS

Per inviare dati di tracciamento a X-Ray, puoi utilizzare una delle due librerie SDK:

- [AWS Distro for OpenTelemetry \(ADOT\)](#): una distribuzione sicura, pronta per la produzione e AWS supportata dell'SDK (OtEL). OpenTelemetry
- [SDK AWS X-Ray per Java](#)— Un SDK per la generazione e l'invio di dati di traccia a. X-Ray
- [Powertools for AWS Lambda \(Java\)](#): un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori.

Ciascuno degli SDK offre modi per inviare i dati di telemetria al servizio. X-Ray Puoi quindi utilizzarlo X-Ray per visualizzare, filtrare e ottenere informazioni dettagliate sulle metriche delle prestazioni dell'applicazione per identificare problemi e opportunità di ottimizzazione.

### Important

The X-Ray e Powertools for AWS Lambda SDK fanno parte di una soluzione di strumentazione strettamente integrata offerta da. AWS I livelli Lambda ADOT fanno parte di uno standard di settore per la strumentazione di tracciamento che in generale raccoglie più dati, ma potrebbero non essere adatti a tutti i casi d'uso. È possibile implementare il tracciamento end-to-end utilizzando entrambe le soluzioni. X-Ray Per saperne di più sulla scelta tra di esse, consulta [Scelta tra AWS Distro for Open Telemetry e SDK. X-Ray](#)

## Sections

- [Usare Powertools per AWS Lambda \(Java\) e AWS SAM per tracciare](#)
- [Usare Powertools per AWS Lambda \(Java\) e AWS CDK per tracciare](#)
- [Utilizzo di ADOT per strumentare le funzioni Java](#)
- [Utilizzo dell'SDK per strumentare le funzioni Java X-Ray](#)
- [Attivazione del tracciamento con la console Lambda](#)
- [Attivazione del tracciamento con l'API Lambda](#)

- [Attivazione del tracciamento con CloudFormation](#)
- [Interpretazione e tracciamento X-Ray](#)
- [Archiviazione delle dipendenze di runtime in un livello \(SDK\) X-Ray](#)
- [X-Ray tracciamento in applicazioni di esempio \(SDK\) X-Ray](#)

Usare Powertools per AWS Lambda (Java) e AWS SAM per tracciare

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Java Hello World di esempio con [moduli](#) Powertools for AWS Lambda (Java) integrati utilizzando AWS SAM. Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a CloudWatch AWS X-Ray. La funzione restituisce un messaggio hello world.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Java 11 o versione successiva
- [AWS CLI versione 2](#)
- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI AWS SAM](#)

Distribuisci un esempio AWS SAM applicazione

1. Inizializza l'applicazione utilizzando il modello Java Hello World.

```
sam init --app-template hello-world-powertools-java --name sam-app --package-type Zip --runtime java11 --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi Enter.

 Note

Perché HelloWorldFunction potrebbe non avere un'autorizzazione definita, va bene? , assicurati di entrare.

5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query  
'Stacks[0].Outputs[?OutputKey=='HelloWorldApi'].OutputValue' --output text
```

6. Richiama l'endpoint dell'API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere le tracce per la funzione, esegui [sam traces](#).

```
sam traces
```

L'output della traccia ha il seguente aspetto:

```
New XRay Service Graph  
Start time: 2025-02-03 14:31:48+01:00  
End time: 2025-02-03 14:31:48+01:00  
Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-y9Iu1FLJJBGD -  
Edges: []  
Summary_statistics:  
  - total requests: 1  
  - ok count(2XX): 1  
  - error count(4XX): 0  
  - fault count(5XX): 0  
  - total response time: 5.587  
Reference Id: 1 - client - sam-app-HelloWorldFunction-y9Iu1FLJJBGD - Edges: [0]  
Summary_statistics:
```



```
- total requests: 0
- ok count(2XX): 0
- error count(4XX): 0
- fault count(5XX): 0
- total response time: 0
```

```
XRay Event [revision 3] at (2025-02-03T14:31:48.500000) with id
(1-63dd0cc4-3c869dec72a586875da39777) and duration (5.603s)
```

```
- 5.587s - sam-app-HelloWorldFunction-y9Iu1FLJJBGD [HTTP: 200]
- 4.053s - sam-app-HelloWorldFunction-y9Iu1FLJJBGD
  - 1.181s - Initialization
  - 4.037s - Invocation
    - 1.981s - ## handleRequest
      - 1.840s - ## getPageContents
  - 0.000s - Overhead
```

8. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

Usare Powertools per AWS Lambda (Java) e AWS CDK per tracciare

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Java Hello World di esempio con [moduli](#) Powertools for AWS Lambda (Java) integrati utilizzando AWS CDK. Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a CloudWatch AWS X-Ray. La funzione restituisce un messaggio hello world.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Java 11 o versione successiva
- [AWS CLI versione 2](#)
- [AWS CDK versione 2](#)
- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI. AWS SAM](#)

## Distribuisci un esempio AWS CDK applicazione

1. Crea una directory di progetto per la nuova applicazione.

```
mkdir hello-world  
cd hello-world
```

2. Inizializza l'app.

```
cdk init app --language java
```

3. Crea un progetto maven utilizzando il comando seguente:

```
mkdir app  
cd app  
mvn archetype:generate -DgroupId=helloworld -DartifactId=Function -  
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

4. Apri pom.xml nella directory hello-world\app\Function e sostituisci il codice esistente con il codice seguente, che include dipendenze e plug-in maven per Powertools.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://  
www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/  
maven-v4_0_0.xsd">  
  <modelVersion>4.0.0</modelVersion>  
  <groupId>helloworld</groupId>  
  <artifactId>Function</artifactId>  
  <packaging>jar</packaging>  
  <version>1.0-SNAPSHOT</version>  
  <name>Function</name>  
  <url>http://maven.apache.org</url>  
  <properties>  
    <maven.compiler.source>11</maven.compiler.source>  
    <maven.compiler.target>11</maven.compiler.target>  
    <log4j.version>2.17.2</log4j.version>  
  </properties>  
  <dependencies>  
    <dependency>  
      <groupId>junit</groupId>  
      <artifactId>junit</artifactId>  
      <version>3.8.1</version>  
      <scope>test</scope>
```

```

    </dependency>
    <dependency>
      <groupId>software.amazon.lambda</groupId>
      <artifactId>powertools-tracing</artifactId>
      <version>1.3.0</version>
    </dependency>
    <dependency>
      <groupId>software.amazon.lambda</groupId>
      <artifactId>powertools-metrics</artifactId>
      <version>1.3.0</version>
    </dependency>
    <dependency>
      <groupId>software.amazon.lambda</groupId>
      <artifactId>powertools-logging</artifactId>
      <version>1.3.0</version>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-core</artifactId>
      <version>1.2.2</version>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-events</artifactId>
      <version>3.11.1</version>
    </dependency>
  </dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>aspectj-maven-plugin</artifactId>
      <version>1.14.0</version>
      <configuration>
        <source>${maven.compiler.source}</source>
        <target>${maven.compiler.target}</target>
        <complianceLevel>${maven.compiler.target}</complianceLevel>
        <aspectLibraries>
          <aspectLibrary>
            <groupId>software.amazon.lambda</groupId>
            <artifactId>powertools-tracing</artifactId>
          </aspectLibrary>
          <aspectLibrary>
            <groupId>software.amazon.lambda</groupId>

```

```

        <artifactId>powertools-metrics</artifactId>
      </aspectLibrary>
    </aspectLibraries>
  </configuration>
</plugin>
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <version>3.4.1</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>shade</goal>
      </goals>
      <configuration>
        <transformers>
          <transformer
implementation="com.github.edwgiz.maven_shade_plugin.log4j2_cache_transformer.PluginsCache
          </transformer>
        </transformers>
        <createDependencyReducedPom>>false</
createDependencyReducedPom>
        <finalName>function</finalName>

      </configuration>
    </execution>
  </executions>
</dependencies>
<dependency>
  <groupId>com.github.edwgiz</groupId>

```

```

        <artifactId>maven-shade-plugin.log4j2-cache-file-
transformer</artifactId>
        <version>2.15</version>
    </dependency>
</dependencies>
</plugin>
</plugins>
</build>
</project>

```

5. Crea la directory `hello-world\app\src\main\resource` e crea `log4j.xml` per la configurazione del log.

```

mkdir -p src/main/resource
cd src/main/resource
touch log4j.xml

```

6. Apri `log4j.xml` e aggiungi il seguente codice.

```

<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <Console name="JsonAppender" target="SYSTEM_OUT">
            <JsonTemplateLayout
eventTemplateUri="classpath:LambdaJsonLayout.json" />
        </Console>
    </Appenders>
    <Loggers>
        <Logger name="JsonLogger" level="INFO" additivity="false">
            <AppenderRef ref="JsonAppender"/>
        </Logger>
        <Root level="info">
            <AppenderRef ref="JsonAppender"/>
        </Root>
    </Loggers>
</Configuration>

```

7. Apri `App.java` dalla directory `hello-world\app\Function\src\main\java\helloworld` e sostituisci il codice esistente con il codice seguente. Questo è il codice per la funzione Lambda.

```

package helloworld;

```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import software.amazon.lambda.powertools.logging.Logging;
import software.amazon.lambda.powertools.metrics.Metrics;
import software.amazon.lambda.powertools.tracing.CaptureMode;
import software.amazon.lambda.powertools.tracing.Tracing;

import static software.amazon.lambda.powertools.tracing.CaptureMode.*;

/**
 * Handler for requests to Lambda function.
 */
public class App implements RequestHandler<APIGatewayProxyRequestEvent,
    APIGatewayProxyResponseEvent> {
    Logger log = LogManager.getLogger(App.class);

    @Logging(logEvent = true)
    @Tracing(captureMode = DISABLED)
    @Metrics(captureColdStart = true)
    public APIGatewayProxyResponseEvent handleRequest(final
    APIGatewayProxyRequestEvent input, final Context context) {
        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "application/json");
        headers.put("X-Custom-Header", "application/json");

        APIGatewayProxyResponseEvent response = new APIGatewayProxyResponseEvent()
            .withHeaders(headers);
        try {
            final String pageContents = this.getPageContents("https://
            checkip.amazonaws.com");
        }
    }
}
```

```

        String output = String.format("{ \"message\": \"hello world\",
        \"location\": \"%s\" }", pageContents);

        return response
            .withStatusCode(200)
            .withBody(output);
    } catch (IOException e) {
        return response
            .withBody("{}")
            .withStatusCode(500);
    }
}

@Tracing(namespace = "getPageContents")
private String getPageContents(String address) throws IOException {
    log.info("Retrieving {}", address);
    URL url = new URL(address);
    try (BufferedReader br = new BufferedReader(new
InputStreamReader(url.openStream())) {
        return br.lines().collect(Collectors.joining(System.lineSeparator()));
    }
}
}

```

8. Apri `HelloWorldStack.java` dalla directory `hello-world/src/main/java/com/myorg` e sostituisci il codice esistente con il codice seguente. Questo codice utilizza [Lambda Constructor](#) e [ApiGatewayv2 Constructor](#) per creare un'API REST e una funzione Lambda.

```

package com.myorg;

import software.amazon.awscdk.*;
import software.amazon.awscdk.services.apigatewayv2.alpha.*;
import
    software.amazon.awscdk.services.apigatewayv2.integrations.alpha.HttpLambdaIntegration;
import
    software.amazon.awscdk.services.apigatewayv2.integrations.alpha.HttpLambdaIntegrationProps;
import software.amazon.awscdk.services.lambda.Code;
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.FunctionProps;
import software.amazon.awscdk.services.lambda.Runtime;
import software.amazon.awscdk.services.lambda.Tracing;
import software.amazon.awscdk.services.logs.RetentionDays;
import software.amazon.awscdk.services.s3.assets.AssetOptions;
import software.constructs.Construct;

```

```

import java.util.Arrays;
import java.util.List;

import static java.util.Collections.singletonList;
import static software.amazon.awscdk.BundlingOutput.ARCHIVED;

public class HelloWorldStack extends Stack {
    public HelloWorldStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloWorldStack(final Construct scope, final String id, final StackProps
props) {
        super(scope, id, props);

        List<String> functionPackagingInstructions = Arrays.asList(
            "/bin/sh",
            "-c",
            "cd Function " +
                "&& mvn clean install " +
                "&& cp /asset-input/Function/target/function.jar /asset-
output/"
        );
        BundlingOptions.Builder builderOptions = BundlingOptions.builder()
            .command(functionPackagingInstructions)
            .image(Runtime.JAVA_11.getBundlingImage())
            .volumes(singletonList(
                // Mount local .m2 repo to avoid download all the
dependencies again inside the container
                DockerVolume.builder()
                    .hostPath(System.getProperty("user.home") +
"/.m2/")
                    .containerPath("/root/.m2/")
                    .build()
            ))
            .user("root")
            .outputType(ARCHIVED);

        Function function = new Function(this, "Function", FunctionProps.builder()
            .runtime(Runtime.JAVA_11)
            .code(Code.fromAsset("app", AssetOptions.builder()
                .bundling(builderOptions)
                .command(functionPackagingInstructions)

```



```

        .build())
        .build()))
    .handler("helloworld.App::handleRequest")
    .memorySize(1024)
    .tracing(Tracing.ACTIVE)
    .timeout(Duration.seconds(10))
    .logRetention(RetentionDays.ONE_WEEK)
    .build());

    HttpApi httpApi = new HttpApi(this, "sample-api", HttpApiProps.builder()
        .apiName("sample-api")
        .build());

    httpApi.addRoutes(AddRoutesOptions.builder()
        .path("/")
        .methods(singleList(HttpMethod.GET))
        .integration(new HttpLambdaIntegration("function", function,
            HttpLambdaIntegrationProps.builder()
                .payloadFormatVersion(PayloadFormatVersion.VERSION_2_0)
                .build()))
        .build());

    new CfnOutput(this, "HttpApi", CfnOutputProps.builder()
        .description("Url for Http Api")
        .value(httpApi.getApiEndpoint())
        .build());
    }
}

```

9. Apri `pom.xml` dalla directory `hello-world` e sostituisci il codice esistente con il codice seguente.

```

<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd"
    xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.myorg</groupId>
    <artifactId>hello-world</artifactId>
    <version>0.1</version>

```

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <cdk.version>2.70.0</cdk.version>
  <constructs.version>[10.0.0,11.0.0)</constructs.version>
  <junit.version>5.7.1</junit.version>
</properties>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>

    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>exec-maven-plugin</artifactId>
      <version>3.0.0</version>
      <configuration>
        <mainClass>com.myorg.HelloWorldApp</mainClass>
      </configuration>
    </plugin>
  </plugins>
</build>

<dependencies>
  <!-- AWS Cloud Development Kit -->
  <dependency>
    <groupId>software.amazon.awscdk</groupId>
    <artifactId>aws-cdk-lib</artifactId>
    <version>${cdk.version}</version>
  </dependency>
  <dependency>
    <groupId>software.constructs</groupId>
    <artifactId>constructs</artifactId>
    <version>${constructs.version}</version>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
```

```

        <artifactId>junit-jupiter</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>software.amazon.awscdk</groupId>
        <artifactId>apigatewayv2-alpha</artifactId>
        <version>${cdk.version}-alpha.0</version>
    </dependency>
    <dependency>
        <groupId>software.amazon.awscdk</groupId>
        <artifactId>apigatewayv2-integrations-alpha</artifactId>
        <version>${cdk.version}-alpha.0</version>
    </dependency>
</dependencies>
</project>

```

10. Assicurati di essere nella `hello-world` directory e distribuisci la tua applicazione.

```
cdk deploy
```

11. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
'Stacks[0].Outputs[?OutputKey=='HttpApi'].OutputValue' --output text
```

12. Richiama l'endpoint dell'API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

13. Per ottenere le tracce per la funzione, esegui [sam traces](#).

```
sam traces
```

L'output della traccia ha il seguente aspetto:

```
New XRay Service Graph
```

```

Start time: 2025-02-03 14:59:50+00:00
End time: 2025-02-03 14:59:50+00:00
Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-YBg8yfYt0c9j -
Edges: [1]
  Summary_statistics:
    - total requests: 1
    - ok count(2XX): 1
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 0.924
Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-YBg8yfYt0c9j
- Edges: []
  Summary_statistics:
    - total requests: 1
    - ok count(2XX): 1
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 0.016
Reference Id: 2 - client - sam-app-HelloWorldFunction-YBg8yfYt0c9j - Edges: [0]
  Summary_statistics:
    - total requests: 0
    - ok count(2XX): 0
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 0

XRay Event [revision 1] at (2025-02-03T14:59:50.204000) with id
(1-63dd2166-434a12c22e1307ff2114f299) and duration (0.924s)
- 0.924s - sam-app-HelloWorldFunction-YBg8yfYt0c9j [HTTP: 200]
- 0.016s - sam-app-HelloWorldFunction-YBg8yfYt0c9j
  - 0.739s - Initialization
  - 0.016s - Invocation
    - 0.013s - ## lambda_handler
      - 0.000s - ## app.hello
    - 0.000s - Overhead

```

14. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
cdk destroy
```

## Utilizzo di ADOT per strumentare le funzioni Java

ADOT fornisce [livelli](#) Lambda completamente gestiti che mettono insieme tutto il necessario per raccogliere i dati di telemetria utilizzando l'SDK OTel. Usando questo livello, è possibile strumentare le funzioni Lambda senza dover modificare alcun codice funzione. È inoltre possibile configurare il livello per eseguire l'inizializzazione personalizzata di OTel. Per ulteriori informazioni, consulta la sezione relativa alla [configurazione personalizzata per ADOT Collector su Lambda](#) nella documentazione di ADOT.

Per i runtime Java, è possibile scegliere tra due livelli da utilizzare:

- AWS layer Lambda gestito per ADOT Java (Auto-instrumentation Agent): questo livello trasforma automaticamente il codice funzione all'avvio per raccogliere dati di tracciamento. Per istruzioni dettagliate su come utilizzare questo livello insieme all'agente Java ADOT, vedi [AWS Distro for OpenTelemetry Lambda Support for Java \(Auto-instrumentation Agent\) nella documentazione ADOT](#).
- AWS layer Lambda gestito per ADOT Java: questo livello fornisce anche strumentazione integrata per le funzioni Lambda, ma richiede alcune modifiche manuali al codice per inizializzare l'SDK Otel. Per istruzioni dettagliate su come utilizzare questo livello, consulta [AWS Distro for OpenTelemetry Lambda Support for Java nella documentazione ADOT](#).

Per istruzioni sulla conversione degli eventi dell'API Lambda Telemetry in intervalli, consulta. OpenTelemetry [the section called “Conversione di eventi in intervalli OTel”](#)

## Utilizzo dell'SDK per strumentare le funzioni Java X-Ray

Per registrare i dati sulle chiamate effettuate dalla funzione ad altre risorse e servizi dell'applicazione, è possibile aggiungere l' X-Ray SDK per Java alla configurazione di build. L'esempio seguente mostra una configurazione di build Gradle che include le librerie che attivano la strumentazione automatica dei client. AWS SDK for Java 2.x

Example [build.gradle](#) — Tracciamento delle dipendenze

```
dependencies {  
    implementation platform('software.amazon.awssdk:bom:2.16.1')  
    implementation platform('com.amazonaws:aws-xray-recorder-sdk-bom:2.11.0')  
    ...  
    implementation 'com.amazonaws:aws-xray-recorder-sdk-core'  
    implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk'  
}
```

```
implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-v2-instrumentor'  
...  
}
```

Dopo aver aggiunto le dipendenze corrette e apportato le modifiche al codice necessarie, attiva il tracciamento nella configurazione della funzione tramite la console Lambda o l'API.

### Attivazione del tracciamento con la console Lambda

Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. In Strumenti di monitoraggio aggiuntivi scegli Modifica.
5. In CloudWatch Application Signals and AWS X-Ray, scegli **Abilita** per le tracce del servizio Lambda.
6. Scegli Save (Salva).

### Attivazione del tracciamento con l'API Lambda

Configura il tracciamento sulla tua funzione Lambda con AWS CLI or AWS SDK, utilizza le seguenti operazioni API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

Il seguente AWS CLI comando di esempio abilita la traccia attiva su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \  
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

### Attivazione del tracciamento con CloudFormation

Per attivare la traccia su una `AWS::Lambda::Function` risorsa in un CloudFormation modello, utilizzate la proprietà `TracingConfig`

Example [function-inline.yml](#) — Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
    ...
```

Per una `AWS::Serverless::Function` risorsa AWS Serverless Application Model (AWS SAM), usa la `Tracing` proprietà.

Example [template.yml](#) — Configurazione di tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
    ...
```

### Interpretazione e tracciamento X-Ray

La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al [ruolo di esecuzione](#) della funzione. Altrimenti, aggiungi la [AWSXRayDaemonWriteAccess](#) policy al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico del [X-Ray servizio](#) mostra le informazioni sull'applicazione e su tutti i suoi

componenti. Il seguente esempio mostra un'applicazione con due funzioni. La funzione principale elabora gli eventi e talvolta restituisce errori. La seconda funzione in alto elabora gli errori che appaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon CloudWatch Logs.



X-Ray non traccia tutte le richieste all'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento sia efficiente, fornendo comunque un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non puoi configurare la frequenza di X-Ray campionamento per le tue funzioni.

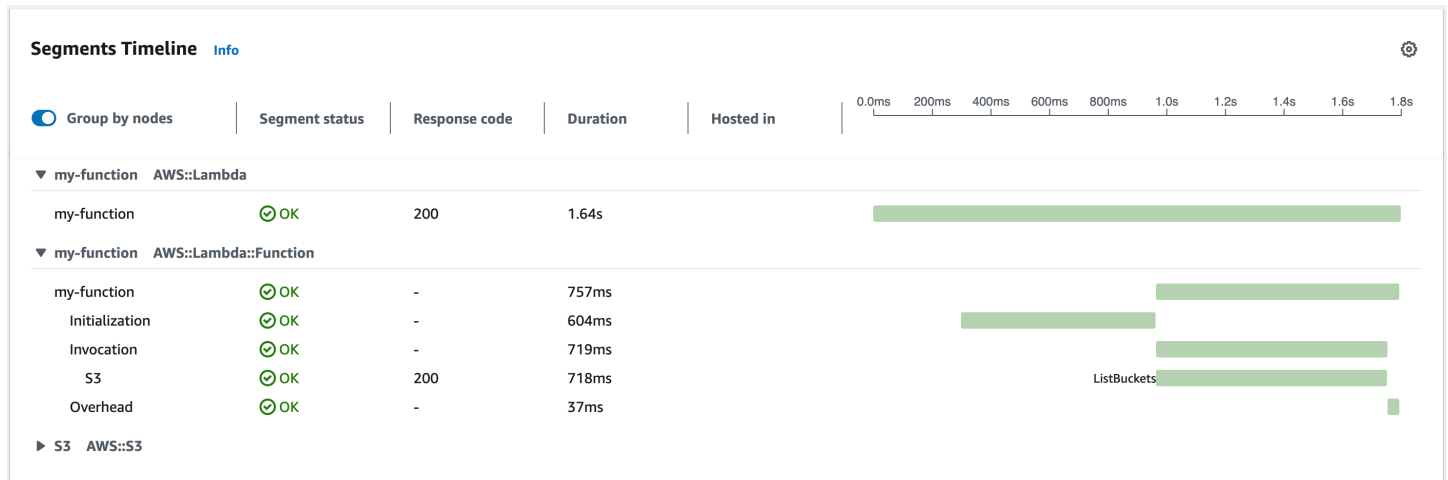
In X-Ray, una traccia registra le informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati my-function, ma uno ha l'origine AWS :: Lambda e l'altro ha l'origine



`AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.

#### Note

AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti.

Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regioni, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia nel nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- **Inizializzazione** – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il [codice di inizializzazione](#). Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.
- **Chiamata**: rappresenta il tempo impiegato per eseguire il codice del gestore.
- **Overhead**: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla struttura dei segmenti di funzioni vecchio e nuovo stile, consulta [the section called “Comprensione delle tracce X-Ray”](#).

#### Note

Le funzioni [SnapStart](#) includono anche un sottosegmento `Restore`. Il sottosegmento `Restore` mostra il tempo impiegato da Lambda per ripristinare uno snapshot, caricare il runtime ed eseguire qualsiasi [hook di runtime](#). Il processo di ripristino degli snapshot può includere il tempo dedicato ad attività esterne alla MicroVM. Questa volta è riportato nel segmento secondario `Restore`. Non ti viene addebitato il tempo trascorso fuori dalla microVM per il ripristino di una snapshot.

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta la sezione [SDK AWS X-Ray per Java](#) nella Guida per gli sviluppatori di AWS X-Ray .

#### Prezzi

Puoi utilizzare il X-Ray tracciamento gratuitamente ogni mese fino a un determinato limite come parte del piano gratuito. AWS Oltre tale soglia, vengono X-Ray addebitati i costi per l'archiviazione e il recupero delle tracce. Per ulteriori informazioni, consultare [Prezzi di AWS X-Ray](#).

### Archiviazione delle dipendenze di runtime in un livello (SDK) X-Ray

Se si utilizza l' X-Ray SDK per configurare il codice funzione dei client AWS SDK, il pacchetto di distribuzione può diventare piuttosto ingombrante. Per evitare di caricare le dipendenze di runtime ogni volta che aggiorni il codice funzione, impacchetta l' X-Ray SDK in un layer Lambda. [???](#)

L'esempio seguente mostra una `AWS::Serverless::LayerVersion` risorsa che memorizza l'SDK AWS SDK per Java and X-Ray per Java.

Example [template.yml](#) — Livello delle dipendenze

Resources:

```
function:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: build/distributions/blank-java.zip
    Tracing: Active
    Layers:
      - !Ref libs
    ...
libs:
  Type: AWS::Serverless::LayerVersion
  Properties:
    LayerName: blank-java-lib
    Description: Dependencies for the blank-java sample app.
    ContentUri: build/blank-java-lib.zip
    CompatibleRuntimes:
      - java25
```

Con questa configurazione, si aggiorna il livello della libreria solo se si modificano le dipendenze di runtime. Poiché il pacchetto di implementazione della funzione contiene solo il codice, questo può contribuire a ridurre i tempi di caricamento.

La creazione di un livello per le dipendenze richiede modifiche alla configurazione di compilazione per generare l'archivio del livello prima della distribuzione. Per un esempio funzionante, vedi l'applicazione di esempio [java-basic](#) su GitHub

X-Ray tracciamento in applicazioni di esempio (SDK) X-Ray

L' GitHub archivio di questa guida include applicazioni di esempio che dimostrano l'uso del tracciamento. X-Ray Ogni applicazione di esempio include script per semplificare l'implementazione e la pulizia, un AWS SAM modello e risorse di supporto.

Applicazioni Lambda di esempio in Java

- [example-java](#): una funzione Java che dimostra come utilizzare Lambda per elaborare gli ordini. Questa funzione illustra come definire e deserializzare un oggetto evento di input personalizzato, utilizzare l'SDK e registrare l' AWS output.
- [java-basic](#): una raccolta di funzioni Java minimali con unit test e configurazione della registrazione dei log delle variabili.
- [java-events](#): una raccolta di funzioni Java che contengono codice skeleton per la gestione degli eventi di vari servizi, ad esempio Gateway Amazon API, Amazon SQS e Amazon Kinesis. Queste

funzioni utilizzano la versione più recente della libreria [aws-lambda-java-events](#) (3.0.0 e versioni successive). Questi esempi non richiedono l'SDK come dipendenza. AWS

- [s3-java](#) – Una funzione Java che elabora gli eventi di notifica da Amazon S3 e utilizza la Java Class Library (JCL) per creare anteprime dai file di immagine caricati.
- [layer-java](#): una funzione Java che illustra come utilizzare un livello Lambda per impacchettare dipendenze separate dal codice della funzione principale.

Tutte le applicazioni di esempio hanno il tracciamento attivo abilitato per le funzioni Lambda. Ad esempio, l'`s3-java` applicazione mostra la strumentazione automatica dei AWS SDK for Java 2.x client, la gestione dei segmenti per i test, i sottosegmenti personalizzati e l'uso dei livelli Lambda per memorizzare le dipendenze di runtime.

## Applicazioni di esempio Java per AWS Lambda

L' GitHub archivio di questa guida fornisce applicazioni di esempio che dimostrano l'uso di Java in AWS Lambda. Ogni applicazione di esempio include script per semplificare l'implementazione e la pulizia, un CloudFormation modello e risorse di supporto.

### Applicazioni Lambda di esempio in Java

- [example-java](#): una funzione Java che dimostra come utilizzare Lambda per elaborare gli ordini. Questa funzione illustra come definire e deserializzare un oggetto evento di input personalizzato, utilizzare l'SDK e registrare l' AWS output.
- [java-basic](#): una raccolta di funzioni Java minimali con unit test e configurazione della registrazione dei log delle variabili.
- [java-events](#): una raccolta di funzioni Java che contengono codice skeleton per la gestione degli eventi di vari servizi, ad esempio Gateway Amazon API, Amazon SQS e Amazon Kinesis. Queste funzioni utilizzano la versione più recente della libreria [aws-lambda-java-events](#) (3.0.0 e versioni successive). Questi esempi non richiedono l'SDK come dipendenza. AWS
- [s3-java](#) – Una funzione Java che elabora gli eventi di notifica da Amazon S3 e utilizza la Java Class Library (JCL) per creare anteprime dai file di immagine caricati.
- [layer-java](#): una funzione Java che illustra come utilizzare un livello Lambda per impacchettare dipendenze separate dal codice della funzione principale.

### Esecuzione dei framework Java più diffusi su Lambda

- [spring-cloud-function-samples](#) — Un esempio tratto da Spring che mostra come utilizzare il framework Spring Cloud Function per creare funzioni [Lambda](#). AWS
- [Demo dell'applicazione Spring Boot senza server](#): un esempio che mostra come configurare una tipica applicazione Spring Boot in un runtime Java gestito con e senza o come immagine nativa SnapStart GraalVM con un runtime personalizzato.
- [Demo dell'applicazione Serverless Micronaut](#): un esempio che mostra come utilizzare Micronaut in un runtime Java gestito con e senza o come immagine nativa GraalVM con un runtime SnapStart personalizzato. Scopri [Micronaut/Lambda di più nelle guide](#).
- [Demo dell'applicazione Serverless Quarkus](#): un esempio che mostra come utilizzare Quarkus in un runtime Java gestito con e senza SnapStart o come immagine nativa GraalVM con un runtime personalizzato. Scopri di più nella guida e nella guida. [Quarkus/Lambda](#) [Quarkus/SnapStart](#)

Se non hai mai utilizzato le funzioni Lambda in Java, inizia con gli esempi `java-basic`. Per iniziare con le origini eventi Lambda, consulta gli esempi `java-events`. Entrambi questi set di esempi mostrano l'uso delle librerie Java, delle variabili di ambiente, dell'SDK e dell' AWS SDK di Lambda. AWS X-Ray Questi esempi richiedono una configurazione minima e possono essere implementati dalla riga di comando in meno di un minuto.

## Compilazione di funzioni Lambda con Go

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato nativamente in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un [OS-only runtime](#) (la famiglia `provided` di runtime) per distribuire le funzioni Go su Lambda.

Per informazioni sull'uso dei livelli con Go, consulta [the section called “Livelli”](#)

### Argomenti

- [Supporto per il runtime Go](#)
- [Strumenti e librerie](#)
- [Definisci i gestori di funzioni Lambda in Go](#)
- [Utilizzo dell'oggetto contestuale Lambda per recuperare le informazioni sulla funzione Go](#)
- [Distribuisci funzioni Lambda per Go con gli archivi di file .zip](#)
- [Distribuzione delle funzioni Go Lambda con immagini di container](#)
- [Utilizzo dei livelli per le funzioni Lambda in Go](#)
- [Registrare e monitorare le funzioni Lambda con Go](#)
- [Strumentazione del codice Go AWS Lambda](#)

### Supporto per il runtime Go

Il runtime gestito Go 1.x per Lambda è [obsoleto](#). Se hai funzioni che utilizzano il runtime Go 1.x, devi eseguire la migrazione delle tue funzioni a `provided.al2023` o `provided.al2`. I runtime `provided.al2023` e `provided.al2` offrono diversi vantaggi rispetto a Go 1.x, tra cui il supporto per l'architettura arm64 (processori AWS Graviton2), file binari più piccoli e tempi di richiamo leggermente più rapidi.

Per questa migrazione, non sono necessarie modifiche al codice. Le uniche modifiche richieste riguardano la modalità di creazione del pacchetto di implementazione e il runtime utilizzato per creare

la funzione. Per ulteriori informazioni, consulta [Migrazione AWS Lambda delle funzioni dal Go1.x runtime al runtime personalizzato su Amazon Linux 2 nel blog di Compute](#). AWS

| Nome            | Identificatore  | Sistema operativo | Data di ritiro | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|-----------------|-----------------|-------------------|----------------|---------------------------|-------------------------------|
| OS-only Runtime | provided.al2023 | Amazon Linux 2023 | 30 giugno 2029 | 31 luglio 2029            | 31 agosto 2029                |

## Strumenti e librerie

Lambda fornisce i seguenti strumenti e librerie per il runtime Go:

- [AWS SDK per Go v2](#): L' AWS SDK ufficiale per il linguaggio di programmazione Go.
- [github.com/aws/aws-lambda-go/lambda](https://github.com/aws/aws-lambda-go/lambda): L'implementazione del modello di programmazione Lambda per Go. Questo pacchetto viene utilizzato da per richiamare il gestore. AWS Lambda ???
- [github.com/aws/aws-lambda-go/lambdacontext](https://github.com/aws/aws-lambda-go/lambdacontext): aiutanti per accedere alle informazioni di contesto dall'oggetto di contesto. ???
- [github.com/aws/aws-lambda-go/events](https://github.com/aws/aws-lambda-go/events): questa libreria fornisce definizioni di tipo per le integrazioni di sorgenti di eventi comuni.
- [github.com/aws/aws-lambda-go/cmd/build-lambda-zip](https://github.com/aws/aws-lambda-go/cmd/build-lambda-zip): questo strumento può essere utilizzato per creare un archivio di file .zip su Windows.

Per ulteriori informazioni, consulta aws-lambda-go on. <https://github.com/aws/aws-lambda-go> GitHub

Lambda fornisce le seguenti applicazioni di esempio per il runtime di Go:

### Applicazioni Lambda di esempio in Go

- [go-al2](#): una funzione hello world che restituisce l'indirizzo IP pubblico. Questa app utilizza il runtime personalizzato provided.al2.
- [blank-go](#) — Una funzione Go che mostra l'uso delle librerie Go di Lambda, della registrazione, delle variabili di ambiente e dell'SDK. AWS Questa app utilizza il runtime go1.x.

## Definisci i gestori di funzioni Lambda in Go

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come lavorare con i gestori di funzioni Lambda in Go, inclusa la configurazione del progetto, le convenzioni di denominazione e le migliori pratiche. Questa pagina include anche un esempio di funzione Go Lambda che acquisisce informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon Simple Storage Service (Amazon S3). Per informazioni su come distribuire una funzione dopo averla scritta, consulta o. [the section called “Implementazione di archivi di file .zip”](#) [the section called “Implementazione di immagini di container”](#)

### Argomenti

- [Configurazione di Go Handler](#)
- [Esempio di codice della funzione Lambda](#)
- [Convenzioni di denominazione dei gestori](#)
- [Definizione e accesso all'oggetto evento di input](#)
- [Accesso e utilizzo dell'oggetto contestuale Lambda](#)
- [Firme dell'handler valide per gli handler Go](#)
- [Utilizzo di AWS SDK per Go v2 nel tuo gestore](#)
- [Accesso alle variabili d'ambiente](#)
- [Utilizzo dello stato globale](#)
- [Procedure consigliate di codice per le funzioni Go Lambda](#)

### Configurazione di Go Handler

Una funzione Lambda scritta in [Go](#) viene creata come eseguibile di Go. È possibile inizializzare un progetto di funzione Go Lambda nello stesso modo in cui iniziassi qualsiasi altro progetto Go utilizzando il seguente comando: `go mod init`

```
go mod init example-go
```

Ecco il nome del `example-go` modulo. Puoi sostituire con un valore qualsiasi. Questo comando inizializza il progetto e genera il `go.mod` file che elenca le dipendenze del progetto.



Utilizzate il `go get` comando per aggiungere eventuali dipendenze esterne al progetto. Ad esempio, per tutte le funzioni Lambda in Go, devi includere github. [com/aws-go/lambda](https://github.com/aws/aws-lambda-go) pacchetto `/aws-lambda-`, che implementa il modello di programmazione Lambda per Go. Installa il pacchetto con il seguente comando `go get`:

```
go get github.com/aws/aws-lambda-go
```

Il codice della funzione dovrebbe risiedere in un file Go. In questo esempio viene assegnato un nome a questo file `main.go`. In questo file, implementa la logica della funzione principale in un metodo di gestione, oltre a una funzione `main()` che chiama questo handler.

### Esempio di codice della funzione Lambda

Il seguente esempio di codice della funzione Go Lambda raccoglie le informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3.

#### Example **funzione** Lambda `main.go`

```
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "log"
    "os"
    "strings"

    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

type Order struct {
    OrderID string `json:"order_id"`
    Amount  float64 `json:"amount"`
    Item    string  `json:"item"`
}

var (
    s3Client *s3.Client
)
```

```
func init() {
    // Initialize the S3 client outside of the handler, during the init phase
    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Fatalf("unable to load SDK config, %v", err)
    }

    s3Client = s3.NewFromConfig(cfg)
}

func uploadReceiptToS3(ctx context.Context, bucketName, key, receiptContent string)
    error {
    _, err := s3Client.PutObject(ctx, &s3.PutObjectInput{
        Bucket: &bucketName,
        Key:     &key,
        Body:    strings.NewReader(receiptContent),
    })
    if err != nil {
        log.Printf("Failed to upload receipt to S3: %v", err)
        return err
    }
    return nil
}

func handleRequest(ctx context.Context, event json.RawMessage) error {
    // Parse the input event
    var order Order
    if err := json.Unmarshal(event, &order); err != nil {
        log.Printf("Failed to unmarshal event: %v", err)
        return err
    }

    // Access environment variables
    bucketName := os.Getenv("RECEIPT_BUCKET")
    if bucketName == "" {
        log.Printf("RECEIPT_BUCKET environment variable is not set")
        return fmt.Errorf("missing required environment variable RECEIPT_BUCKET")
    }

    // Create the receipt content and key destination
    receiptContent := fmt.Sprintf("OrderID: %s\nAmount: $%.2f\nItem: %s",
        order.OrderID, order.Amount, order.Item)
    key := "receipts/" + order.OrderID + ".txt"
```

```
// Upload the receipt to S3 using the helper method
if err := uploadReceiptToS3(ctx, bucketName, key, receiptContent); err != nil {
    return err
}

log.Printf("Successfully processed order %s and stored receipt in S3 bucket %s",
order.OrderID, bucketName)
return nil
}

func main() {
    lambda.Start(handleRequest)
}
```

Questo file `main.go` contiene le sezioni seguenti:

- `package main`: in Go, il pacchetto contenente la funzione `func main()` deve essere sempre denominato `main`.
- Blocco `import`: utilizza questo blocco per includere le librerie richieste dalla funzione Lambda.
- `type Order struct {}` block: definisce la forma dell'evento di input previsto in questa struttura Go.
- `var ()` block: usa questo blocco per definire tutte le variabili globali che utilizzerai nella tua funzione Lambda.
- `func init() {}`: includi in questo metodo qualsiasi codice che desideri che Lambda esegua durante la [fase di inizializzazione](#). `init()`
- `func uploadReceiptToS3(...) {}`: questo è un metodo helper a cui fa riferimento il metodo dell'handler principale `handleRequest`.
- `func handleRequest(ctx context.Context, event json.RawMessage) error {}`: questo è il metodo dell'handler principale, che contiene la logica principale dell'applicazione.
- `func main() {}`: questo è un punto di ingresso obbligatorio per il tuo handler Lambda. L'argomento del metodo `lambda.Start()` è il metodo principale dell'handler.

Affinché questa funzione funzioni correttamente, il suo [ruolo di esecuzione](#) deve consentire l'`s3:PutObject` azione. Inoltre, assicuratevi di definire la variabile di ambiente `RECEIPT_BUCKET`. Dopo una chiamata riuscita, il bucket Amazon S3 dovrebbe contenere un file di ricevuta.

## Convenzioni di denominazione dei gestori

Per le funzioni Lambda in Go, puoi usare qualsiasi nome per l'handler. In questo esempio, il nome del metodo dell'handler è `handleRequest`. Per fare riferimento al valore del gestore nel codice, puoi usare la variabile di ambiente `_HANDLER`.

Per le funzioni Go implementate mediante un [pacchetto di implementazione .zip](#), il file eseguibile che contiene il codice della funzione deve essere denominato `bootstrap`. Il file `bootstrap` deve trovarsi nella posizione root del file `.zip`. Per le funzioni Go implementate mediante una [immagine del container](#), per il file eseguibile è possibile usare qualsiasi nome.

## Definizione e accesso all'oggetto evento di input

JSON è il formato di input più comune e standard per le funzioni Lambda. In questo esempio, la funzione prevede un input simile a quanto segue:

```
{
  "order_id": "12345",
  "amount": 199.99,
  "item": "Wireless Headphones"
}
```

Quando si utilizzano le funzioni Lambda in Go, è possibile definire la forma dell'evento di input previsto come struttura Go. In questo esempio, definiamo una struttura per rappresentare un `Order`:

```
type Order struct {
  OrderID string `json:"order_id"`
  Amount  float64 `json:"amount"`
  Item    string `json:"item"`
}
```

Questa struttura corrisponde alla forma di input prevista. Dopo aver definito la struttura, puoi scrivere una firma del gestore che accetti un tipo JSON generico compatibile con la libreria standard.

[encoding/json](#) [Puoi quindi deserializzarlo nella tua struttura usando la funzione `func Unmarshal`](#). Ciò è illustrato nelle prime righe dell'handler:

```
func handleRequest(ctx context.Context, event json.RawMessage) error {
  // Parse the input event
  var order Order
  if err := json.Unmarshal(event, &order); err != nil {
    log.Printf("Failed to unmarshal event: %v", err)
  }
}
```

```
        return err
    ...
}
```

Dopo questa deserializzazione, è possibile accedere ai campi della variabile. `order` Ad esempio, `order.OrderID` recupera il valore di `"order_id"` dall'input originale.

#### Note

Il `encoding/json` pacchetto può accedere solo ai campi esportati. Affinché siano esportati, i nomi dei campi nella struttura dell'evento devono avere l'iniziale maiuscola.

## Accesso e utilizzo dell'oggetto contestuale Lambda

L'[oggetto contesto](#): contiene informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. In questo esempio, abbiamo dichiarato questa variabile come `ctx` nella firma dell'handler:

```
func handleRequest(ctx context.Context, event json.RawMessage) error {
    ...
}
```

L'input `ctx context.Context` è un argomento facoltativo nell'handler della funzione. Per ulteriori informazioni sulle firme dell'handler accettate, consulta [the section called “Firme dell'handler valide per gli handler Go”](#).

Se si effettuano chiamate ad altri servizi utilizzando l' AWS SDK, l'oggetto context è richiesto in alcune aree chiave. Ad esempio, per inizializzare correttamente i client SDK, è possibile caricare la configurazione AWS SDK corretta utilizzando l'oggetto di contesto come segue:

```
// Load AWS SDK configuration using the default credential provider chain
cfg, err := config.LoadDefaultConfig(ctx)
```

Le chiamate SDK stesse potrebbero richiedere l'oggetto context come input. Ad esempio, la `s3Client.PutObject` chiamata accetta l'oggetto context come primo argomento:

```
// Upload the receipt to S3
_, err = s3Client.PutObject(ctx, &s3.PutObjectInput{
```

```
} ) ...
```

Al di fuori delle richieste AWS SDK, puoi anche utilizzare l'oggetto `context` per il monitoraggio delle funzioni. Per ulteriori informazioni sulla copia di oggetti, consulta la sezione [the section called “Contesto”](#).

Firme dell'handler valide per gli handler Go

Durante la creazione di un gestore della funzione Lambda in Go sono disponibili diverse opzioni, ma è necessario attenersi alle seguenti regole:

- Il gestore deve essere una funzione.
- Il gestore può accettare da 0 a 2 argomenti. Nel caso di due argomenti, il primo argomento deve implementare `context.Context`.
- Il gestore può restituire da 0 a 2 argomenti. Nel caso di un singolo valore restituito, deve implementare `error`. Nel caso di due valori restituiti, il secondo valore deve implementare `error`.

Di seguito sono elencate le firme valide del gestore. `TIn` e `TOut` rappresentano tipi compatibili con la libreria `encoding/json` standard. Per ulteriori informazioni su come deserializzare queste tipologie, consultare [func Unmarshal](#).

- `func ()`
- `func () error`
- `func () (TOut, error)`
- `func (TIn) error`
- `func (TIn) (TOut, error)`
- `func (context.Context) error`
- `func (context.Context) (TOut, error)`
- `func (context.Context, TIn) error`

- `func (context.Context, TIn) (TOut, error)`

## Utilizzo di AWS SDK per Go v2 nel tuo gestore

Spesso, utilizzerai le funzioni Lambda per interagire o apportare aggiornamenti ad altre risorse. AWS Il modo più semplice per interfacciarsi con queste risorse è usare la [AWS SDK per Go v2](#).

### Note

La AWS SDK per Go (v1) è in modalità manutenzione e terminerà il supporto il 31 luglio 2025. Ti consigliamo di utilizzare solo la versione v2 d'ora in poi. AWS SDK per Go

Per aggiungere dipendenze SDK alla tua funzione, usa il `go get` comando per i client SDK specifici di cui hai bisogno. Nel codice di esempio precedente, abbiamo usato la libreria `config` e la libreria `s3`. Aggiungi queste dipendenze eseguendo i seguenti comandi nella directory che contiene i file `go.mod` e `main.go` :

```
go get github.com/aws/aws-sdk-go-v2/config
go get github.com/aws/aws-sdk-go-v2/service/s3
```

Quindi, importa le dipendenze di conseguenza nel blocco di importazione della tua funzione:

```
import (
    ...
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)
```

Quando utilizzi l'SDK nel tuo handler, configura i tuoi client con le impostazioni corrette. Il modo più semplice per farlo è utilizzare la catena di provider di [credenziali predefinita](#). Questo esempio illustra un modo per caricare questa configurazione:

```
// Load AWS SDK configuration using the default credential provider chain
cfg, err := config.LoadDefaultConfig(ctx)
if err != nil {
    log.Printf("Failed to load AWS SDK config: %v", err)
    return err
}
```

```
}
```

Dopo aver caricato questa configurazione nella variabile `cfg`, è possibile passare la variabile nelle istanze del client. Il codice di esempio crea un client Amazon S3 come segue:

```
// Create an S3 client
s3Client := s3.NewFromConfig(cfg)
```

In questo esempio, abbiamo inizializzato il nostro client Amazon S3 nella `init()` funzione per evitare di doverlo inizializzare ogni volta che richiamiamo la nostra funzione. Il problema è che nella `init()` funzione, Lambda non ha accesso all'oggetto `context`. Come soluzione alternativa, puoi inserire un segnaposto come `context.TODO()` durante la fase di inizializzazione. Successivamente, quando effettui una chiamata utilizzando il client, inserisci l'oggetto contestuale completo. Questa soluzione alternativa è descritta anche in [the section called “Utilizzando il contesto in AWS Inizializzazioni e chiamate del client SDK”](#).

Dopo aver configurato e inizializzato il client SDK, puoi utilizzarlo per interagire con altri servizi. AWS Il codice di esempio richiama l'API `PutObject` Amazon S3 nel modo seguente:

```
_, err = s3Client.PutObject(ctx, &s3.PutObjectInput{
    Bucket: &bucketName,
    Key:    &key,
    Body:   strings.NewReader(receiptContent),
})
```

## Accesso alle variabili d'ambiente

Nel codice dell'handler, puoi fare riferimento a qualsiasi [variabile di ambiente](#) utilizzando il metodo `os.Getenv()`. In questo esempio, facciamo riferimento alla variabile di `RECEIPT_BUCKET` ambiente definita utilizzando la seguente riga di codice:

```
// Access environment variables
bucketName := os.Getenv("RECEIPT_BUCKET")
if bucketName == "" {
    log.Printf("RECEIPT_BUCKET environment variable is not set")
    return fmt.Errorf("missing required environment variable RECEIPT_BUCKET")
}
```



## Utilizzo dello stato globale

Per evitare di creare nuove risorse ogni volta che richiami la funzione, puoi dichiarare e modificare le variabili globali all'esterno del codice dell'handler della funzione Lambda. Queste variabili globali vengono definite in un blocco o in un'istruzione. Inoltre, il gestore può dichiarare una `init()` funzione che viene eseguita durante la [fase di inizializzazione](#). Il `init` metodo si comporta allo stesso modo dei AWS Lambda programmi Go standard.

### Procedure consigliate di codice per le funzioni Go Lambda

Segui le linee guida riportate nell'elenco seguente per utilizzare le best practice di codifica durante la creazione delle funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. Con questo approccio, è possibile creare una funzione più testabile in unità.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del [contesto di esecuzione](#).
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. Ciò riduce il tempo necessario per scaricare e decomprimere il pacchetto di distribuzione prima dell'invocazione.

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva `keep-alive` per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva `keep-alive` associata al runtime. Per un esempio, vedi [Riutilizzo delle connessioni con in. Keep-Alive Node.js](#)

Utilizzare [le variabili di ambiente](#) per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su 0 per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionali alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha una dipendenza su queste API non pubbliche. Consulta il [riferimento all'API](#) per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta [Come posso rendere idempotente la mia funzione Lambda?](#).

## Utilizzo dell'oggetto contestuale Lambda per recuperare le informazioni sulla funzione Go

Questo oggetto fornisce i metodi e le proprietà con informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. Quando Lambda esegue la funzione, passa un oggetto Context al [gestore](#). Per utilizzare l'oggetto context nel tuo handler, puoi facoltativamente dichiararlo come parametro di input per il tuo handler. L'oggetto contesto è necessario se desideri eseguire le operazioni indicate di seguito nell'handler:

- È necessario accedere a tutte le [variabili, i metodi o le proprietà globali](#) offerti dall'oggetto context. Questi metodi e proprietà sono utili per attività come determinare l'entità che ha richiamato la funzione o misurare il tempo di invocazione della funzione, come illustrato in [the section called “Accesso alle informazioni relative al contesto di invocazione”](#)
- È necessario utilizzare il AWS SDK per Go per effettuare chiamate verso altri servizi. L'oggetto context è un parametro di input importante per la maggior parte di queste chiamate. Per ulteriori informazioni, consulta [the section called “Utilizzando il contesto in AWS Inizializzazioni e chiamate del client SDK”](#).

### Argomenti

- [Variabili supportate, metodi e proprietà nell'oggetto contesto](#)
- [Accesso alle informazioni relative al contesto di invocazione](#)
- [Utilizzando il contesto in AWS Inizializzazioni e chiamate del client SDK](#)

### Variabili supportate, metodi e proprietà nell'oggetto contesto

La libreria contesto Lambda offre le variabili globali, i metodi e le proprietà indicate di seguito.

#### Variabili globali

- `FunctionName`: il nome della funzione Lambda.
- `FunctionVersion`: la [versione](#) della funzione.
- `MemoryLimitInMB`: la quantità di memoria allocata per la funzione.
- `LogGroupName`: il gruppo di log per la funzione.
- `LogStreamName`: il flusso di log per l'istanza della funzione.

## Metodi del contesto

- **Deadline**: restituisce la data del timeout dell'esecuzione in millisecondi Unix.

## Proprietà del contesto

- **InvokedFunctionArn**: l'ARN (Amazon Resource Name) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- **AwsRequestID**: l'identificatore della richiesta di invocazione.
- **Identity**: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
- **ClientContext**: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.

## Accesso alle informazioni relative al contesto di invocazione

Le funzioni Lambda hanno accesso ai metadata relativi al loro ambiente e alla richiesta di invocazione. È possibile accedervi mediante il [Package context](#). Qualora il gestore includa `context.Context` come parametro, Lambda inserirà le informazioni relative alla funzione nella proprietà `Value` del contesto. È necessario importare la libreria `lambdacontext` per accedere ai contenuti dell'oggetto `context.Context`.

```
package main

import (
    "context"
    "log"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-lambda-go/lambdacontext"
)

func CognitoHandler(ctx context.Context) {
    lc, _ := lambdacontext.FromContext(ctx)
    log.Print(lc.Identity.CognitoIdentityPoolID)
}

func main() {
    lambda.Start(CognitoHandler)
}
```

Nell'esempio precedente, `lc` è la variabile utilizzata per consumare le informazioni acquisite dall'oggetto di contesto e `log.Print(lc.Identity.CognitoIdentityPoolID)` stampa tali informazioni, in questo caso, il `CognitoIdentityPoolID`.

L'esempio che segue illustra come utilizzare l'oggetto contesto per monitorare il tempo impiegato per il completamento della funzione Lambda. In questo modo, è possibile analizzare le aspettative di prestazioni e modificare di conseguenza il codice della funzione, se necessario.

```
package main

import (
    "context"
    "log"
    "time"
    "github.com/aws/aws-lambda-go/lambda"
)

func LongRunningHandler(ctx context.Context) (string, error) {

    deadline, _ := ctx.Deadline()
    deadline = deadline.Add(-100 * time.Millisecond)
    timeoutChannel := time.After(time.Until(deadline))

    for {

        select {

            case <- timeoutChannel:
                return "Finished before timing out.", nil

            default:
                log.Print("hello!")
                time.Sleep(50 * time.Millisecond)
        }
    }
}

func main() {
    lambda.Start(LongRunningHandler)
}
```

## Utilizzando il contesto in AWS Inizializzazioni e chiamate del client SDK

Se il tuo gestore deve usare il per effettuare chiamate AWS SDK per Go ad altri servizi, includi l'oggetto `context` come input per il tuo gestore. In AWS, è consigliabile passare l'oggetto di contesto nella maggior parte delle chiamate AWS SDK. Ad esempio, la `PutObject` chiamata Amazon S3 accetta l'oggetto `context (ctx)` come primo argomento:

```
// Upload an object to S3
_, err = s3Client.PutObject(ctx, &s3.PutObjectInput{
    ...
})
```

Per inizializzare correttamente i client SDK, puoi anche utilizzare l'oggetto del contesto per caricare la configurazione corretta prima di passare l'oggetto di configurazione al client:

```
// Load AWS SDK configuration using the default credential provider chain
cfg, err := config.LoadDefaultConfig(ctx)
...
s3Client = s3.NewFromConfig(cfg)
```

Se desideri inizializzare i client SDK al di fuori dell'handler principale (ad esempio durante la fase di inizializzazione), puoi passare un oggetto del contesto segnaposto:

```
func init() {
    // Initialize the S3 client outside of the handler, during the init phase
    cfg, err := config.LoadDefaultConfig(context.TODO())
    ...
    s3Client = s3.NewFromConfig(cfg)
}
```

Se inizi i client in questo modo, assicurati di inserire l'oggetto contestuale corretto nelle chiamate SDK dall'handler principale.

## Distribuisci funzioni Lambda per Go con gli archivi di file .zip

Il codice della tua AWS Lambda funzione è costituito da script o programmi compilati e dalle relative dipendenze. Utilizza un pacchetto di implementazione per distribuire il codice della funzione a Lambda. Lambda supporta due tipi di pacchetti di implementazione: immagini di container e archivi di file .zip.

Questa pagina descrive come creare un file .zip come pacchetto di distribuzione per il runtime Go e quindi utilizzare il file .zip per distribuire il codice funzione AWS Lambda utilizzando Console di gestione AWS, AWS Command Line Interface () e ().AWS CLI AWS Serverless Application Model AWS SAM

Nota che Lambda utilizza i permessi dei file POSIX, quindi potrebbe essere necessario [impostare le autorizzazioni per la cartella del pacchetto di distribuzione prima di creare l'archivio di file .zip](#).

### Sections

- [Creazione di un file .zip su macOS e Linux](#)
- [Creazione di un file .zip su Windows](#)
- [Creazione e aggiornamento delle funzioni Lambda di Go utilizzando file .zip](#)

### Creazione di un file .zip su macOS e Linux

I passaggi seguenti mostrano come compilare il file eseguibile utilizzando il comando `go build` e creare un pacchetto di implementazione con file .zip per Lambda. Prima di compilare il codice, assicurati di aver installato il pacchetto lambda da <https://github.com/aws/aws-lambda-go/tree/master/lambda> GitHub Questo modulo fornisce un'implementazione dell'interfaccia di runtime, che gestisce l'interazione tra Lambda e il codice della funzione. Per scaricare questa libreria, esegui il comando seguente.

```
go get github.com/aws/aws-lambda-go/lambda
```

Se la tua funzione utilizza il AWS SDK per Go, scarica il set standard di moduli SDK, insieme a tutti i client API di AWS servizio richiesti dalla tua applicazione. Per informazioni su come installare l'SDK for Go, vedi [Guida introduttiva alla AWS SDK per Go V2](#).

## Utilizzo della famiglia di runtime forniti

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato nativamente in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un [OS-only runtime](#) (la famiglia provided di runtime) per distribuire le funzioni Go su Lambda.

Per creare un pacchetto di distribuzione .zip () macOS/Linux

1. Nella directory del progetto contenente il file `main.go` dell'applicazione, compila il tuo eseguibile. Tenere presente quanto segue:

- L'eseguibile deve essere denominato `bootstrap`. Per ulteriori informazioni, consulta [Convenzioni di denominazione dei gestori](#).
- Imposta l'architettura del set di [istruzioni di destinazione](#). OS-only i runtime supportano sia `arm64` che `x86_64`.
- Puoi usare il tag facoltativo `lambda.norpc` per escludere il componente Remote Procedure Call (RPC) della libreria [lambda](#). Il componente RPC è richiesto solo quando si utilizza il runtime Go 1.x obsoleto. L'esclusione dell'RPC riduce le dimensioni del pacchetto di implementazione.

Per l'architettura `arm64`:

```
GOOS=linux GOARCH=arm64 go build -tags lambda.norpc -o bootstrap main.go
```

Per l'architettura `x86_64`:

```
GOOS=linux GOARCH=amd64 go build -tags lambda.norpc -o bootstrap main.go
```

2. (Opzionale) Potrebbe essere necessario compilare pacchetti con `CGO_ENABLED=0` impostato su Linux:

```
GOOS=linux GOARCH=arm64 CGO_ENABLED=0 go build -o bootstrap -tags lambda.norpc main.go
```

Questo comando crea un pacchetto binario stabile per le versioni standard della libreria C (`libc`) che possono essere diverse su Lambda e su altri dispositivi.

3. Crea un pacchetto di distribuzione comprimendo l'eseguibile in un file .zip.



```
zip myFunction.zip bootstrap
```

#### Note

Il file bootstrap deve trovarsi nella posizione root del file .zip.

#### 4. Crea la funzione . Tenere presente quanto segue:

- Il file binario deve essere denominato bootstrap ma il nome del gestore può essere qualsiasi cosa. Per ulteriori informazioni, consulta [Convenzioni di denominazione dei gestori](#).
- L'opzione `--architectures` è necessaria solo se si utilizza arm64. Il valore predefinito è x86\_64.
- Per `--role`, specifica il nome della risorsa Amazon (ARN) del [ruolo di esecuzione](#).

```
aws lambda create-function --function-name myFunction \  
--runtime provided.al2023 --handler bootstrap \  
--architectures arm64 \  
--role arn:aws:iam::111122223333:role/lambda-ex \  
--zip-file fileb://myFunction.zip
```

#### Creazione di un file .zip su Windows

I passaggi seguenti mostrano come scaricare lo [strumento](#) build-lambda-zip per Windows da GitHub, compilare il file eseguibile e creare un pacchetto di distribuzione .zip.

#### Note

Se non è già presente, è necessario installare [git](#) e quindi aggiungere il percorso dell'eseguibile git alla variabile di ambiente di Windows %PATH%.

Prima di compilare il codice, assicurati di aver installato la libreria lambda da. <https://github.com/aws/aws-lambda-go/tree/master/lambda> GitHub Per scaricare questa libreria, esegui il comando seguente.

```
go get github.com/aws/aws-lambda-go/lambda
```

Se la tua funzione utilizza il AWS SDK per Go, scarica il set standard di moduli SDK, insieme a tutti i client API di AWS servizio richiesti dalla tua applicazione. Per informazioni su come installare l'SDK for Go, vedi [Guida introduttiva alla AWS SDK per Go V2](#).

## Utilizzo della famiglia di runtime forniti

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato nativamente in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un [OS-only runtime](#) (la famiglia provided di runtime) per distribuire le funzioni Go su Lambda.

## Creazione di un pacchetto di implementazione .zip (Windows)

1. Scarica lo strumento `build-lambda-zip` da GitHub

```
go install github.com/aws/aws-lambda-go/cmd/build-lambda-zip@latest
```

2. Utilizza lo strumento dal tuo GOPATH per creare un file .zip. Se si dispone di un'installazione predefinita di Go, lo strumento si trova solitamente in `%USERPROFILE%\Go\bin`. Altrimenti, vai alla posizione in cui è stato installato il runtime Go ed esegui una delle seguenti operazioni:

`cmd.exe`

In `cmd.exe`, esegui una delle seguenti operazioni, a seconda dell'architettura del set di istruzioni di destinazione. [??? OS-only](#) i runtime supportano sia `arm64` che `x86_64`.

Puoi usare il tag facoltativo `lambda.norpc` per escludere il componente Remote Procedure Call (RPC) della libreria [lambda](#). Il componente RPC è richiesto solo quando si utilizza il runtime Go 1.x obsoleto. L'esclusione dell'RPC riduce le dimensioni del pacchetto di implementazione.

Example- Per l'architettura `x86_64`

```
set GOS=linux
set GOARCH=amd64
set CGO_ENABLED=0
go build -tags lambda.norpc -o bootstrap main.go
%USERPROFILE%\Go\bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

Example- Per l'architettura `arm64`

```
set GOS=linux
```

```
set GOARCH=arm64
set CGO_ENABLED=0
go build -tags lambda.norpc -o bootstrap main.go
%USERPROFILE%\Go\bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

## PowerShell

In PowerShell, esegui una delle seguenti operazioni, a seconda dell'architettura del set di istruzioni di destinazione. [??? OS-only](#) i runtime supportano sia arm64 che x86\_64.

Puoi usare il tag facoltativo `lambda.norpc` per escludere il componente Remote Procedure Call (RPC) della libreria [lambda](#). Il componente RPC è richiesto solo quando si utilizza il runtime Go 1.x obsoleto. L'esclusione dell'RPC riduce le dimensioni del pacchetto di implementazione.

Per l'architettura x86\_64:

```
$env:GOOS = "linux"
$env:GOARCH = "amd64"
$env:CGO_ENABLED = "0"
go build -tags lambda.norpc -o bootstrap main.go
~\Go\Bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

Per l'architettura arm64:

```
$env:GOOS = "linux"
$env:GOARCH = "arm64"
$env:CGO_ENABLED = "0"
go build -tags lambda.norpc -o bootstrap main.go
~\Go\Bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

### 3. Crea la funzione . Tenere presente quanto segue:

- Il file binario deve essere denominato `bootstrap` ma il nome del gestore può essere qualsiasi cosa. Per ulteriori informazioni, consulta [Convenzioni di denominazione dei gestori](#).
- L'opzione `--architectures` è necessaria solo se si utilizza arm64. Il valore predefinito è `x86_64`.
- Per `--role`, specifica il nome della risorsa Amazon (ARN) del [ruolo di esecuzione](#).

```
aws lambda create-function --function-name myFunction \
--runtime provided.al2023 --handler bootstrap \
--architectures arm64 \
--role arn:aws:iam::111122223333:role/lambda-ex \
--zip-file fileb://myFunction.zip
```

## Creazione e aggiornamento delle funzioni Lambda di Go utilizzando file .zip

Dopo aver creato il pacchetto di implementazione .zip, puoi utilizzarlo per creare una nuova funzione Lambda o aggiornarne una esistente. Puoi distribuire il tuo pacchetto .zip utilizzando la console Lambda, l'API e Lambda. AWS Command Line Interface Puoi anche creare e aggiornare le funzioni Lambda usando AWS Serverless Application Model (AWS SAM) e CloudFormation.

La dimensione massima per un pacchetto di implementazione .zip per Lambda è di 250 MB (dopo l'estrazione). Nota che questo limite si applica alla dimensione combinata di tutti i file caricati, inclusi eventuali livelli Lambda.

Il runtime Lambda necessita dell'autorizzazione per leggere i file nel pacchetto di distribuzione. Nella notazione ottale delle autorizzazioni di Linux, Lambda richiede 644 autorizzazioni per i file non eseguibili (rw-r--r--) e 755 autorizzazioni (rwxr-xr-x) per le directory e i file eseguibili.

In Linux e macOS, utilizza il comando `chmod` per modificare le autorizzazioni file su file e directory nel pacchetto di implementazione. Ad esempio, per assegnare a un file non eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 644 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta [Set, View, Change, or Remove Permissions on an Object](#) nella documentazione di Microsoft Windows.

### Note

Se non concedi a Lambda le autorizzazioni necessarie per accedere alle directory del pacchetto di implementazione, Lambda imposta le autorizzazioni per tali directory su 755 (rwxr-xr-x).

## Creazione e aggiornamento delle funzioni con file .zip utilizzando la console

Per creare una nuova funzione, devi prima creare la funzione nella console, quindi devi caricare il tuo archivio .zip. Per aggiornare una funzione esistente, apri la pagina relativa alla funzione, quindi segui la stessa procedura per aggiungere il file .zip aggiornato.

Se il file .zip ha dimensioni inferiori a 50 MB, è possibile creare o aggiornare una funzione caricando il file direttamente dal computer locale. Per i file .zip di dimensioni superiori a 50 MB, prima è necessario caricare il pacchetto in un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il Console di gestione AWS, consulta [Guida introduttiva ad Amazon S3](#). Per caricare file utilizzando il AWS CLI, consulta [Sposta oggetti](#) nella Guida per l'AWS CLI utente.

### Note

Non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio .zip. È necessario creare una nuova funzione.

## Creazione di una nuova funzione (console)

1. Apri la [pagina Funzioni](#) della console Lambda e scegli Crea funzione.
2. Scegli Author from scratch (Crea da zero).
3. In Basic information (Informazioni di base) eseguire queste operazioni:
  - a. In Nome funzione, inserisci il nome della funzione.
  - b. In Runtime, selezionare `provided.al2023`.
4. (Opzionale) In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito. Puoi creare un nuovo ruolo di esecuzione o utilizzare un ruolo esistente.
5. Scegli Crea funzione. Lambda crea una funzione di base "Hello world" utilizzando il runtime scelto.

## Caricamento di un archivio .zip dal computer locale (console)

1. Nella [pagina Funzioni](#) della console Lambda, scegli la funzione per cui vuoi caricare il file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.

4. Scegli File .zip.
5. Per caricare il file .zip, procedi come segue:
  - a. Seleziona Carica, quindi seleziona il tuo file .zip nel selettore di file.
  - b. Seleziona Apri.
  - c. Scegli Save (Salva).

#### Caricamento di un archivio .zip da un bucket Amazon S3 (console)

1. Nella [pagina Funzioni](#) della console Lambda, scegli la funzione per cui vuoi caricare un nuovo file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli Posizione Amazon S3.
5. Incolla l'URL del link Amazon S3 del tuo file .zip e scegli Salva.

#### Creazione e aggiornamento di funzioni con file .zip utilizzando AWS CLI

È possibile utilizzare la [AWS CLI](#) per creare una nuova funzione o aggiornare una funzione esistente mediante un file .zip. Utilizza i comandi [create-function](#) e [update-function-code](#) per implementare il tuo pacchetto .zip. Se il file .zip ha dimensioni inferiori a 50 MB, è possibile caricare il pacchetto .zip da una posizione di file nella macchina di compilazione locale. Per i file di dimensioni maggiori, è necessario caricare il pacchetto .zip da un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il AWS CLI, consulta [Move objects](#) nella Guida per l'AWS CLI utente.

#### Note

Se carichi il tuo file .zip da un bucket Amazon S3 utilizzando il AWS CLI, il bucket deve trovarsi nella stessa posizione della tua funzione. Regione AWS

Per creare una nuova funzione utilizzando un file .zip con AWS CLI, devi specificare quanto segue:

- Il nome della funzione (`--function-name`)
- Il runtime della tua funzione (`--runtime`)

- Il nome della risorsa Amazon (ARN) del [ruolo di esecuzione](#) della funzione (--role)
- Il nome del metodo del gestore nel codice della funzione (--handler)

È inoltre necessario specificare la posizione del file .zip. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione --zip-file per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda create-function --function-name myFunction \  
--runtime provided.al2023 --handler bootstrap \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza l'opzione --code illustrata nel seguente comando di esempio. È necessario utilizzare il parametro S3ObjectVersion solo per gli oggetti con controllo delle versioni.

```
aws lambda create-function --function-name myFunction \  
--runtime provided.al2023 --handler bootstrap \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--code S3Bucket=amzn-s3-demo-  
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Per aggiornare una funzione esistente mediante la CLI, specifica il nome della funzione utilizzando il parametro --function-name. È inoltre necessario specificare la posizione del file .zip che desideri utilizzare per aggiornare il codice della funzione. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione --zip-file per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda update-function-code --function-name myFunction \  
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza le opzioni --s3-bucket e --s3-key come illustrato nel seguente comando di esempio. È necessario utilizzare il parametro --s3-object-version solo per gli oggetti con controllo delle versioni.

```
aws lambda update-function-code --function-name myFunction \  
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject  
Version
```

## Creazione e aggiornamento delle funzioni con file .zip utilizzando l'API Lambda

Per creare e aggiornare le funzioni mediante un archivio di file .zip, utilizza le seguenti operazioni API:

- [CreateFunction](#)
- [UpdateFunctionCode](#)

## Creazione e aggiornamento di funzioni con file .zip utilizzando AWS SAM

AWS Serverless Application Model (AWS SAM) è un toolkit che aiuta a semplificare il processo di creazione ed esecuzione di applicazioni serverless su AWS. Definisci le risorse per la tua applicazione in un modello YAML o JSON e usi l'interfaccia a riga di AWS SAM comando (AWS SAM CLI) per creare, impacchettare e distribuire le tue applicazioni. Quando crei una funzione Lambda da un AWS SAM modello, crea AWS SAM automaticamente un pacchetto di distribuzione .zip o un'immagine del contenitore con il codice della funzione e le dipendenze specificate. Per saperne di più sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda, consulta la [Guida introduttiva nella Developer Guide. AWS SAM](#) AWS Serverless Application Model

Puoi anche usare AWS SAM per creare una funzione Lambda utilizzando un archivio di file .zip esistente. Per creare una funzione Lambda utilizzando AWS SAM, puoi salvare il tuo file .zip in un bucket Amazon S3 o in una cartella locale sulla macchina di compilazione. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il AWS CLI, consulta [Move objects nella Guida per l'utente.](#) AWS CLI

Nel AWS SAM modello, la `AWS::Serverless::Function` risorsa specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `CodeUri`: impostato sull'URI Amazon S3, sul percorso della cartella locale o sull'oggetto del codice funzione [FunctionCode](#)
- `Runtime`: imposta il runtime prescelto

Con AWS SAM, se il tuo file .zip è più grande di 50 MB, non devi prima caricarlo su un bucket Amazon S3. AWS SAM puoi caricare pacchetti .zip fino alla dimensione massima consentita di 250 MB (decompressi) da una posizione sul computer di compilazione locale.



Per saperne di più sulla distribuzione delle funzioni utilizzando il file .zip in AWS SAM, consulta [AWS::Serverless::Function nella Developer Guide](#). AWS SAM

Esempio: utilizzo AWS SAM per creare una funzione Go con provided.al2023

1. Crea un AWS SAM modello con le seguenti proprietà:
  - BuildMethod: specifica il compilatore per l'applicazione. Utilizza go1.x.
  - Runtime: utilizza provided.al2023.
  - CodeUri: inserisci il percorso del codice.
  - Architetture: usa [arm64] per l'architettura arm64. Per l'architettura del set di istruzioni x86\_64, utilizza [amd64] oppure rimuovi la proprietà Architectures.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: go1.x
    Properties:
      CodeUri: hello-world/ # folder where your main program resides
      Handler: bootstrap
      Runtime: provided.al2023
      Architectures: [arm64]
```

2. Usa il comando [sam build](#) per compilare l'eseguibile.

```
sam build
```

3. Utilizza il comando [sam deploy](#) per implementare la funzione su Lambda.

```
sam deploy --guided
```

## Creazione e aggiornamento di funzioni con file .zip utilizzando CloudFormation

È possibile utilizzare CloudFormation per creare una funzione Lambda utilizzando un archivio di file .zip. Per creare una funzione Lambda da un file .zip, devi prima caricare il file su un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando AWS CLI, consulta [Move objects](#) nella Guida per l'utente. AWS CLI

Nel CloudFormation modello, la `AWS::Lambda::Function` risorsa specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `Code`: inserisci il nome del bucket Amazon S3 e il nome del file .zip nei campi `S3Bucket` e `S3Key`
- `Runtime`: imposta il runtime prescelto

Il file .zip CloudFormation generato non può superare i 4 MB. Per saperne di più sulla distribuzione delle funzioni utilizzando il file .zip in CloudFormation, vedi [AWS::Lambda::Function](#) nella Guida per l'utente. CloudFormation

## Distribuzione delle funzioni Go Lambda con immagini di container

Esistono due modi per creare un'immagine di container per una funzione Lambda in Go:

- [Utilizzando un AWS OS-only immagine di base](#)

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato nativamente in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un'immagine di [OS-only base](#) per creare immagini Go per Lambda. Per rendere l'immagine compatibile con Lambda, devi includere il pacchetto `aws-lambda-go/lambda` nell'immagine.

- [Usare un non-AWS immagine di base](#)

È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere il pacchetto `aws-lambda-go/lambda` nell'immagine.

### Tip

Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta [Utilizzo di compilazioni a più fasi](#) nella documentazione Docker. Per creare immagini di container efficienti, segui le [best practice per scrivere file Docker](#).

Questa pagina spiega come creare, testare e implementare le immagini di container per Lambda.

### AWS immagini di base per l'implementazione delle funzioni Go

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato nativamente in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un'immagine di [OS-only base](#) per distribuire le funzioni Go su Lambda.

| Nome            | Identificatore  | Sistema operativo | Data di ritiro | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|-----------------|-----------------|-------------------|----------------|---------------------------|-------------------------------|
| OS-only Runtime | provided.al2023 | Amazon Linux 2023 | 30 giugno 2029 | 31 luglio 2029            | 31 agosto 2029                |

Galleria pubblica del registro Amazon Elastic Container: [gallery.ecr.aws/lambda/fornito](#)

Client di interfaccia di runtime per Go

Il pacchetto `aws-lambda-go/lambda` include un'implementazione dell'interfaccia di runtime. Per esempi di come utilizzare `aws-lambda-go/lambda` nell'immagine, consulta le sezioni [Utilizzando un AWS OS-only immagine di base](#) o [Usare un non-AWS immagine di base](#).

Utilizzando un AWS OS-only immagine di base

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato nativamente in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un'immagine di [OS-only base](#) per creare immagini di contenitori per le funzioni Go.

| Tag    | Runtime         | Sistema operativo | Dockerfile                                                   | Raggiunta obsolescenza |
|--------|-----------------|-------------------|--------------------------------------------------------------|------------------------|
| al2023 | OS-only Runtime | Amazon Linux 2023 | <a href="#">Dockerfile per OS-only Runtime attivo GitHub</a> | 30 giugno 2029         |

Per ulteriori informazioni su queste immagini di base, consulta la documentazione [fornita](#) nella galleria pubblica di Amazon ECR.

Devi includere il `go/lambda` [pacchetto](#) `aws-lambda-` con il tuo gestore Go. Questo pacchetto implementa il modello di programmazione per Go, inclusa l'interfaccia di runtime.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [AWS CLI versione 2](#)
- [Docker](#) (versione minima 25.0.0)
- Il [plugin buildx](#) di Docker.
- Go

## Creazione di un'immagine dall'immagine di base `provided.al2023`

Per creare e distribuire una funzione Go con l'immagine di base **`provided.al2023`**

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir hello
cd hello
```

2. Inizializza un nuovo modulo Go.

```
go mod init example.com/hello-world
```

3. Aggiungi la libreria `lambda` come dipendenza del nuovo modulo.

```
go get github.com/aws/aws-lambda-go/lambda
```

4. Crea un file denominato `main.go`, quindi aprilo in un editor di testo. Questo è il codice per la funzione Lambda. A fini di test, puoi utilizzare il codice di esempio seguente o sostituirlo con il tuo codice personalizzato.

```
package main

import (
    "context"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, event events.APIGatewayProxyRequest)
(event events.APIGatewayProxyResponse, error) {
    response := events.APIGatewayProxyResponse{
        StatusCode: 200,
        Body:       "\"Hello from Lambda!\"",
    }
    return response, nil
}

func main() {
    lambda.Start(handler)
}
```

5. Utilizza un editor di testo per creare un file Dockerfile nella directory del progetto.
- L'esempio seguente, Dockerfile utilizza una build in [più fasi sul sito Web della documentazione Docker](#). Con una build in più fasi, puoi utilizzare un'immagine di base diversa in ogni passaggio. Puoi utilizzare un'immagine, ad esempio un'immagine di base [Go](#) su Docker Hub, per compilare il codice e creare il binario eseguibile. È quindi possibile utilizzare un'immagine diversa, ad esempio `provided.al2023`, nell'istruzione FROM finale per definire l'immagine da implementare in Lambda. La compilazione in più fasi separa il processo di compilazione dall'immagine di distribuzione finale, quindi l'immagine finale contiene solo i file necessari per eseguire l'applicazione.
  - Puoi usare il tag facoltativo `lambda.norpc` per escludere il componente Remote Procedure Call (RPC) della libreria [lambda](#). Il componente RPC è richiesto solo quando si utilizza il runtime Go 1.x obsoleto. L'esclusione dell'RPC riduce le dimensioni del pacchetto di implementazione.
  - Nota che l'esempio Dockerfile non include un'istruzione [USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione USER.

#### Example— crea Dockerfile Multi-stage

##### Note

Assicurati che la versione di Go specificata nel tuo Dockerfile (ad esempio `golang:1.20`) sia la stessa versione di Go che hai usato per creare l'applicazione.

```
FROM golang:1.20 as build
WORKDIR /helloworld
# Copy dependencies list
COPY go.mod go.sum ./
# Build with optional lambda.norpc tag
COPY main.go .
RUN go build -tags lambda.norpc -o main main.go
# Copy artifacts to a clean image
FROM public.ecr.aws/lambda/provided:al2023
COPY --from=build /helloworld/main ./main
```

```
ENTRYPOINT [ "./main" ]
```

6. Crea l'immagine Docker con il comando [docker build](#). L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il [tag](#) `test`. Per rendere la tua immagine compatibile con Lambda, devi usare l'opzione. `--provenance=false`

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test .
```

#### Note

Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Per creare una funzione Lambda utilizzando l'architettura del set di istruzioni ARM64, accertati di modificare il comando per utilizzare l'opzione `--platform linux/arm64`.

### (Facoltativo) Test dell'immagine in locale

Usa il [simulatore dell'interfaccia di runtime](#) per testare l'immagine in locale. Il simulatore dell'interfaccia di runtime è incluso nell'immagine di base `provided.al2023`.

### Esecuzione del simulatore dell'interfaccia di runtime sul computer locale

1. Avvia l'immagine Docker con il comando `docker run`. Tenere presente quanto segue:
  - `docker-image` è il nome dell'immagine e `test` è il tag.
  - `./main` è l'ENTRYPOINT del Dockerfile.

```
docker run -d -p 9000:8080 \  
--entrypoint /usr/local/bin/aws-lambda-rie \  
docker-image:test ./main
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

2. Da una nuova finestra di terminale, invia un evento al seguente endpoint utilizzando un comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Alcune funzioni potrebbero richiedere un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

3. Ottieni l'ID del container.

```
docker ps
```

4. Utilizza il comando [docker kill](#) per arrestare il container. In questo comando, sostituisci 3766c4ab331c con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

## Implementazione dell'immagine

### Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Per autenticare la CLI Docker nel registro Amazon ECR, esegui il comando [get-login-password](#).
  - Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
  - Sostituisci 111122223333 con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --  
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-  
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```



 Note

Il repository Amazon ECR deve essere Regione AWS uguale alla funzione Lambda.

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando [docker tag](#) per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:
  - `docker-image:test` è il nome e [tag](#) dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
  - Sostituisci l'<ECRrepositoryUri> con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando [docker push](#) per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Crea un ruolo di esecuzione](#) per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per ImageUri, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puoi creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta [Autorizzazioni multiaccount Amazon ECR](#).

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nel repository Amazon ECR e quindi utilizzare il comando [update-function-code](#) per implementare l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per implementare la nuova immagine nella stessa funzione Lambda, devi utilizzare il comando [update-function-code](#) anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \  
  --function-name hello-world \  
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --publish
```

## Usare un non-AWS immagine di base

Puoi creare un'immagine contenitore per Go da un'immagine non di AWS base. Nei passaggi di esempio che seguono, Dockerfile utilizza un'[immagine di base Alpine](#).

Devi includere il go/lambda [pacchetto](#) `aws-lambda-` con il tuo gestore Go. Questo pacchetto implementa il modello di programmazione per Go, inclusa l'interfaccia di runtime.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [AWS CLI versione 2](#)
- [Docker](#) (versione minima 25.0.0)
- Il [plugin buildx](#) di Docker.
- Go

## Creazione di un'immagine da un'immagine di base alternativa

### Creazione e implementazione di una funzione Go con un'immagine di base Alpine

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir hello
cd hello
```

2. Inizializza un nuovo modulo Go.

```
go mod init example.com/hello-world
```

3. Aggiungi la libreria lambda come dipendenza del nuovo modulo.

```
go get github.com/aws/aws-lambda-go/lambda
```

4. Crea un file denominato `main.go`, quindi aprilo in un editor di testo. Questo è il codice per la funzione Lambda. A fini di test, puoi utilizzare il codice di esempio seguente o sostituirlo con il tuo codice personalizzato.

```
package main

import (
    "context"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, event events.APIGatewayProxyRequest)
(event events.APIGatewayProxyResponse, error) {
    response := events.APIGatewayProxyResponse{
        StatusCode: 200,
        Body:       "\"Hello from Lambda!\",",
    }
    return response, nil
}

func main() {
    lambda.Start(handler)
}
```

5. Utilizza un editor di testo per creare un file `Dockerfile` nella directory del progetto. Nell'esempio che segue, `Dockerfile` utilizza un'[immagine di base Alpine](#). Nota che l'esempio `Dockerfile` non include un'[istruzione USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi.

Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

### Example Dockerfile

#### Note

Assicurati che la versione di Go specificata nel tuo Dockerfile (ad esempio `golang:1.20`) sia la stessa versione di Go che hai usato per creare l'applicazione.

```
FROM golang:1.20.2-alpine3.16 as build
WORKDIR /helloworld
# Copy dependencies list
COPY go.mod go.sum ./
# Build
COPY main.go .
RUN go build -o main main.go
# Copy artifacts to a clean image
FROM alpine:3.16
COPY --from=build /helloworld/main /main
ENTRYPOINT [ "/main" ]
```

6. Crea l'immagine Docker con il comando [docker build](#). L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il [tag](#) `test`. Per rendere l'immagine compatibile con Lambda, devi usare l'opzione `--provenance=false`.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test .
```

#### Note

Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Per creare una funzione Lambda utilizzando l'architettura del set di istruzioni ARM64, accertati di modificare il comando per utilizzare l'opzione `--platform linux/arm64`.

## (Facoltativo) Test dell'immagine in locale

Usa il [simulatore dell'interfaccia di runtime](#) per testare l'immagine in locale. Puoi [creare l'emulatore nella tua immagine](#) o seguire la procedura riportata e installarlo sul tuo computer locale.

Installazione ed esecuzione dell'emulatore di interfaccia di runtime sul computer locale

1. Dalla directory del progetto, esegui il comando seguente per scaricare l'emulatore di interfaccia di runtime (architettura x86-64) GitHub e installarlo sul tuo computer locale.

### Linux/macOS

```
mkdir -p ~/.aws-lambda-rie && \
  curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-
runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
  chmod +x ~/.aws-lambda-rie/aws-lambda-rie
```

Per installare l'emulatore arm64, sostituite l'URL del GitHub repository nel comando precedente con il seguente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

### PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"
if (-not (Test-Path $dirPath)) {
    New-Item -Path $dirPath -ItemType Directory
}

$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/
releases/latest/download/aws-lambda-rie"
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Per installare l'emulatore arm64, sostituisci \$downloadLink con quanto segue:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Avvia l'immagine Docker con il comando `docker run`. Tenere presente quanto segue:

- `docker-image` è il nome dell'immagine e `test` è il tag.
- `/main` è l'ENTRYPOINT del Dockerfile.

## Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
9000:8080 \
  --entrypoint /aws-lambda/aws-lambda-rie \
  docker-image:test \
  /main
```

## PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
9000:8080 `
--entrypoint /aws-lambda/aws-lambda-rie `
docker-image:test `
/main
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

### Note

Per creare l'immagine Docker per l'architettura del set di istruzioni ARM64, assicurati di utilizzare l'opzione `--platform linux/arm64` anziché `--platform linux/amd64`.

3. Pubblica un evento nell'endpoint locale.

## Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

## PowerShell

In PowerShell, esegui il seguente comando: Invoke-WebRequest

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{} ' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType  
"application/json"
```

4. Ottieni l'ID del container.

```
docker ps
```

5. Utilizza il comando [docker kill](#) per arrestare il container. In questo comando, sostituisci 3766c4ab331c con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

## Implementazione dell'immagine

### Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Per autenticare la CLI Docker nel registro Amazon ECR, esegui il comando [get-login-password](#).



- Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
- Sostituisci `111122223333` con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

#### Note

Il repository Amazon ECR deve essere Regione AWS uguale alla funzione Lambda.

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copia il `repositoryUri` dall'output del passaggio precedente.

4. Esegui il comando [docker tag](#) per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:
  - `docker-image:test` è il nome e [tag](#) dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
  - Sostituisci l'<ECRrepositoryUri> con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando [docker push](#) per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Crea un ruolo di esecuzione](#) per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puoi creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta [Autorizzazioni multiaccount Amazon ECR](#).

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nel repository Amazon ECR e quindi utilizzare il comando [update-function-code](#) per implementare l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per implementare la nuova immagine nella stessa funzione Lambda, devi utilizzare il comando [update-function-code](#) anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \  
  --function-name hello-world \  
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --publish
```

## Utilizzo dei livelli per le funzioni Lambda in Go

Consigliamo di non utilizzare i livelli per gestire le dipendenze per le funzioni Lambda scritte in Go. Questo perché le funzioni Lambda in Go vengono compilate in un unico eseguibile, che viene fornito a Lambda quando si distribuisce la funzione. Questo eseguibile contiene il codice di funzione compilato, insieme a tutte le sue dipendenze. L'uso dei livelli non solo complica questo processo, ma comporta anche un aumento dei tempi di avvio a freddo, poiché le funzioni devono caricare manualmente assieme aggiuntivi in memoria durante la fase di inizializzazione.

Per utilizzare dipendenze esterne con i gestori Go, includile direttamente nel pacchetto di implementazione. In questo modo, semplifichi il processo di implementazione e sfrutti anche le ottimizzazioni integrate del compilatore Go. Per un esempio di come importare e utilizzare una dipendenza come l'SDK AWS per Go nella tua funzione, consulta [the section called “Gestore”](#).

## Registrare e monitorare le funzioni Lambda con Go

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon CloudWatch. La tua funzione Lambda include un gruppo di CloudWatch log e un flusso di log per ogni istanza della tua funzione. L'ambiente del runtime Lambda invia i dettagli su ogni richiamo al flusso di log e inoltra i log e l'output del codice della funzione. Per ulteriori informazioni, consulta [Invio dei log delle funzioni Lambda ai CloudWatch log](#).

Questa pagina descrive come produrre un output di log dal codice della funzione Lambda e accedere ai log utilizzando la AWS Command Line Interface console Lambda o la console CloudWatch.

### Sections

- [Creazione di una funzione che restituisce i registri](#)
- [Visualizzazione dei log nella console Lambda](#)
- [Visualizzazione dei log nella console CloudWatch](#)
- [Visualizzazione dei log utilizzando AWS Command Line Interface \(AWS CLI\)](#)
- [Eliminazione dei log](#)

### Creazione di una funzione che restituisce i registri

Per i log di output del codice della funzione, puoi usare i metodi del [pacchetto `fmt`](#) o qualsiasi libreria di registrazione che scriva in `stdout` o `stderr`. Nell'esempio seguente viene utilizzato [il pacchetto `log`](#).

### Example [main.go](#) — Registrazione

```
func handleRequest(ctx context.Context, event events.SQSEvent) (string, error) {
    // event
    eventJson, _ := json.MarshalIndent(event, "", "  ")
    log.Printf("EVENT: %s", eventJson)
    // environment variables
    log.Printf("REGION: %s", os.Getenv("AWS_REGION"))
    log.Println("ALL ENV VARS:")
    for _, element := range os.Environ() {
        log.Println(element)
    }
}
```

## Example Formato dei log

```
START RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71 Version: $LATEST
2020/03/27 03:40:05 EVENT: {
  "Records": [
    {
      "messageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
      "receiptHandle": "MessageReceiptHandle",
      "body": "Hello from SQS!",
      "md5fBody": "7b27xmplb47ff90a553787216d55d91d",
      "md5fMessageAttributes": "",
      "attributes": {
        "ApproximateFirstReceiveTimestamp": "1523232000001",
        "ApproximateReceiveCount": "1",
        "SenderId": "123456789012",
        "SentTimestamp": "1523232000000"
      }
    },
    ...
  ]
}
2020/03/27 03:40:05 AWS_LAMBDA_LOG_STREAM_NAME=2020/03/27/
[$LATEST]569cxmplc3c34c7489e6a97ad08b4419
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_NAME=blank-go-function-9DV3XMPL6XBC
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_MEMORY_SIZE=128
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_VERSION=$LATEST
2020/03/27 03:40:05 AWS_EXECUTION_ENV=AWS_Lambda_go1.x
END RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71
REPORT RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71 Duration: 38.66 ms Billed
Duration: 243 ms Memory Size: 128 MB Max Memory Used: 54 MB Init Duration: 203.69 ms
XRAY TraceId: 1-5e7d7595-212fxmpl9ee07c4884191322 SegmentId: 42ffxmpl0645f474 Sampled:
true
```

Il runtime di Go registra START, END e REPORT per ogni chiamata. La riga del report fornisce i seguenti dettagli.

### Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per l'invocazione.
- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.

- **Quantità max utilizzata** – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- **Durata Init** – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- **XRAY TraceId**: per le richieste tracciate, l'ID di traccia. [AWS X-Ray](#)
- **SegmentId**— Per le richieste tracciate, l'ID del X-Ray segmento.
- **Campionato** – Per le richieste tracciate, il risultato del campionamento.

### Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor di codice incorporato, i log sono contenuti nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, trovi l'output del registro nella sezione **Dettagli**.

### Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Aprire la pagina dei gruppi di [log](#) sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (`/aws/lambda/your-function-name`).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'[istanza della funzione](#). Un flusso di log viene visualizzato quando si aggiorna la funzione Lambda e quando vengono create istanze aggiuntive per gestire chiamate simultanee. Per trovare i log per una chiamata specifica, ti consigliamo di strumentare la tua funzione con AWS X-Ray. X-Ray registra i dettagli sulla richiesta e sul flusso di log nella traccia.

## Visualizzazione dei log utilizzando AWS Command Line Interface (AWS CLI)

AWS CLI È uno strumento open source che puoi utilizzare per interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della [AWS CLI versione 2](#).

È possibile utilizzare [AWS CLI](#) per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

### Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "LogResult":
    "U1RBULQgUmVxdWVzdElk0iA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
  "ExecutedVersion": "$LATEST"
}
```

### Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'base64 utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'`cli-binary-format` opzione è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:



```

START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed
Duration: 80 ms Memory Size: 128 MB Max Memory Used: 73 MB

```

L'utilità `base64` è disponibile su Linux, macOS e [Ubuntu su Windows](#). Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

### Example Script `get-logs.sh`

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'opzione `cli-binary-format` è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

```

#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"/'/g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5

```

### Example(solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il comando seguente per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

## Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\n$LATEST\n",
\r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
```

```
      "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75\n\tMB\t\n",\n      "ingestionTime": 1559763018353\n    }\n  ],\n  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",\n  "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"\n}
```

## Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o [configurare un periodo di conservazione](#) trascorso il quale i log vengono eliminati automaticamente.

## Strumentazione del codice Go AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzarlo X-Ray per tracciare una richiesta mentre attraversa le risorse dell'applicazione, che possono includere funzioni Lambda e altri servizi. AWS

Per inviare dati di tracciamento a X-Ray, puoi utilizzare una delle due librerie SDK:

- [AWS Distro for OpenTelemetry \(ADOT\)](#): una distribuzione sicura, pronta per la produzione e AWS supportata dell'SDK (OtEL). OpenTelemetry
- [AWS X-Ray SDK for Go](#): un SDK per la generazione e l'invio di dati di traccia a X-Ray

Ciascuno degli SDK offre modi per inviare i dati di telemetria al servizio. X-Ray Puoi quindi utilizzarlo X-Ray per visualizzare, filtrare e ottenere informazioni dettagliate sulle metriche delle prestazioni dell'applicazione per identificare problemi e opportunità di ottimizzazione.

### Important

The X-Ray e Powertools for AWS Lambda SDK fanno parte di una soluzione di strumentazione strettamente integrata offerta da AWS. I livelli Lambda ADOT fanno parte di uno standard di settore per la strumentazione di tracciamento che in generale raccoglie più dati, ma potrebbero non essere adatti a tutti i casi d'uso. È possibile implementare il tracciamento end-to-end utilizzando entrambe le soluzioni. X-Ray Per saperne di più sulla scelta tra di esse, consulta [Scelta tra AWS Distro for Open Telemetry e SDK. X-Ray](#)

## Sections

- [Utilizzo di ADOT per strumentare le funzioni Go](#)
- [Utilizzo dell'SDK per strumentare le tue funzioni Go X-Ray](#)
- [Attivazione del tracciamento con la console Lambda](#)
- [Attivazione del tracciamento con l'API Lambda](#)
- [Attivazione del tracciamento con CloudFormation](#)
- [Interpretazione e tracciamento X-Ray](#)

## Utilizzo di ADOT per strumentare le funzioni Go

ADOT fornisce [livelli](#) Lambda completamente gestiti che mettono insieme tutto il necessario per raccogliere i dati di telemetria utilizzando l'SDK OTel. Usando questo livello, è possibile strumentare le funzioni Lambda senza dover modificare alcun codice funzione. È inoltre possibile configurare il livello per eseguire l'inizializzazione personalizzata di OTel. Per ulteriori informazioni, consulta la sezione relativa alla [configurazione personalizzata per ADOT Collector su Lambda](#) nella documentazione di ADOT.

Per i runtime Go, puoi aggiungere il livello Lambda gestito da AWS per ADOT Go per strumentare automaticamente le tue funzioni. Per istruzioni dettagliate su come aggiungere questo layer, vedi [AWS Distro for OpenTelemetry Lambda Support for Go nella documentazione ADOT](#). Per istruzioni sulla conversione degli eventi dell'API Lambda Telemetry in intervalli, consulta. OpenTelemetry [the section called “Conversione di eventi in intervalli OTel”](#)

## Utilizzo dell'SDK per strumentare le tue funzioni Go X-Ray

Per registrare i dettagli sulle chiamate effettuate dalla funzione Lambda ad altre risorse dell'applicazione, puoi anche utilizzare l' AWS X-Ray SDK for Go. Per scaricare l'SDK, scaricalo dal suo repository con: [GitHub](#) `go get`

```
go get github.com/aws/aws-xray-sdk-go
```

Per utilizzare i client AWS SDK, passa il client al metodo. `xray.AWS( )` Quindi potrai tracciare le chiamate utilizzando la versione `WithContext` del metodo.

```
svc := s3.New(session.New())
xray.AWS(svc.Client)
...
svc.ListBucketsWithContext(ctx aws.Context, input *ListBucketsInput)
```

Dopo aver aggiunto le dipendenze corrette e apportato le modifiche al codice necessarie, attiva il tracciamento nella configurazione della funzione tramite la console Lambda o l'API.

## Attivazione del tracciamento con la console Lambda

Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

## Per attivare il tracciamento attivo

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. In Strumenti di monitoraggio aggiuntivi scegli Modifica.
5. In CloudWatch Application Signals and AWS X-Ray, scegli **Abilita** per le tracce del servizio Lambda.
6. Scegli Save (Salva).

## Attivazione del tracciamento con l'API Lambda

Configura il tracciamento sulla tua funzione Lambda con AWS CLI or AWS SDK, utilizza le seguenti operazioni API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

Il seguente AWS CLI comando di esempio abilita la traccia attiva su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

## Attivazione del tracciamento con CloudFormation

Per attivare la traccia su una `AWS::Lambda::Function` risorsa in un CloudFormation modello, utilizzate la proprietà `TracingConfig`

Example [function-inline.yml](#) — Configurazione del tracciamento

Resources:

```
function:
  Type: AWS::Lambda::Function
  Properties:
    TracingConfig:
      Mode: Active
    ...
```

Per una `AWS::Serverless::Function` risorsa AWS Serverless Application Model (AWS SAM), usa la `Tracing` proprietà.

Example [template.yml](#) — Configurazione di tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
    ...
```

## Interpretazione e tracciamento X-Ray

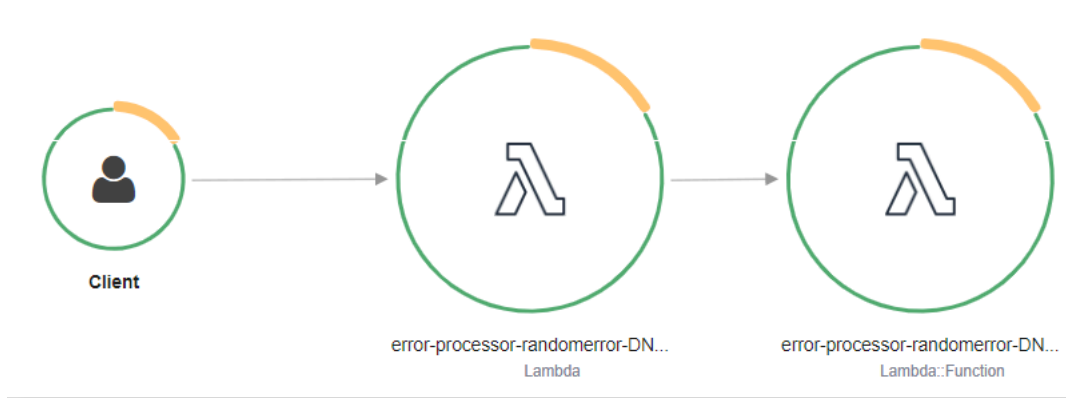
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al [ruolo di esecuzione](#) della funzione. Altrimenti, aggiungi la [AWSXRayDaemonWriteAccess](#) policy al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico del [X-Ray servizio](#) mostra le informazioni sull'applicazione e su tutti i suoi componenti. Il seguente esempio mostra un'applicazione con due funzioni. La funzione principale elabora gli eventi e talvolta restituisce errori. La seconda funzione in alto elabora gli errori che appaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon CloudWatch Logs.



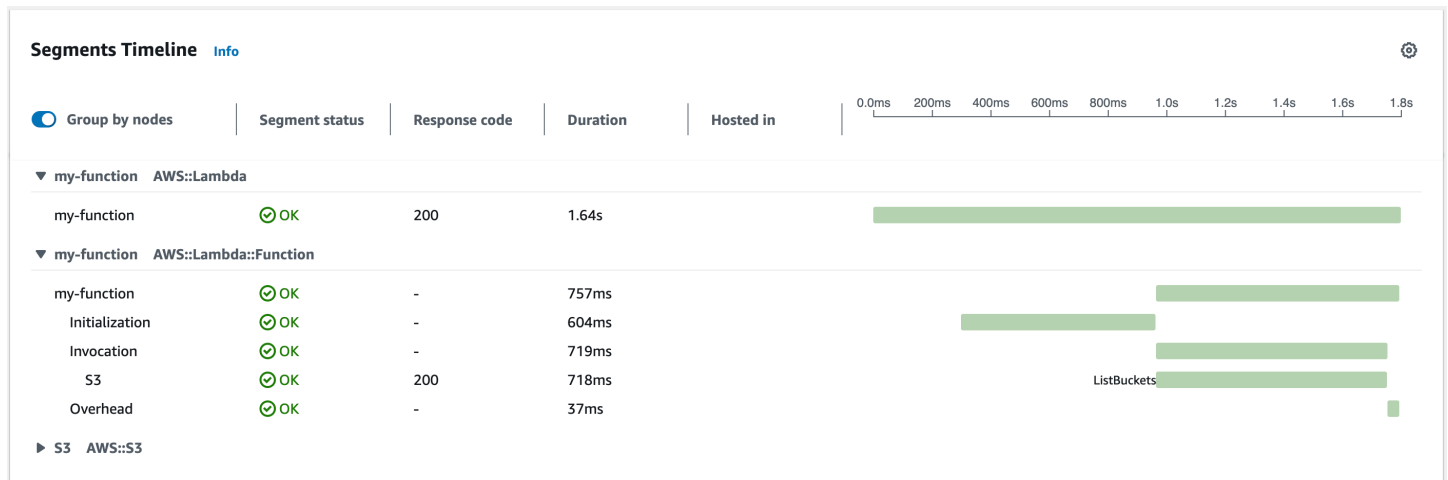
X-Ray non traccia tutte le richieste all'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento sia efficiente, fornendo comunque un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non puoi configurare la frequenza di X-Ray campionamento per le tue funzioni.

In X-Ray, una traccia registra le informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.





Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.

### Note

AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti.

Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regions, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia nel nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- **Inizializzazione** – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il [codice di inizializzazione](#). Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.
- **Chiamata**: rappresenta il tempo impiegato per eseguire il codice del gestore.
- **Overhead**: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla

struttura dei segmenti di funzioni vecchio e nuovo stile, consulta [the section called “Comprensione delle tracce X-Ray”](#).

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta [SDK AWS X-Ray per Go](#) nella Guida per gli sviluppatori di AWS X-Ray .

### Prezzi

Puoi utilizzare il X-Ray tracciamento gratuitamente ogni mese fino a un determinato limite come parte del piano gratuito. AWS Oltre tale soglia, vengono X-Ray addebitati i costi per l'archiviazione e il recupero delle tracce. Per ulteriori informazioni, consultare [Prezzi di AWS X-Ray](#).

## Compilazione di funzioni Lambda con C#

Puoi eseguire l'applicazione.NET in Lambda utilizzando il runtime .NET 8 gestito, un runtime personalizzato o un'immagine contenitore. Dopo aver compilato il codice dell'applicazione, potrai implementarlo su Lambda come file .zip o come immagine di container. Lambda fornisce i runtime seguenti per i linguaggi .NET:

Per informazioni sull'uso dei livelli.NET, consulta [the section called “Livelli”](#). Per indicazioni su come testare le funzioni C#, vedere [the section called “Test in corso”](#).

| Nome                      | Identificatore | Sistema operativo | Data di ritiro   | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|---------------------------|----------------|-------------------|------------------|---------------------------|-------------------------------|
| .NET 10                   | dotnet10       | Amazon Linux 2023 | 14 novembre 2028 | 14 dicembre 2028          | 15 gennaio 2029               |
| .NET 9 (solo contenitore) | dotnet9        | Amazon Linux 2023 | 10 novembre 2026 | Non programmato           | Non programmato               |
| .NET 8                    | dotnet8        | Amazon Linux 2023 | 10 novembre 2026 | 1 febbraio 2027           | 3 marzo 2027                  |

## Configurazione dell'ambiente di sviluppo .NET

Per sviluppare e creare le tue funzioni Lambda, puoi utilizzare uno qualsiasi degli ambienti di sviluppo integrati (IDE) .NET comunemente disponibili, tra cui Microsoft Visual Studio, Visual Studio Code e JetBrains Rider. Per semplificare l'esperienza di sviluppo, AWS fornisce un set di modelli di progetto .NET e l'interfaccia a riga di Amazon.Lambda.Tools comando (CLI).

Emetti i seguenti comandi .NET della CLI per installare questi modelli di progetto e strumenti da riga di comando.

### Installazione dei modelli di progetto .NET

Per installare i modelli di progetto, esegui il comando riportato:

```
dotnet new install Amazon.Lambda.Templates
```

### Installazione e aggiornamento degli strumenti della CLI

Esegui i comandi riportati qui di seguito per installare, aggiornare e disinstallare la CLI di Amazon.Lambda.Tools.

Per installare gli strumenti a riga di comando:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Per aggiornare gli strumenti a riga di comando:

```
dotnet tool update -g Amazon.Lambda.Tools
```

Per disinstallare gli strumenti a riga di comando:

```
dotnet tool uninstall -g Amazon.Lambda.Tools
```

## Definire l'handler della funzione Lambda in C#

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come lavorare con i gestori di funzioni Lambda in C# per operare con il runtime gestito da .NET, incluse le opzioni per la configurazione del progetto, le convenzioni di denominazione e le best practice. Questa pagina include anche un esempio di funzione Lambda C# che raccoglie informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon Simple Storage Service (S3). Per informazioni su come distribuire una funzione dopo averla scritta, consulta o [the section called “Pacchetto di implementazione”](#) [the section called “Implementazione di immagini di container”](#)

### Argomenti

- [Configurazione del progetto del gestore C#](#)
- [Esempio di codice della funzione Lambda C#](#)
- [Gestori di librerie di classi](#)
- [Gestori di assembly eseguibili](#)
- [Firme del gestore valide per le funzioni C#](#)
- [Convenzioni di denominazione dei gestori](#)
- [Serializzazione nelle funzioni Lambda C#](#)
- [File-based funzioni](#)
- [Accesso e utilizzo dell'oggetto contestuale Lambda](#)
- [Utilizzo di SDK per .NET v3 nel tuo gestore](#)
- [Accesso alle variabili d'ambiente](#)
- [Utilizzo dello stato globale](#)
- [Semplificazione del codice delle funzioni con il framework Lambda Annotations](#)
- [Best practice di codice per le funzioni Lambda con C#](#)

### Configurazione del progetto del gestore C#

Quando lavori con le funzioni Lambda in C#, il processo prevede la scrittura del codice, quindi l'implementazione del codice in Lambda. Esistono due diversi modelli di esecuzione per

l'implementazione delle funzioni Lambda in .NET: l'approccio della libreria di classi e l'approccio dell'assembly eseguibile.

Quando segui l'approccio della libreria di classi, impacchetti il codice della funzione come assembly .NET (.dll) e lo implementi in Lambda con il runtime gestito da .NET (dotnet8). Per il nome del gestore, Lambda si aspetta una stringa nel formato `AssemblyName::Namespace.Classname::Methodname`. Durante la fase di inizializzazione della funzione, la classe della funzione viene inizializzata e viene eseguito qualsiasi codice nel costruttore.

Quando segui l'approccio dell'assembly eseguibile, utilizzi la [funzionalità delle istruzioni di primo livello](#) introdotta per la prima volta in C# 9. Questo approccio genera un assembly eseguibile che Lambda esegue ogni volta che riceve un comando invoke per la tua funzione. Con questo approccio, utilizzi anche il runtime gestito da .NET (dotnet8). Per il nome del gestore, fornisci a Lambda il nome dell'assembly eseguibile da eseguire.

L'esempio principale in questa pagina illustra l'approccio delle librerie di classi. Puoi inizializzare il progetto C# Lambda in vari modi, ma il modo più semplice è utilizzare l'interfaccia CLI .NET con la CLI `Amazon.Lambda.Tools`. Configura la CLI `Amazon.Lambda.Tools` seguendo i passaggi indicati in [the section called “Ambiente di sviluppo”](#). Quindi, inizializza il progetto con il seguente comando.

```
dotnet new lambda.EmptyFunction --name ExampleCS
```

Questo comando genera la seguente struttura di file:

```
/project-root
# src
# ExampleCS
#   Function.cs (contains main handler)
#   Readme.md
#   aws-lambda-tools-defaults.json
#   ExampleCS.csproj
# test
# ExampleCS.Tests
#   FunctionTest.cs (contains main handler)
#   ExampleCS.Tests.csproj
```

In questa struttura di file, la logica del gestore principale per la funzione risiede nel file `Function.cs`.

## Esempio di codice della funzione Lambda C#

Il seguente esempio di codice della funzione Lambda C# raccoglie le informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3.

### Example Funzione Lambda **Function.cs**

```
using System;
using System.Text;
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using Amazon.S3.Model;

// Assembly attribute to enable Lambda function logging
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace ExampleLambda;

public class Order
{
    public string OrderId { get; set; } = string.Empty;
    public double Amount { get; set; }
    public string Item { get; set; } = string.Empty;
}

public class OrderHandler
{
    private static readonly AmazonS3Client s3Client = new();

    public async Task<string> HandleRequest(Order order, ILambdaContext context)
    {
        try
        {
            string? bucketName = Environment.GetEnvironmentVariable("RECEIPT_BUCKET");
            if (string.IsNullOrEmpty(bucketName))
            {
                throw new ArgumentException("RECEIPT_BUCKET environment variable is not set");
            }

            string receiptContent = $"OrderID: {order.OrderId}\nAmount:
            ${order.Amount:F2}\nItem: {order.Item}";
```

```

        string key = $"receipts/{order.OrderId}.txt";

        await UploadReceiptToS3(bucketName, key, receiptContent);

        context.Logger.LogInformation($"Successfully processed order
{order.OrderId} and stored receipt in S3 bucket {bucketName}");
        return "Success";
    }
    catch (Exception ex)
    {
        context.Logger.LogError($"Failed to process order: {ex.Message}");
        throw;
    }
}

private async Task UploadReceiptToS3(string bucketName, string key, string
receiptContent)
{
    try
    {
        var putRequest = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = key,
            ContentBody = receiptContent,
            ContentType = "text/plain"
        };

        await s3Client.PutObjectAsync(putRequest);
    }
    catch (AmazonS3Exception ex)
    {
        throw new Exception($"Failed to upload receipt to S3: {ex.Message}", ex);
    }
}
}

```

Questo file `Function.cs` contiene le sezioni seguenti:

- Istruzioni `using`: usate per importare le classi C# richieste dalla funzione Lambda.
- `[assembly: LambdaSerializer(...)]`: `LambdaSerializer` è un attributo assembly che indica a Lambda di convertire automaticamente i payload di eventi JSON in oggetti C# prima di passarli alla funzione.

- `namespace ExampleLambda`: definisce il namespace. In C#, il nome del namespace non deve corrispondere al nome del file.
- `public class Order {...}`: definisce la forma dell'evento di input previsto.
- `public class OrderHandler {...}`: definisce la tua classe C#. Al suo interno, definirai il metodo del gestore principale e qualsiasi altro metodo di supporto.
- `private static readonly AmazonS3Client s3Client = new();`: inizializza un client Amazon S3 con la catena di provider delle credenziali predefinita, al di fuori del metodo del gestore principale. Ciò fa sì che Lambda esegua questo codice durante la [fase di inizializzazione](#).
- `public async ... HandleRequest (Order order, ILambdaContext context)`: questo è il metodo dell'handler principale, che contiene la logica principale dell'applicazione.
- `private async Task UploadReceiptToS3(...)` {}: questo è un metodo helper a cui fa riferimento il metodo dell'handler principale `handleRequest`.

Poiché questa funzione richiede un client SDK Amazon S3, devi aggiungerlo alle dipendenze del progetto. Puoi farlo andando in `src/ExampleCS` e impartendo il comando seguente:

```
dotnet add package AWSSDK.S3
```

Aggiungi informazioni sui metadati a `aws-lambda-tools-defaults.json`

Per impostazione predefinita, il file `aws-lambda-tools-defaults.json` generato non contiene informazioni `profile` o `region` sulla tua funzione. Inoltre, aggiorna la stringa `function-handler` al valore corretto (`ExampleCS::ExampleLambda.OrderHandler::HandleRequest`). Puoi effettuare questo aggiornamento manualmente e aggiungere i metadati necessari per utilizzare un profilo di credenziali e una regione specifici per la tua funzione. Ad esempio, il file `aws-lambda-tools-defaults.json` dovrebbe essere simile all'esempio seguente:

```
{
  "Information": [
    "This file provides default values for the deployment wizard inside Visual Studio",
    "and the AWS Lambda commands added to the .NET Core CLI.",
    "To learn more about the Lambda commands with the .NET Core CLI execute the",
    "following command at the command line in the project root directory.",
    "dotnet lambda help",
    "All the command line options for the Lambda command can be specified in this",
    "file."
  ],
}
```



```

"profile": "default",
"region": "us-east-1",
"configuration": "Release",
"function-architecture": "x86_64",
"function-runtime": "dotnet8",
"function-memory-size": 512,
"function-timeout": 30,
"function-handler": "ExampleCS::ExampleLambda.OrderHandler::HandleRequest"
}

```

Affinché questa funzione funzioni correttamente, il suo [ruolo di esecuzione](#) deve consentire l'`s3:PutObject` azione. Inoltre, assicuratevi di definire la variabile di ambiente `RECEIPT_BUCKET`. Dopo una chiamata riuscita, il bucket Amazon S3 dovrebbe contenere un file di ricevuta.

### Gestori di librerie di classi

Il [codice di esempio](#) principale in questa pagina illustra un gestore di librerie di classi. I gestori di librerie di classi hanno la seguente struttura:

```

[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace NAMESPACE;

...

public class CLASSNAME {
    public async Task<string> METHODNAME (...) {
        ...
    }
}

```

Quando crei una funzione Lambda, devi fornire a Lambda informazioni sul gestore della funzione sotto forma di stringa nel [campo Gestore](#). Questo indica a Lambda quale metodo del codice eseguire quando la funzione viene richiamata. In C#, per i gestori di librerie di classi, il formato della stringa del gestore è `ASSEMBLY::TYPE::METHOD`, dove:

- `ASSEMBLY` è il nome del file di assembly .NET per l'applicazione. Se stai usando `Amazon.Lambda.Tools` CLI per creare l'applicazione e non specifichi il nome dell'assembly con la proprietà `AssemblyName` nel file `.csproj`, allora `ASSEMBLY` è semplicemente il nome del file `.csproj`.

- TYPE è il nome completo del tipo di gestore, cioè NAMESPACE . CLASSNAME.
- METHOD è il nome del metodo del gestore principale nel codice, cioè METHODNAME.

Per il codice di esempio principale, se l'assembly è denominato ExampleCS, la stringa completa del gestore è ExampleCS::ExampleLambda.OrderHandler::HandleRequest.

### Gestori di assembly eseguibili

È inoltre possibile definire le funzioni Lambda in C# come assembly eseguibile. I gestori di assembly eseguibili utilizzano la funzionalità di istruzioni di primo livello di C#, in cui il compilatore genera il Main() metodo e inserisce il codice della funzione al suo interno. Quando si utilizzano assembly eseguibili, è necessario avviare il runtime Lambda.

Per far ciò, utilizza il metodo LambdaBootstrapBuilder.Create nel codice. Gli input per questo metodo sono la funzione del gestore principale e il serializzatore Lambda da utilizzare. Di seguito viene illustrato un esempio di gestore di assembly eseguibile in C#:

```
namespace GetProductHandler;

IDatabaseRepository repo = new DatabaseRepository();

await LambdaBootstrapBuilder.Create<APIGatewayProxyRequest>(Handler, new
    DefaultLambdaJsonSerializer())
    .Build()
    .RunAsync();

async Task<APIGatewayProxyResponse> Handler(APIGatewayProxyRequest apigProxyEvent,
    ILambdaContext context)
{
    var id = apigProxyEvent.PathParameters["id"];
    var databaseRecord = await this.repo.GetById(id);

    return new APIGatewayProxyResponse
    {
        StatusCode = (int)HttpStatusCode.OK,
        Body = JsonSerializer.Serialize(databaseRecord)
    };
};
```

Nel [campo Gestore](#) per i gestori di assembly eseguibili, la stringa del gestore che indica a Lambda come eseguire il codice è il nome dell'assembly. In questo esempio è GetProductHandler.

## Firme del gestore valide per le funzioni C#

In C#, le firme del gestore Lambda valide richiedono da 0 a 2 argomenti. In genere, la firma del gestore ha due argomenti, come mostrato nell'esempio principale:

```
public async Task<string> HandleRequest(Order order, ILambdaContext context)
```

Quando si forniscono due argomenti, il primo argomento deve essere l'input dell'evento e il secondo argomento deve essere l'oggetto di contesto Lambda. Entrambi gli argomenti sono facoltativi. Ad esempio, le seguenti sono anche firme valide di gestori Lambda in C#:

- `public async Task<string> HandleRequest()`
- `public async Task<string> HandleRequest(Order order)`
- `public async Task<string> HandleRequest(ILambdaContext context)`

Oltre alla sintassi di base della firma del gestore, esistono alcune restrizioni aggiuntive:

- Non è possibile utilizzare la parola chiave `unsafe` nella firma del gestore. Tuttavia, puoi utilizzare il contesto `unsafe` all'interno del metodo del gestore e delle sue dipendenze. Per ulteriori informazioni, consulta [unsafe \(Riferimenti per C#\)](#) sul sito Web di Microsoft.
- Il gestore non può utilizzare la `params` parola chiave o utilizzarla `ArgIterator` come parametro di input o di ritorno. Queste parole chiave supportano un numero variabile di parametri. Il gestore può accettare al massimo due argomenti.
- Il gestore non può essere un metodo generico. In altre parole, non può utilizzare parametri di tipo generico come `<T>`.
- Lambda non supporta gestori asincroni con `async void` nella firma.

## Convenzioni di denominazione dei gestori

I gestori Lambda in C# non hanno restrizioni di denominazione rigide. Tuttavia, devi assicurarti di fornire la stringa del gestore corretta a Lambda quando implementi la funzione. La stringa del gestore corretta dipende da se stai implementando un [gestore di libreria di classi](#) o un [gestore di assembly eseguibile](#).

Sebbene sia possibile utilizzare qualsiasi nome per il gestore, i nomi delle funzioni in C# sono generalmente in PascalCase. Inoltre, sebbene non sia necessario che il nome del file corrisponda

al nome della classe o del gestore, in genere è consigliabile utilizzare un nome di file come `OrderHandler.cs` se il nome della classe è `OrderHandler`. Ad esempio, è possibile modificare il nome del file in questo esempio da `Function.cs` a `OrderHandler.cs`.

## Serializzazione nelle funzioni Lambda C#

JSON è il formato di input più comune e standard per le funzioni Lambda. In questo esempio, la funzione prevede un input simile a quanto segue:

```
{
  "orderId": "12345",
  "amount": 199.99,
  "item": "Wireless Headphones"
}
```

In C#, puoi definire la forma dell'evento di input previsto in una classe. In questo esempio, definiamo la classe `Order` per modellare questo input:

```
public class Order
{
    public string OrderId { get; set; } = string.Empty;
    public double Amount { get; set; }
    public string Item { get; set; } = string.Empty;
}
```

Se la funzione Lambda utilizza i tipi di input/output diversi da un oggetto `Stream`, è necessario aggiungere una libreria di serializzazione all'applicazione. È quindi possibile convertire l'input JSON in un'istanza della classe definita. Esistono due metodi di serializzazione per le funzioni C# in Lambda: serializzazione basata sulla riflessione e serializzazione generata dal codice sorgente.

## Reflection-based serializzazione

AWS fornisce librerie predefinite che è possibile aggiungere rapidamente all'applicazione. Queste librerie implementano la serializzazione utilizzando la [riflessione](#). Utilizza uno dei seguenti pacchetti per implementare la serializzazione basata sulla riflessione:

- `Amazon.Lambda.Serialization.SystemTextJson`: nel backend, questo pacchetto utilizza `System.Text.Json` per eseguire attività di serializzazione.
- `Amazon.Lambda.Serialization.Json`: nel backend, questo pacchetto utilizza `Newtonsoft.Json` per eseguire attività di serializzazione.

È possibile inoltre creare una libreria di serializzazione personalizzata implementando l'interfaccia `ILambdaSerializer`, disponibile come parte della libreria `Amazon.Lambda.Core`. L'interfaccia definisce due metodi:

- `T Deserialize<T>(Stream requestStream);`

Questo metodo viene implementato per deserializzare il payload della richiesta dall'API `Invoke` nell'oggetto passato al gestore della funzione `Lambda`.

- `T Serialize<T>(T response, Stream responseStream);`

Questo metodo viene implementato per serializzare il risultato restituito dal gestore della funzione `Lambda` nel payload della risposta restituito dall'operazione dell'API `Invoke`.

L'esempio principale di questa pagina utilizza la serializzazione basata sulla riflessione.

Reflection-based la serializzazione funziona immediatamente AWS Lambda e non richiede alcuna configurazione aggiuntiva, il che la rende una buona scelta per la semplicità. Tuttavia, richiede un maggiore utilizzo della memoria funzionale. È inoltre possibile che si verifichino latenze di funzione più elevate dovute alla riflessione in fase di esecuzione.

### Source-generated serializzazione

Con la serializzazione generata dal codice sorgente, il codice di serializzazione viene generato in fase di compilazione. Ciò elimina la necessità di riflessione e può migliorare le prestazioni della tua funzione. Per utilizzare la serializzazione generata dal codice sorgente nella tua funzione, devi procedere come indicato di seguito:

- Crea una nuova classe parziale che eredita da `JsonSerializerContext`, aggiungendo attributi `JsonSerializable` per tutti i tipi che richiedono la serializzazione o la deserializzazione.
- Configura il `LambdaSerializer` per utilizzare un `SourceGeneratorLambdaJsonSerializer<T>`.
- Aggiorna qualsiasi serializzazione e deserializzazione manuale nel codice dell'applicazione per utilizzare la classe appena creata.

L'esempio seguente mostra come puoi modificare l'esempio principale di questa pagina, che utilizza la serializzazione basata sulla riflessione, in modo da utilizzare invece la serializzazione generata dal codice sorgente.

```
using System.Text.Json;
```

```
using System.Text.Json.Serialization;

...

public class Order
{
    public string OrderId { get; set; } = string.Empty;
    public double Amount { get; set; }
    public string Item { get; set; } = string.Empty;
}

[JsonSerializable(typeof(Order))]
public partial class OrderJsonContext : JsonSerializerContext {}

public class OrderHandler
{
    ...

    public async Task<string> HandleRequest(string input, ILambdaContext context)
    {
        var order = JsonSerializer.Deserialize(input, OrderJsonContext.Default.Order);

        ...
    }
}
```

Source-generated la serializzazione richiede una configurazione maggiore rispetto alla serializzazione basata sulla riflessione. Tuttavia, le funzioni che utilizzano la generazione da codice sorgente tendono a utilizzare meno memoria e offrono prestazioni migliori grazie alla generazione di codice in fase di compilazione. Per contribuire a eliminare gli [avvii a freddo](#) delle funzioni, prendi in considerazione la possibilità di passare alla serializzazione generata dal codice sorgente.

#### Note

Se desideri utilizzare la [compilazione anticipata \(AOT\)](#) nativa con Lambda, devi utilizzare la serializzazione generata dal codice sorgente.

## File-based funzioni

Introdotte in .NET 10, le app basate su file consentono di creare applicazioni.NET da un singolo .cs file, senza una struttura di .csproj file o directory. Lambda supporta funzioni basate su file, a partire da .NET 10. Offrono un modo semplice e leggero per creare funzioni Lambda in C#.

Il modo più veloce per iniziare a creare una funzione Lambda basata su file C# è utilizzare il pacchetto `Amazon.Lambda.Templates`. Per installare il pacchetto, esegui il comando seguente:

```
dotnet new install Amazon.Lambda.Templates
```

Quindi, crea una funzione di esempio Lambda basata su file C#:

```
dotnet new lambda.FileBased -n MyLambdaFunction
```

File-based le funzioni utilizzano gestori di assembly eseguibili. È quindi necessario includere il `Amazon.Lambda.RuntimeSupport` NuGet pacchetto e utilizzare il `LambdaBootstrapBuilder.Create` metodo per registrare la funzione del gestore .NET per il tipo di evento e avviare il client di runtime .NET Lambda.

File-based le funzioni utilizzano .NET Native AOT per impostazione predefinita, che richiede la serializzazione generata dal codice sorgente. È possibile disattivare Native AOT specificandolo nel file sorgente. `#:property PublishAot=false` Per ulteriori informazioni sull'utilizzo di Native AOT in Lambda, consulta [the section called “Compilazione AOT nativa”](#)

### Accesso e utilizzo dell'oggetto contestuale Lambda

L'[oggetto contesto](#): contiene informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. In questo esempio, l'oggetto contesto è di tipo `Amazon.Lambda.Core.ILambdaContext` ed è il secondo argomento della funzione del gestore principale.

```
public async Task<string> HandleRequest(Order order, ILambdaContext context) {  
    ...  
}
```

L'oggetto contesto è un input opzionale. Per ulteriori informazioni sulle firme valide accettate del gestore, consulta [the section called “Firme del gestore valide per le funzioni C#”](#).

L'oggetto `context` è utile per produrre log delle funzioni su Amazon CloudWatch. Puoi usare il metodo `context.getLogger()` per ottenere un oggetto `LambdaLogger` per la registrazione. In questo esempio, possiamo usare il logger per registrare il log di un messaggio di errore se l'elaborazione fallisce per qualche motivo:

```
context.Logger.LogError($"Failed to process order: {ex.Message}");
```

Oltre alla registrazione di log, puoi anche utilizzare l'oggetto contesto per il monitoraggio delle funzioni. Per ulteriori informazioni sulla copia di oggetti, consulta la sezione [the section called "Contesto"](#).

### Utilizzo di SDK per .NET v3 nel tuo gestore

Spesso, utilizzerai le funzioni Lambda per interagire o apportare aggiornamenti ad altre risorse. AWS Il modo più semplice per interfacciarsi con queste risorse è utilizzare la SDK per .NET v3.

#### Note

La SDK per .NET (v2) è obsoleta. Si consiglia di utilizzare solo la versione v3. SDK per .NET

Puoi aggiungere dipendenze SDK al tuo progetto usando il comando `Amazon.Lambda.Tools` seguente:

```
dotnet add package <package_name>
```

Ad esempio, nell'esempio principale di questa pagina, è necessario utilizzare l'API Amazon S3 per caricare una ricevuta su S3. Possiamo importare il client SDK Amazon S3 con il seguente comando:

```
dotnet add package AWSSDK.S3
```

Questo comando aggiunge la dipendenza al tuo progetto. Dovresti anche vedere una riga simile alla seguente nel file `.csproj` del tuo progetto:

```
<PackageReference Include="AWSSDK.S3" Version="3.7.2.18" />
```

Quindi, importa le dipendenze direttamente nel tuo codice C#:

```
using Amazon.S3;
```



```
using Amazon.S3.Model;
```

Il codice di esempio inizializza quindi un client Amazon S3 (utilizzando la [catena di provider delle credenziali predefinita](#)) come indicato di seguito:

```
private static readonly AmazonS3Client s3Client = new();
```

In questo esempio, abbiamo inizializzato il nostro client Amazon S3 all'esterno della funzione del gestore principale per evitare di doverlo inizializzare ogni volta che invochiamo la nostra funzione. Dopo aver inizializzato il client SDK, puoi utilizzarlo per interagire con altri servizi. AWS Il codice di esempio richiama l'API PutObject Amazon S3 nel modo seguente:

```
var putRequest = new PutObjectRequest
{
    BucketName = bucketName,
    Key = key,
    ContentBody = receiptContent,
    ContentType = "text/plain"
};

await s3Client.PutObjectAsync(putRequest);
```

## Accesso alle variabili d'ambiente

Nel codice dell'handler, puoi fare riferimento a qualsiasi [variabile di ambiente](#) utilizzando il metodo `System.Environment.GetEnvironmentVariable`. In questo esempio, facciamo riferimento alla variabile di ambiente `RECEIPT_BUCKET` definita utilizzando le seguenti righe di codice:

```
string? bucketName = Environment.GetEnvironmentVariable("RECEIPT_BUCKET");
if (string.IsNullOrEmpty(bucketName))
{
    throw new ArgumentException("RECEIPT_BUCKET environment variable is not set");
}
```

## Utilizzo dello stato globale

Lambda esegue il codice statico e il costruttore della classe durante la [fase di inizializzazione](#) prima di richiamare la funzione per la prima volta. Le risorse create durante l'inizializzazione restano in memoria tra le invocazioni, in modo da evitare di doverle creare ogni volta che si invoca la funzione.

Nel codice di esempio, il codice di inizializzazione del client S3 non rientra nel metodo del gestore principale. Il runtime inizializza il client prima che la funzione gestisca il suo primo evento, il che può portare a tempi di elaborazione più lunghi. Gli eventi successivi sono molto più veloci perché Lambda non ha bisogno di inizializzare nuovamente il client.

## Semplificazione del codice delle funzioni con il framework Lambda Annotations

[Lambda Annotations](#) è un framework per .NET 8 che semplifica la scrittura di funzioni Lambda utilizzando C#. Il framework Annotations utilizza [generatori di codice sorgente](#) per generare codice che si traduce dal modello di programmazione Lambda al codice semplificato. Con il framework Annotations, è possibile sostituire gran parte del codice in una funzione Lambda scritta utilizzando il normale modello di programmazione. Il codice scritto utilizzando il framework utilizza espressioni più semplici che consentono di concentrarsi sulla logica aziendale. Per alcuni esempi [Amazon.Lambda.Annotations](#), consulta la documentazione di nuget.

Per un esempio di applicazione completa che utilizza Lambda Annotations, guarda l'[PhotoAssetManager](#) esempio nel repository. `awsdocs/aws-doc-sdk-examples` GitHub Il file `Function.cs` principale nella directory `PamApiAnnotations` usa Lambda Annotations. Per fare un confronto, la directory `PamApi` contiene file equivalenti scritti utilizzando il normale modello di programmazione Lambda.

## Iniezione delle dipendenze con il framework Lambda Annotations

Puoi utilizzare il framework Lambda Annotations anche per aggiungere l'iniezione di dipendenze alle tue funzioni Lambda utilizzando una sintassi che conosci. Quando aggiungi un `[LambdaStartup]` attributo a un `Startup.cs` file, il framework Lambda Annotations genera il codice richiesto in fase di compilazione.

```
[LambdaStartup]
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<IDatabaseRepository, DatabaseRepository>();
    }
}
```

La tua funzione Lambda può iniettare servizi utilizzando l'iniezione del costruttore o iniettandoli in metodi individuali utilizzando l'attributo `[FromServices]`.

```
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))

namespace GetProductHandler;

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function(IDatabaseRepository repo)
    {
        this._repo = repo;
    }

    [LambdaFunction]
    [HttpApi(LambdaHttpMethod.Get, "/product/{id}")]
    public async Task<Product> FunctionHandler([FromServices] IDatabaseRepository
repository, string id)
    {
        return await this._repo.GetById(id);
    }
}
```

## Best practice di codice per le funzioni Lambda con C#

Segui le linee guida riportate nell'elenco seguente per utilizzare le best practice di codifica durante la creazione delle funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. Con questo approccio, è possibile creare una funzione più testabile a livello di unità.
- Controllare le dipendenze nel pacchetto di distribuzione della funzione. L'ambiente di esecuzione AWS Lambda contiene diverse librerie. Per abilitare l'ultimo set di funzionalità e aggiornamenti di sicurezza, Lambda aggiorna periodicamente queste librerie. Questi aggiornamenti potrebbero introdurre lievi modifiche al comportamento della funzione Lambda. Per mantenere il controllo completo delle dipendenze utilizzate dalla funzione, inserire tutte le dipendenze nel pacchetto di implementazione.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del [contesto di esecuzione](#).

- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. Ciò riduce il tempo necessario per scaricare e decomprimere il pacchetto di distribuzione prima dell'invocazione. Per le funzioni create in .NET, evita di caricare l'intera libreria AWS SDK come parte del pacchetto di distribuzione. Dipendi invece in modo selettivo dai moduli che raccolgono i componenti dell'SDK di cui hai bisogno (ad esempio, DynamoDB, moduli SDK Amazon S3 e librerie di base Lambda).

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva `keep-alive` per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva `keep-alive` associata al runtime. Per un esempio, consulta [Reusing Connections with in. `Keep-Alive Node.js`](#)

Utilizzare [le variabili di ambiente](#) per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su `0` per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionali alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha

una dipendenza su queste API non pubbliche. Consulta il [riferimento all'API](#) per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta [Come posso rendere idempotente la mia funzione Lambda?](#).

## Crea e implementa le funzioni Lambda C# con gli archivi di file .zip

Un pacchetto di implementazione .NET (archivio di file .zip) contiene l'assembly compilato della funzione insieme a tutte le dipendenze dell'assembly stesso. Il pacchetto contiene inoltre un file *proj.deps.json*, Ciò indica al runtime di .NET tutte le dipendenze della funzione e un file *proj.runtimeconfig.json* utilizzato per configurare il runtime.

Per implementare singole funzioni Lambda, puoi utilizzare l'interfaccia a riga di comando globale Lambda .NET Amazon.Lambda.Tools. L'utilizzo del comando `dotnet lambda deploy-function` crea automaticamente un pacchetto di implementazione .zip e lo distribuisce su Lambda. Tuttavia, ti consigliamo di utilizzare framework come AWS Serverless Application Model (AWS SAM) o the Kit di sviluppo per il cloud AWS (AWS CDK) per distribuire le tue applicazioni.NET. AWS

Le applicazioni serverless di solito comprendono una combinazione di funzioni Lambda e altre funzioni gestite che Servizi AWS collaborano per eseguire una particolare attività aziendale. AWS SAM e AWS CDK semplificano la creazione e l'implementazione di funzioni Lambda insieme ad altre su larga scala. Servizi AWS Le specifiche del [AWS SAM modello](#) forniscono una sintassi semplice e chiara per descrivere le funzioni, le API, le autorizzazioni, le configurazioni e altre AWS risorse Lambda che costituiscono l'applicazione serverless. Con il [AWS CDK](#) si definisce l'infrastruttura cloud come codice per creare applicazioni affidabili, scalabili e convenienti nel cloud utilizzando linguaggi di programmazione e framework di programmazione moderni come .NET. Sia il che AWS SAM utilizzano la AWS CDK CLI .NET Lambda Global per impacchettare le funzioni.

Anche se è possibile utilizzare [livelli Lambda](#) con funzioni in C# [tramite la CLI di .NET Core](#), tale scelta non è preferibile. Le funzioni in C# che utilizzano livelli caricano manualmente gli assembly condivisi in memoria durante il [Fase di init](#), per cui i tempi di avvio a freddo possono aumentare. Includi invece tutto il codice condiviso in fase di compilazione per evitare che il caricamento degli assembly al runtime comprometta le prestazioni.

Nelle sezioni seguenti sono disponibili le istruzioni per la creazione e la distribuzione di funzioni .NET Lambda Global utilizzando AWS SAM AWS CDK, the e .NET Lambda Global CLI.

### Argomenti

- [Utilizzo della CLI globale Lambda di .NET](#)
- [Distribuisce le funzioni C# Lambda usando AWS SAM](#)
- [Distribuisce funzioni C# Lambda utilizzando AWS CDK](#)
- [Distribuisce ASP.NET applicazioni](#)

## Utilizzo della CLI globale Lambda di .NET

L'estensione .NET Lambda Global Tools () e l'estensione .NET Lambda Global Tools (Amazon.Lambda.Tools) offrono un metodo di creazione multiplatforma .NET-based Applicazioni Lambda, impacchettatele e distribuite su Lambda. In questa sezione, scoprirai come creare nuovi progetti Lambda .NET utilizzando la CLI .NET e i modelli Amazon Lambda e come impacchettarli e implementarli tramite Amazon.Lambda.Tools

### Argomenti

- [Prerequisiti](#)
- [Creazione di progetti .NET con la CLI .NET](#)
- [Implementazione di progetti .NET con la CLI .NET](#)
- [Utilizzo dei livelli Lambda con la CLI di .NET](#)

### Prerequisiti

#### SDK .NET 8

Se non l'hai già fatto, installa l'SDK [.NET 8](#) e Runtime.

#### AWS Amazon.Lambda.Templates Modelli di progetto.NET

Per generare il codice della funzione Lambda, usa il [Amazon.Lambda.Templates](#) NuGet pacchetto. Per installare questo pacchetto del modello, esegui il comando riportato:

```
dotnet new install Amazon.Lambda.Templates
```

#### AWS Amazon.Lambda.Tools Strumenti CLI.NET Global

Per creare le funzioni Lambda, usa l'[estensione degli strumenti globali .NET Amazon.Lambda.Tools](#). Per installare Amazon.Lambda.Tools, esegui il comando riportato:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Per ulteriori informazioni sull'Amazon.Lambda.Tools estensione.NET CLI, vedere il repository [AWS Extensions for .NET CLI su](#). GitHub

## Creazione di progetti .NET con la CLI .NET

Nella CLI .NET, utilizzi il comando `dotnet new` per creare progetti .NET da una riga di comando. Lambda offre modelli aggiuntivi utilizzando il pacchetto. [Amazon.Lambda.Templates](#) NuGet

Dopo aver installato questo pacchetto, esegui il comando di seguito per visualizzare un elenco di modelli disponibili.

```
dotnet new list
```

Per esaminare i dettagli relativi a un modello, utilizzare l'opzione `help`. Ad esempio, per visualizzare i dettagli del modello `lambda.EmptyFunction`, emetti il seguente comando.

```
dotnet new lambda.EmptyFunction --help
```

Per creare un modello di base per una funzione Lambda .NET, usa il modello `lambda.EmptyFunction`. In questo modo viene creata una funzione semplice che accetta una stringa come input e la converte in lettere maiuscole utilizzando il metodo `ToUpper`. Questo modello supporta le seguenti opzioni:

- `--name`: il nome della funzione.
- `--region`— La AWS regione in cui creare la funzione.
- `--profile`— Il nome di un profilo nel file delle AWS SDK per .NET credenziali. Per saperne di più sui profili di credenziali in .NET, vedi [Configurare AWS le credenziali](#) nella AWS SDK for .NET Developer Guide.

In questo esempio, creiamo una nuova funzione vuota denominata `myDotnetFunction` utilizzando il profilo e le impostazioni predefiniti: Regione AWS

```
dotnet new lambda.EmptyFunction --name myDotnetFunction
```

Questo comando crea i seguenti file e directory nella directory di progetto.

```
### myDotnetFunction
### src
#   ### myDotnetFunction
#       ### Function.cs
#       ### Readme.md
```



```
#      ### aws-lambda-tools-defaults.json
#      ### myDotnetFunction.csproj
### test
    ### myDotnetFunction.Tests
        ### FunctionTest.cs
        ### myDotnetFunction.Tests.csproj
```

Nella directory `src/myDotnetFunction` esaminare i file seguenti:

- `aws-lambda-tools-defaults.json`: file in cui si specificano le opzioni della riga di comando quando si distribuisce la funzione Lambda. Ad esempio:

```
"profile" : "default",
"region" : "us-east-2",
"configuration" : "Release",
"function-architecture": "x86_64",
"function-runtime": "dotnet8",
"function-memory-size" : 256,
"function-timeout" : 30,
"function-handler" : "myDotnetFunction::myDotnetFunction.Function::FunctionHandler"
```

- `Function.cs`: il codice della funzione del gestore Lambda. Si tratta di un modello C# che include la libreria `Amazon.Lambda.Core` e un attributo `LambdaSerializer` predefiniti. Per ulteriori informazioni sui requisiti e sulle opzioni di serializzazione, consulta [Serializzazione nelle funzioni Lambda C#](#). Include anche una funzione di esempio che è possibile modificare per applicare il codice della funzione Lambda.

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted into
// a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace myDotnetFunction;

public class Function
{
    /// <summary>
    /// A simple function that takes a string and does a ToUpper
    /// </summary>
```

```

    /// <param name="input"></param>
    /// <param name="context"></param>
    /// <returns></returns>
    public string FunctionHandler(string input, ILambdaContext context)
    {
        return input.ToUpper();
    }
}

```

- my DotnetFunction.csproj: un file [MSBuild che elenca i](#) file e gli assembly che compongono l'applicazione.

```

<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AWSProjectType>Lambda</AWSProjectType>
    <!-- This property makes the build directory similar to a publish directory and
    helps the AWS .NET Lambda Mock Test Tool find project dependencies. -->
    <CopyLocalLockFileAssemblies>true</CopyLocalLockFileAssemblies>
    <!-- Generate ready to run images during publishing to improve cold start time.
    -->
    <PublishReadyToRun>true</PublishReadyToRun>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="2.2.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.SystemTextJson"
    Version="2.4.0" />
  </ItemGroup>
</Project>

```

- Readme: file che consente di documentare la funzione Lambda.

Nella directory myfunction/test esaminare i file seguenti:

- my DotnetFunction.Tests.csproj: Come notato in precedenza, questo è un file [MSBuild che elenca i](#) file e gli assembly che compongono il progetto di test. Si noti anche che nel file è inclusa la libreria Amazon.Lambda.Core, che consente di integrare in modo semplice qualsiasi modello Lambda necessario per eseguire il test della funzione.

```
<Project Sdk="Microsoft.NET.Sdk">
  ...

  <PackageReference Include="Amazon.Lambda.Core" Version="2.2.0 " />
  ...
```

- **FunctionTest.cs:** Lo stesso file modello di codice C# incluso nella directory. `src` Modificare questo file per eseguire il mirroring del codice di produzione della funzione e per eseguirne il test prima di caricare la funzione Lambda in un ambiente di produzione.

```
using Xunit;
using Amazon.Lambda.Core;
using Amazon.Lambda.TestUtilities;

using MyFunction;

namespace MyFunction.Tests
{
    public class FunctionTest
    {
        [Fact]
        public void TestToUpperFunction()
        {
            // Invoke the lambda function and confirm the string was upper cased.
            var function = new Function();
            var context = new TestLambdaContext();
            var upperCase = function.FunctionHandler("hello world", context);

            Assert.Equal("HELLO WORLD", upperCase);
        }
    }
}
```

## Implementazione di progetti .NET con la CLI .NET

Per creare il pacchetto di implementazione e implementarlo in Lambda, utilizzi gli strumenti della CLI `Amazon.Lambda.Tools`. Per implementare la funzione dai file creati nei passaggi precedenti, per prima cosa accedi alla cartella contenente il file `.csproj` della funzione.

```
cd myDotnetFunction/src/myDotnetFunction
```

Per implementare il codice in Lambda come pacchetto di implementazione .zip, emetti il comando seguente. Scegli il nome della tua funzione.

```
dotnet lambda deploy-function myDotnetFunction
```

Durante l'implementazione, la procedura guidata chiede di selezionare un [the section called “Ruolo di esecuzione \(autorizzazioni per le funzioni per accedere ad altre risorse\)”](#). Per questo esempio, seleziona `lambda_basic_role`.

Dopo aver implementato la funzione, puoi testarla nel cloud con il comando `dotnet lambda invoke-function`. Per il codice di esempio nel modello `lambda.EmptyFunction`, puoi testare la tua funzione inserendo una stringa utilizzando l'opzione `--payload`.

```
dotnet lambda invoke-function myDotnetFunction --payload "Just checking if everything is OK"
```

Se la funzione è stata implementata con successo, dovresti vedere un output simile al seguente.

```
dotnet lambda invoke-function myDotnetFunction --payload "Just checking if everything is OK"
Amazon Lambda Tools for .NET Core applications (5.8.0)
Project Home: https://github.com/aws/aws-extensions-for-dotnet-cli, https://github.com/aws/aws-lambda-dotnet

Payload:
"JUST CHECKING IF EVERYTHING IS OK"

Log Tail:
START RequestId: id Version: $LATEST
END RequestId: id
REPORT RequestId: id Duration: 0.99 ms      Billed Duration: 1 ms      Memory
Size: 256 MB      Max Memory Used: 12 MB
```

## Utilizzo dei livelli Lambda con la CLI di .NET

### Note

Anche se è possibile utilizzare [livelli](#) con funzioni in .NET, tale scelta non è preferibile. Le funzioni in .NET che utilizzano livelli caricano manualmente gli assembly condivisi in memoria durante la fase `Init`, per cui i tempi di avvio a freddo possono aumentare. Includi, invece, tutto il codice condiviso in fase di compilazione per sfruttare le ottimizzazioni integrate del compilatore .NET.

La CLI di .NET supporta comandi che facilitano la pubblicazione di livelli e l'implementazione di funzioni C# che utilizzano livelli. Per pubblicare un livello in un bucket Amazon S3 specificato, utilizza il comando seguente nella stessa directory del tuo file `.csproj`:

```
dotnet lambda publish-layer <layer_name> --layer-type runtime-package-store --s3-bucket <s3_bucket_name>
```

Quando implementi la funzione utilizzando la CLI di .NET, quindi, specifica l'ARN utilizzato dal livello nel seguente comando:

```
dotnet lambda deploy-function <function_name> --function-layers arn:aws:lambda:us-east-1:123456789012:layer:layer-name:1
```

Per un esempio completo di una funzione Hello World, consulta l'esempio [blank-csharp-with-layer](#).

## Distribuisce le funzioni C# Lambda usando AWS SAM

AWS Serverless Application Model (AWS SAM) è un toolkit che aiuta a semplificare il processo di creazione ed esecuzione di applicazioni serverless su AWS. Definisci le risorse per la tua applicazione in un modello YAML o JSON e usi l'interfaccia a riga di AWS SAM comando (AWS SAM CLI) per creare, impacchettare e distribuire le tue applicazioni. Quando crei una funzione Lambda da un AWS SAM modello, crea AWS SAM automaticamente un pacchetto di distribuzione `.zip` o un'immagine del contenitore con il codice della funzione e le dipendenze specificate. AWS SAM quindi distribuisce la funzione utilizzando uno stack. [CloudFormation](#) Per saperne di più sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda, consulta [Getting started with AWS SAM](#) nella Developer Guide. AWS Serverless Application Model

La seguente procedura mostra come scaricare, creare e implementare un'applicazione Hello World .NET di esempio con AWS SAM. Questa applicazione di esempio utilizza una funzione Lambda e un endpoint Gateway Amazon API per implementare un backend API di base. Quando invii una richiesta HTTP GET all'endpoint Gateway API, Gateway API richiama la funzione Lambda. La funzione restituisce un messaggio "hello world", insieme all'indirizzo IP dell'istanza della funzione Lambda che elabora la richiesta.

Quando si crea e si distribuisce l'applicazione utilizzando AWS SAM, dietro le quinte, la AWS SAM CLI utilizza il `dotnet lambda package` comando per impacchettare i singoli pacchetti di codici funzione Lambda.

## Prerequisiti

### SDK .NET 8

Installa l'SDK [.NET 8](#) e Runtime.

AWS SAM CLI versione 1.39 o successiva

Per informazioni su come installare la versione più recente della AWS SAM CLI, vedi [Installazione della CLI. AWS SAM](#)

## Distribuisce un esempio AWS SAM applicazione

1. Inizializza l'applicazione utilizzando il modello .NET Hello world con il comando riportato di seguito.

```
sam init --app-template hello-world --name sam-app \
--package-type Zip --runtime dotnet8
```

Questo comando crea i seguenti file e directory nella directory di progetto.

```
### sam-app
### README.md
### events
#   ### event.json
### omnisharp.json
### samconfig.toml
### src
#   ### HelloWorld
#       ### Function.cs
```

```
#      ### HelloWorld.csproj
#      ### aws-lambda-tools-defaults.json
### template.yaml
### test
    ### HelloWorld.Test
        ### FunctionTest.cs
        ### HelloWorld.Tests.csproj
```

2. Passa alla directory contenente il `template.yaml` file. Questo file è un modello che definisce le risorse AWS per l'applicazione, tra cui la funzione Lambda e un'API di Gateway API.

```
cd sam-app
```

3. Per creare il codice sorgente dell'applicazione, digita il comando riportato di seguito.

```
sam build
```

4. Per distribuire l'applicazione su AWS, esegui il comando seguente.

```
sam deploy --guided
```

Questo comando impacchetta e implementa l'applicazione con la seguente serie di prompt. Per accettare le opzioni predefinite, premi Invio.

#### Note

Perché HelloWorldFunction potrebbe non avere l'autorizzazione definita, va bene? , assicurati di entrare y.

- Nome dello stack: il nome dello stack da implementare su CloudFormation. Questo nome deve essere univoco per il tuo Account AWS e Regione AWS.
- Regione AWS: Il Regione AWS luogo in cui desideri distribuire la tua app.
- Conferma le modifiche prima dell'implementazione: seleziona Sì per rivedere manualmente tutti i set di modifiche prima che AWS SAM implementi modifiche all'applicazione. Se si seleziona no, la AWS SAM CLI distribuisce automaticamente le modifiche all'applicazione.
- Consenti la creazione di ruoli IAM CLI SAM: molti AWS SAM modelli, incluso quello di Hello world in questo esempio, creano ruoli AWS Identity and Access Management (IAM) per concedere alle tue funzioni Lambda il permesso di accedere ad altre funzioni. Servizi AWS

Seleziona Sì per fornire l'autorizzazione a distribuire uno CloudFormation stack che crea o modifica i ruoli IAM.

- Disabilita il rollback: per impostazione predefinita, se AWS SAM si verifica un errore durante la creazione o la distribuzione dello stack, ripristina lo stack alla versione precedente. Seleziona No per accettare questa impostazione predefinita.
  - HelloWorldFunction potrebbe non avere l'autorizzazione definita, va bene: Invio. y
  - Salva gli argomenti in samconfig.toml: seleziona Sì per salvare le tue scelte di configurazione. In futuro, potrai eseguire nuovamente `sam deploy` senza parametri per implementare le modifiche all'applicazione.
5. Una volta completata l'implementazione dell'applicazione, la CLI restituisce il nome della risorsa Amazon (ARN) della funzione Lambda Hello World e il ruolo IAM creato per essa. Consente inoltre di visualizzare anche l'endpoint dell'API di Gateway API. Per testare l'applicazione, apri l'endpoint in un browser. Si avrà una risposta simile alla seguente.

```
{"message":"hello world","location":"34.244.135.203"}
```

6. Per eliminare le risorse, emetti il seguente comando. Tieni presente che l'endpoint dell'API che hai creato è un endpoint pubblico accessibile su Internet. È consigliabile eliminare questo endpoint dopo il test.

```
sam delete
```

## Fasi successive

Per saperne di più sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda usando .NET, consulta le seguenti risorse:

- La [AWS Serverless Application Model Guida per gli sviluppatori di \(AWS SAM\)](#)
- [Creazione di applicazioni.NET serverless con AWS Lambda e SAM CLI](#)

## Distribuisce funzioni C# Lambda utilizzando AWS CDK

Kit di sviluppo per il cloud AWS (AWS CDK) È un framework di sviluppo software open source per definire l'infrastruttura cloud come codice con linguaggi di programmazione e framework moderni come .NET. AWS CDK i progetti vengono eseguiti per generare CloudFormation modelli che vengono poi utilizzati per distribuire il codice.



Per creare e distribuire un'applicazione Hello world .NET di esempio utilizzando il AWS CDK, segui le istruzioni nelle sezioni seguenti. L'applicazione di esempio implementa un backend di API di base che consiste di un endpoint Gateway API e di una funzione Lambda. Quando si invia una richiesta HTTP GET all'endpoint, Gateway API richiama la funzione Lambda. La funzione restituisce un messaggio "hello world", insieme all'indirizzo IP dell'istanza Lambda che elabora la richiesta.

## Prerequisiti

### SDK .NET 8

Installa l'SDK [.NET 8](#) e Runtime.

### AWS CDK versione 2

Per informazioni su come installare la versione più recente di [Getting started with AWS CDK](#) in the Kit di sviluppo per il cloud AWS (AWS CDK) v2 Developer Guide. AWS CDK

## Distribuisci un esempio AWS CDK applicazione

1. Crea una directory di progetto per l'applicazione di esempio e passa ad essa.

```
mkdir hello-world  
cd hello-world
```

2. Inizializza una nuova AWS CDK applicazione eseguendo il comando seguente.

```
cdk init app --language csharp
```

Questo comando crea i seguenti file e directory nella directory di progetto.

```
### README.md  
### cdk.json  
### src  
  ### HelloWorld  
  #   ### GlobalSuppressions.cs  
  #   ### HelloWorld.csproj  
  #   ### HelloWorldStack.cs  
  #   ### Program.cs  
  ### HelloWorld.sln
```

3. Apri la directory `src` e crea una nuova funzione Lambda utilizzando la CLI .NET. Questa è la funzione che si implementa utilizzando AWS CDK. In questo esempio, viene creata una funzione Hello world denominata `HelloWorldLambda` utilizzando il modello `lambda.EmptyFunction`.

```
cd src
dotnet new lambda.EmptyFunction -n HelloWorldLambda
```

Dopo questo passaggio, la struttura di directory all'interno della directory del progetto dovrebbe avere un aspetto simile al seguente.

```
### README.md
### cdk.json
### src
  ### HelloWorld
  #   ### GlobalSuppressions.cs
  #   ### HelloWorld.csproj
  #   ### HelloWorldStack.cs
  #   ### Program.cs
  ### HelloWorld.sln
  ### HelloWorldLambda
    ### src
    #   ### HelloWorldLambda
    #   ### Function.cs
    #   ### HelloWorldLambda.csproj
    #   ### Readme.md
    #   ### aws-lambda-tools-defaults.json
  ### test
    ### HelloWorldLambda.Tests
    ### FunctionTest.cs
    ### HelloWorldLambda.Tests.csproj
```

4. Apri il file `HelloWorldStack.cs` nella directory `src/HelloWorld`. Sostituisci il contenuto del file con il seguente codice.

```
using Amazon.CDK;
using Amazon.CDK.AWS.Lambda;
using Amazon.CDK.AWS.Logs;
using Constructs;

namespace CdkTest
{
    public class HelloWorldStack : Stack
```

```

    {
        internal HelloWorldStack(Construct scope, string id, IStackProps props =
null) : base(scope, id, props)
        {
            var buildOption = new BundlingOptions()
            {
                Image = Runtime.DOTNET_8.BundlingImage,
                User = "root",
                OutputType = BundlingOutput.ARCHIVED,
                Command = new string[]{
"/bin/sh",
                "-c",
                " dotnet tool install -g Amazon.Lambda.Tools"+
                " && dotnet build"+
                " && dotnet lambda package --output-package /asset-output/
function.zip"
                }
            };

            var helloWorldLambdaFunction = new Function(this,
"HelloWorldFunction", new FunctionProps
            {
                Runtime = Runtime.DOTNET_8,
                MemorySize = 1024,
                LogRetention = RetentionDays.ONE_DAY,
                Handler =
"HelloWorldLambda::HelloWorldLambda.Function::FunctionHandler",
                Code = Code.FromAsset("./src/HelloWorldLambda/src/
HelloWorldLambda", new Amazon.CDK.AWS.S3.Assets.AssetOptions
                {
                    Bundling = buildOption
                }
            ));
        }
    }
}

```

Questo è il codice per compilare e raggruppare il codice dell'applicazione, nonché la definizione della funzione Lambda stessa. L'oggetto `BundlingOptions` consente di creare un file zip, insieme a una serie di comandi utilizzati per generare il contenuto del file zip. In questa istanza, il comando `dotnet lambda package` viene utilizzato per compilare e generare il file zip.

5. Per implementare l'applicazione, emetti il comando riportato di seguito.

```
cdk deploy
```

6. Invoca la funzione Lambda implementata utilizzando la CLI .NET Lambda.

```
dotnet lambda invoke-function HelloWorldFunction -p "hello world"
```

7. Una volta terminato il test, potrai eliminare le risorse create (a meno che non si desideri mantenerle). Per eliminare le risorse, emetti il seguente comando.

```
cdk destroy
```

## Fasi successive

Per saperne di più sull'utilizzo AWS CDK per creare e distribuire funzioni Lambda usando .NET, consulta le seguenti risorse:

- [Lavorare con il AWS CDK in C#](#)
- [Crea, impacchetta e pubblica funzioni .NET C# Lambda con il CDK AWS](#)

## Distribuisci ASP.NET applicazioni

Oltre a ospitare funzioni basate sugli eventi, puoi anche usare .NET with Lambda per ospitare applicazioni leggere. ASP.NET È possibile creare e distribuire ASP.NET applicazioni utilizzando il pacchetto. Amazon.Lambda.AspNetCoreServer NuGet In questa sezione, imparerai come distribuire un'API ASP.NET Web su Lambda utilizzando gli strumenti CLI.NET Lambda.

## Argomenti

- [Prerequisiti](#)
- [Distribuzione di un'API ASP.NET Web su Lambda](#)
- [Distribuzione di API minime su Lambda ASP.NET](#)

## Prerequisiti

### SDK .NET 8

Installa l'SDK [.NET 8](#) e il Core Runtime. ASP.NET

## Amazon.Lambda.Tools

Per creare le funzioni Lambda, usa l'[estensione degli strumenti globali .NET Amazon.Lambda.Tools](#). Per installare Amazon.Lambda.Tools, esegui il comando riportato:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Per ulteriori informazioni sull'Amazon.Lambda.Tools estensione.NET CLI, consulta il repository [AWS Extensions for .NET CLI su GitHub](#)

## Amazon.Lambda.Templates

Per generare il codice della funzione Lambda, usa il pacchetto. [Amazon.Lambda.Templates](#) NuGet Per installare questo pacchetto del modello, esegui il comando riportato:

```
dotnet new --install Amazon.Lambda.Templates
```

## Distribuzione di un'API ASP.NET Web su Lambda

Per distribuire un'API Web utilizzando ASP.NET, puoi utilizzare i modelli .NET Lambda per creare un nuovo progetto di API Web. Usa il comando seguente per inizializzare un nuovo progetto di API ASP.NET web. Nel comando di esempio, diamo un nome al progetto AspNetOnLambda.

```
dotnet new serverless.AspNetCoreWebAPI -n AspNetOnLambda
```

Questo comando crea i seguenti file e directory nella directory di progetto.

```
.
### AspNetOnLambda
  ### src
  #   ### AspNetOnLambda
  #     ### AspNetOnLambda.csproj
  #     ### Controllers
  #     #   ### ValuesController.cs
  #     ### LambdaEntryPoint.cs
  #     ### LocalEntryPoint.cs
  #     ### Readme.md
  #     ### Startup.cs
  #     ### appsettings.Development.json
  #     ### appsettings.json
  #     ### aws-lambda-tools-defaults.json
```

```
#      ### serverless.template
### test
    ### AspNetOnLambda.Tests
        ### AspNetOnLambda.Tests.csproj
        ### SampleRequests
        #      ### ValuesController-Get.json
        ### ValuesControllerTests.cs
        ### appsettings.json
```

Quando Lambda richiama la funzione, il punto di ingresso che utilizza è il file `LambdaEntryPoint.cs`. Il file creato dal modello Lambda .NET contiene il seguente codice.

```
namespace AspNetOnLambda;

public class LambdaEntryPoint : Amazon.Lambda.AspNetCoreServer.APIGatewayProxyFunction
{
    protected override void Init(IWebHostBuilder builder)
    {
        builder
            .UseStartup#Startup#();
    }

    protected override void Init(IHostBuilder builder)
    {
    }
}
```

Il punto di ingresso utilizzato da Lambda deve ereditare da una delle tre classi base del pacchetto `Amazon.Lambda.AspNetCoreServer`. Queste tre classi base sono:

- `APIGatewayProxyFunction`
- `APIGatewayHttpApiV2ProxyFunction`
- `ApplicationLoadBalancerFunction`

La classe predefinita utilizzata quando si crea il file `LambdaEntryPoint.cs` utilizzando il modello Lambda .NET fornito è `APIGatewayProxyFunction`. La classe base che usi nella tua funzione dipende dal livello API che precede la tua funzione Lambda.

Ciascuna delle tre classi base contiene un metodo pubblico denominato `FunctionHandlerAsync`. Il nome di questo metodo fa parte della stringa del [gestore utilizzata](#) da Lambda per richiamare

la funzione. Il `FunctionHandlerAsync` metodo trasforma il payload dell'evento in entrata nel ASP.NET formato corretto e la risposta in un payload di ASP.NET risposta Lambda. Per il progetto `AspNetOnLambda` di esempio mostrato, la stringa del gestore sarebbe la seguente.

```
AspNetOnLambda::AspNetOnLambda.LambdaEntryPoint::FunctionHandlerAsync
```

Per implementare l'API in Lambda, emetti i seguenti comandi per navigare nella directory contenente il file del codice sorgente e implementa la funzione utilizzando CloudFormation.

```
cd AspNetOnLambda/src/AspNetOnLambda
dotnet lambda deploy-serverless
```

### Tip

Quando distribuisce un'API utilizzando il **`dotnet lambda deploy-serverless`** comando, CloudFormation assegna alla tua funzione Lambda un nome basato sul nome dello stack specificato durante la distribuzione. Per assegnare un nome personalizzato alla funzione Lambda, modifica il file `serverless.template` in modo da aggiungere una proprietà `FunctionName` alla risorsa `AWS::Serverless::Function`. Consulta [Tipo di nome](#) nella Guida per l'utente di CloudFormation per ulteriori informazioni.

## Distribuzione di API minime su Lambda ASP.NET

Per distribuire un'API ASP.NET minima su Lambda, puoi utilizzare i modelli .NET Lambda per creare un nuovo progetto API minimo. Utilizza il comando seguente per inizializzare un nuovo progetto di API minima ASP.NET. In questo esempio, al progetto assegnamo il nome `MinimalApiOnLambda`.

```
dotnet new serverless.AspNetCoreMinimalAPI -n MinimalApiOnLambda
```

Questo comando crea i seguenti file e directory nella directory di progetto.

```
### MinimalApiOnLambda
  ### src
    ### MinimalApiOnLambda
      ### Controllers
      #   ### CalculatorController.cs
      ### MinimalApiOnLambda.csproj
      ### Program.cs
```

```
### Readme.md
### appsettings.Development.json
### appsettings.json
### aws-lambda-tools-defaults.json
### serverless.template
```

Il file `Program.cs` contiene il seguente codice.

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();

// Add AWS Lambda support. When application is run in Lambda Kestrel is swapped out as
// the web server with Amazon.Lambda.AspNetCoreServer. This
// package will act as the webserver translating request and responses between the
// Lambda event source and ASP.NET Core.
builder.Services.AddAWSLambdaHosting(LambdaEventSource.RestApi);

var app = builder.Build();

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();

app.MapGet("/", () => "Welcome to running ASP.NET Core Minimal API on AWS Lambda");

app.Run();
```

Per configurare la tua API minima per l'esecuzione su Lambda, potresti dover modificare questo codice in modo che le richieste e le risposte tra Lambda e ASP.NET Core siano tradotte correttamente. Per impostazione predefinita, la funzione è configurata per un'origine di eventi della REST API. Per un'API HTTP o un Application Load Balancer, sostituisci (*LambdaEventSource.RestApi*) con una delle seguenti opzioni:

- (*LambdaEventSource.HttpApi*)
- (*LambdaEventSource.ApplicationLoadBalancer*)

Per implementare l'API minima in Lambda, emetti i seguenti comandi per navigare nella directory contenente il file del codice sorgente e implementa la funzione utilizzando CloudFormation.



```
cd MinimalApiOnLambda/src/MinimalApiOnLambda  
dotnet lambda deploy-serverless
```

## Utilizzo dei livelli per le funzioni Lambda .NET

Consigliamo di non utilizzare i [livelli](#) per gestire le dipendenze per le funzioni Lambda scritte in .NET. Il motivo è che .NET è un linguaggio compilato, e le funzioni devono comunque caricare manualmente gli assembly condivisi in memoria durante la fase di [inizializzazione](#), per cui i tempi di avvio a freddo possono aumentare. L'uso dei livelli non solo complica il processo di implementazione, ma impedisce anche di sfruttare le ottimizzazioni integrate del compilatore.

Per utilizzare dipendenze esterne con i gestori .NET, includile direttamente nel pacchetto di implementazione durante la fase di compilazione. In questo modo, semplifichi il processo di implementazione e sfrutti anche le ottimizzazioni integrate del compilatore .NET. Per un esempio di come importare e utilizzare dipendenze come pacchetti NuGet nella tua funzione, consulta [the section called “Gestore”](#).

## Distribuzione delle funzioni .NET Lambda con immagini di container

Esistono tre modi per creare un'immagine di container per una funzione Lambda in .NET:

- [Utilizzo di un'immagine AWS di base per .NET](#)

[Le immagini di base AWS](#) sono precaricate con un runtime in linguaggio, un client di interfaccia di runtime per gestire l'interazione tra Lambda e il codice della funzione e un emulatore di interfaccia di runtime per i test locali.

- [Usando un AWS OS-only immagine di base](#)

[AWS Le immagini di base solo](#) per il sistema operativo contengono una distribuzione Amazon Linux e l'emulatore [di interfaccia di runtime](#). Queste immagini vengono comunemente utilizzate per creare immagini di container per linguaggi compilati, come [Go](#) e [Rust](#), e per un linguaggio o una versione di linguaggio per cui Lambda non fornisce un'immagine di base, come Node.js 19. Puoi anche utilizzare immagini di base solo per il sistema operativo al fine di implementare un [runtime personalizzato](#). Per rendere l'immagine compatibile con Lambda, devi includere il [client di interfaccia di runtime per .NET](#) nell'immagine.

- [Utilizzo AWS di un'immagine non di base](#)

È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere il [client di interfaccia di runtime per .NET](#) nell'immagine.

### Tip

Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta [Utilizzo di compilazioni a più fasi](#) nella documentazione Docker. Per creare immagini di container efficienti, segui le [best practice per scrivere file Docker](#).

Questa pagina spiega come creare, testare e implementare le immagini di container per Lambda.

### Argomenti

- [AWS immagini di base per .NET](#)
- [Utilizzo di un'immagine AWS di base per .NET](#)

- [Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime](#)

AWS immagini di base per.NET

AWS fornisce le seguenti immagini di base per.NET:

| Tag | Runtime | Sistema operativo | Dockerfile                                      | Raggiunta obsolescenza |
|-----|---------|-------------------|-------------------------------------------------|------------------------|
| 10  | .NET 10 | Amazon Linux 2023 | <a href="#">Dockerfile per.NET 10 su GitHub</a> | 14 novembre 2028       |
| 9   | .NET 9  | Amazon Linux 2023 | <a href="#">Dockerfile per.NET 9 su GitHub</a>  | 10 novembre 2026       |
| 8   | .NET 8  | Amazon Linux 2023 | <a href="#">Dockerfile per.NET 8 su GitHub</a>  | 10 novembre 2026       |

[Archivio Amazon ECR: gallery.ecr.aws/lambda/dotnet](#)

Utilizzo di un'immagine AWS di base per.NET

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [SDK .NET](#): i passaggi seguenti utilizzano l'immagine di base .NET 8. Assicurati che la tua versione di .NET corrisponda alla versione dell'[immagine di base](#) specificata nel tuo Dockerfile.
- [Docker](#) (versione minima 25.0.0)
- Il [plugin buildx](#) di Docker.

Creazione e implementazione di un'immagine utilizzando un'immagine di base

Nei passaggi seguenti, utilizzerai [Amazon.Lambda.Templates](#) e [Amazon.Lambda.Tools](#) per creare un progetto .NET. Quindi, crei un'immagine Docker, carichi l'immagine su Amazon ECR e la implementi su una funzione Lambda.

1. Installa il pacchetto [Amazon.Lambda.Templates](#). NuGet

```
dotnet new install Amazon.Lambda.Templates
```

2. Crea un progetto .NET utilizzando il modello `lambda.image.EmptyFunction`.

```
dotnet new lambda.image.EmptyFunction --name MyFunction --region us-east-1
```

I file di progetto sono archiviati nella directory `MyFunction/src/MyFunction`:

- `aws-lambda-tools-defaults.json`: specifica le opzioni della riga di comando per la distribuzione della funzione Lambda.
- `Function.cs`: codice della funzione del gestore Lambda. Si tratta di un modello C# che include la libreria `Amazon.Lambda.Core` e un attributo `LambdaSerializer` predefiniti. Per ulteriori informazioni sui requisiti e sulle opzioni di serializzazione, consulta la pagina [Serializzazione nelle funzioni Lambda C#](#). A fini di test, puoi utilizzare il codice fornito o sostituirlo con il tuo codice personalizzato.
- `MyFunction.csproj`: un file di [progetto.NET, che elenca i file](#) e gli assembly che compongono l'applicazione.
- `Dockerfile`: a fini di test, puoi utilizzare il `Dockerfile` fornito o sostituirlo con un file personalizzato. Se utilizzi un file personalizzato, assicurati di:
  - Imposta la proprietà `FROM` sull'[URI dell'immagine di base](#). L'immagine di base e la `TargetFramework` nel file `MyFunction.csproj` devono utilizzare entrambe la stessa versione di .NET. Ad esempio, per utilizzare .NET 9:
    - File Docker: `FROM public.ecr.aws/lambda/dotnet:9`
    - `MyFunction.csproj`: `<TargetFramework>net9.0</TargetFramework>`
  - Imposta l'argomento `CMD` specificando il gestore della funzione Lambda. Questo dovrebbe corrispondere a `image-command` in `aws-lambda-tools-defaults.json`.

3. Installa lo [strumento globale .NET](#) `Amazon.Lambda.Tools`.

```
dotnet tool install -g Amazon.Lambda.Tools
```

Se `Amazon.Lambda.Tools` è già installato, assicurati di disporre della versione più recente.

```
dotnet tool update -g Amazon.Lambda.Tools
```

4. Passa alla directory `MyFunction/src/MyFunction`, se non lo hai ancora fatto.

```
cd src/MyFunction
```

5. Utilizza Amazon.Lambda.Tools per creare l'immagine Docker, trasferirla in un nuovo repository Amazon ECR e implementare la funzione Lambda.

Per `--function-role`, specifica il nome del ruolo, non il nome della risorsa Amazon (ARN), del [ruolo di esecuzione](#) della funzione. Ad esempio, `lambda-role`.

```
dotnet lambda deploy-function MyFunction --function-role lambda-role
```

Per ulteriori informazioni sullo strumento globale Amazon.Lambda.Tools, consulta l'archivio Extensions [AWS for .NET CLI](#) su GitHub

6. Richiama la funzione.

```
dotnet lambda invoke-function MyFunction --payload "Testing the function"
```

in caso di esito positivo, vedrai una risposta simile alla seguente:

Payload:

```
{"Lower":"testing the function","Upper":"TESTING THE FUNCTION"}
```

Log Tail:

```
INIT_REPORT Init Duration: 9999.81 ms   Phase: init       Status: timeout
START RequestId: 12378346-f302-419b-b1f2-deaa1e8423ed Version: $LATEST
END RequestId: 12378346-f302-419b-b1f2-deaa1e8423ed
REPORT RequestId: 12378346-f302-419b-b1f2-deaa1e8423ed   Duration: 3173.06 ms
      Billed Duration: 3174 ms                Memory Size: 512 MB      Max Memory Used: 24 MB
```

7. Elimina la funzione Lambda.

```
dotnet lambda delete-function MyFunction
```

Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

Se utilizzi un'[immagine di base solo per il sistema operativo](#) o un'immagine di base alternativa, devi includere il client dell'interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime estende l'[API Runtime](#), che gestisce l'interazione tra Lambda e il codice della funzione.

L'esempio seguente mostra come creare un'immagine contenitore per .NET utilizzando un'immagine non di AWS base e come aggiungere [Amazon.Lambda.RuntimeSupport](#) pacchetto, che è il client dell'interfaccia di runtime Lambda per .NET. Il Dockerfile di esempio utilizza l'immagine di base Microsoft .NET 8.

## Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- [.NET SDK](#): i passaggi seguenti utilizzano un'immagine di base .NET 9. Assicurati che la tua versione di .NET corrisponda alla versione dell'immagine di base specificata nel tuo Dockerfile.
- [Docker](#) (versione minima 25.0.0)
- Il [plugin buildx](#) di Docker.

Creazione e implementazione di un'immagine utilizzando un'immagine di base alternativa

1. Installa il pacchetto [NuGet Amazon.Lambda.Templates](#).

```
dotnet new install Amazon.Lambda.Templates
```

2. Crea un progetto .NET utilizzando il modello `lambda.CustomRuntimeFunction`. [Questo modello include Amazon.Lambda.RuntimeSupport](#) pacchetto.

```
dotnet new lambda.CustomRuntimeFunction --name MyFunction --region us-east-1
```

3. Passa alla directory `MyFunction/src/MyFunction`. Qui sono archiviati i file del progetto. Esamina i seguenti file di log:
  - `aws-lambda-tools-defaults.json`: in questo file si specificano le opzioni della riga di comando quando si distribuisce la funzione Lambda.
  - `Function.cs`: il codice contiene una classe con un metodo `Main` che inizializza la libreria `Amazon.Lambda.RuntimeSupport` come bootstrap. Il metodo `Main` sarà il punto di ingresso per il processo della funzione. Il metodo `Main` racchiude il gestore della funzione in un wrapper con cui il bootstrap può funzionare. [Per ulteriori informazioni, consulta Using Amazon.Lambda.RuntimeSupport come libreria di classi](#) nel repository. GitHub
  - `MyFunction.csproj` — Un file di [progetto.NET, che elenca i file](#) e gli assembly che compongono l'applicazione.
  - `Readme.md`: questo file contiene ulteriori informazioni sulla funzione Lambda di esempio.

4. Apri il file `aws-lambda-tools-defaults.json` e aggiungi le righe seguenti.

```
"package-type": "image",  
"docker-host-build-output-dir": "./bin/Release/lambda-publish"
```

- `package-type`: definisce il pacchetto di implementazione come immagine di container.
- `docker-host-build-output-dir`: imposta la directory di output per il processo di compilazione.

Example `aws-lambda-tools-defaults.json`

```
{  
  "Information": [  
    "This file provides default values for the deployment wizard inside Visual  
    Studio and the AWS Lambda commands added to the .NET Core CLI.",  
    "To learn more about the Lambda commands with the .NET Core CLI execute the  
    following command at the command line in the project root directory.",  
    "dotnet lambda help",  
    "All the command line options for the Lambda command can be specified in this  
    file."  
  ],  
  "profile": "",  
  "region": "us-east-1",  
  "configuration": "Release",  
  "function-runtime": "provided.al2023",  
  "function-memory-size": 256,  
  "function-timeout": 30,  
  "function-handler": "bootstrap",  
  "msbuild-parameters": "--self-contained true",  
  "package-type": "image",  
  "docker-host-build-output-dir": "./bin/Release/lambda-publish"  
}
```

5. Crea un Dockerfile nella directory `MyFunction/src/MyFunction`. Il seguente Dockerfile di esempio utilizza un'immagine di base Microsoft .NET anziché un'[immagine di base AWS](#).

- Imposta la proprietà FROM sull'identificativo dell'immagine di base. L'immagine di base e la TargetFramework nel file `MyFunction.csproj` devono utilizzare entrambe la stessa versione di .NET.
- Utilizza il comando COPY per copiare la funzione nella directory `/var/task`.



- Imposta l'ENTRYPOINT sul modulo su cui desideri che il container Docker venga eseguito all'avvio. In questo caso, il modulo è il bootstrap, che inizializza la libreria `Amazon.Lambda.RuntimeSupport`.

Nota che l'esempio Dockerfile non include un'istruzione [USER](#). Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

### Example Dockerfile

```
# You can also pull these images from DockerHub amazon/aws-lambda-dotnet:8
FROM mcr.microsoft.com/dotnet/runtime:9.0

# Set the image's internal work directory
WORKDIR /var/task

# Copy function code to Lambda-defined environment variable
COPY "bin/Release/net9.0/linux-x64" .

# Set the entrypoint to the bootstrap
ENTRYPOINT ["/usr/bin/dotnet", "exec", "/var/task/bootstrap.dll"]
```

6. Installa lo [strumento globale .NET Core](#) `Amazon.Lambda.Tools`.

```
dotnet tool install -g Amazon.Lambda.Tools
```

Se `Amazon.Lambda.Tools` è già installato, assicurati di disporre della versione più recente.

```
dotnet tool update -g Amazon.Lambda.Tools
```

7. Utilizza `Amazon.Lambda.Tools` per creare l'immagine Docker, trasferirla in un nuovo repository Amazon ECR e implementare la funzione Lambda.

Per `--function-role`, specifica il nome del ruolo, non il nome della risorsa Amazon (ARN), del [ruolo di esecuzione](#) della funzione. Ad esempio, `lambda-role`.

```
dotnet lambda deploy-function MyFunction --function-role lambda-role
```

[Per ulteriori informazioni sull'estensione CLI di Amazon.Lambda.Tools, consulta l'archivio Extensions for .NET CLI su.AWS GitHub](#)

8. Richiama la funzione.

```
dotnet lambda invoke-function MyFunction --payload "Testing the function"
```

In caso di esito positivo, viene visualizzato quanto segue:

Payload:

"TESTING THE FUNCTION"

Log Tail:

START RequestId: *id* Version: \$LATEST

END RequestId: *id*

|                             |                        |                       |        |
|-----------------------------|------------------------|-----------------------|--------|
| REPORT RequestId: <i>id</i> | Duration: 0.99 ms      | Billed Duration: 1 ms | Memory |
| Size: 256 MB                | Max Memory Used: 12 MB |                       |        |

9. Elimina la funzione Lambda.

```
dotnet lambda delete-function MyFunction
```

## Compilare il codice della funzione Lambda .NET in un formato di runtime nativo

.NET 8 supporta la compilazione nativa in anticipo (AOT). Con AOT nativa, puoi compilare il codice della funzione Lambda in un formato di runtime nativo, che elimina la necessità di compilare il codice .NET in fase di runtime. La compilazione AOT nativa può ridurre il tempo di avvio a freddo delle funzioni Lambda scritte in .NET. Per ulteriori informazioni, consulta [Introducing the .NET 8 runtime for AWS Lambda](#) nel AWS Compute Blog.

### Sections

- [Runtime Lambda](#)
- [Prerequisiti](#)
- [Nozioni di base](#)
- [Serializzazione](#)
- [Rifinitura](#)
- [Risoluzione dei problemi](#)

### Runtime Lambda

Per implementare una funzione Lambda creata con la compilazione AOT nativa, usa il runtime Lambda .NET 8 gestito. Questo runtime supporta l'uso sia di architetture x86\_64 che di architetture arm64.

Quando si implementa una funzione Lambda .NET, l'applicazione viene prima compilata in codice IL (Intermediate Language). Al runtime, il compilatore just-in-time (JIT) nel runtime Lambda prende il codice IL e lo compila in codice macchina secondo le necessità. Con una funzione Lambda compilata in anticipo con AOT nativo, si compila il codice in codice macchina quando si implementa la funzione in modo da non dipendere dal runtime .NET o dall'SDK nel runtime Lambda per compilare il codice prima che venga eseguito.

Una limitazione di AOT è che il codice dell'applicazione deve essere compilato in un ambiente con lo stesso sistema operativo Amazon Linux 2023 (AL2023) utilizzando dal runtime .NET 8. La CLI Lambda .NET offre funzionalità per compilare l'applicazione in un container Docker utilizzando un'immagine AL2023.

Per evitare potenziali problemi di compatibilità tra le architetture, consigliamo vivamente di compilare il codice in un ambiente con la stessa architettura di processore configurata per la funzione. Per

ulteriori informazioni sulle limitazioni della compilazione tra architetture diverse, consulta [Cross-compilation](#) la documentazione di Microsoft.NET.

## Prerequisiti

### Docker

Per utilizzare l'AOT nativo, il codice della funzione deve essere compilato in un ambiente con lo stesso sistema operativo AL2023 del runtime .NET 8. I comandi della CLI .NET nelle seguenti sezioni utilizzano Docker per sviluppare e creare funzioni Lambda in un ambiente AL2023.

### SDK .NET 8

La compilazione AOT nativa è una funzionalità di .NET 8. È necessario installare l'[SDK .NET 8](#) sulla macchina di compilazione, non solo il runtime.

### Amazon.Lambda.Tools

Per creare le funzioni Lambda, usa l'[estensione degli strumenti globali .NET Amazon.Lambda.Tools](#). Per installare Amazon.Lambda.Tools, esegui il comando riportato:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Per ulteriori informazioni sull'Amazon.Lambda.Tools estensione.NET CLI, vedi il repository [AWS Extensions for .NET](#) CLI su GitHub

### Amazon.Lambda.Templates

Per generare il codice della funzione Lambda, usa il pacchetto. [Amazon.Lambda.Templates](#) NuGet Per installare questo pacchetto del modello, esegui il comando riportato:

```
dotnet new install Amazon.Lambda.Templates
```

## Nozioni di base

Sia l'interfaccia a riga di comando globale .NET che AWS Serverless Application Model (AWS SAM) forniscono modelli introduttivi per la creazione di applicazioni utilizzando AOT nativo. Per creare la tua prima funzione Lambda AOT nativa, completa le operazioni riportate nelle seguenti istruzioni.

## Inizializzazione e distribuzione di una funzione Lambda compilata in AOT nativa

1. Inizializza un nuovo progetto utilizzando il modello AOT nativo, quindi naviga nella directory contenente i file `.cs` e `.csproj` creati. In questo esempio, diamo un nome alla nostra funzione `NativeAotSample`.

```
dotnet new lambda.NativeAOT -n NativeAotSample
cd ./NativeAotSample/src/NativeAotSample
```

Il file `Function.cs` creato dal modello AOT nativo contiene il seguente codice di funzione.

```
using Amazon.Lambda.Core;
using Amazon.Lambda.RuntimeSupport;
using Amazon.Lambda.Serialization.SystemTextJson;
using System.Text.Json.Serialization;

namespace NativeAotSample;

public class Function
{
    /// <summary>
    /// The main entry point for the Lambda function. The main function is called
    /// once during the Lambda init phase. It
    /// initializes the .NET Lambda runtime client passing in the function handler
    /// to invoke for each Lambda event and
    /// the JSON serializer to use for converting Lambda JSON format to the .NET
    /// types.
    /// </summary>
    private static async Task Main()
    {
        Func<string, ILambdaContext, string> handler = FunctionHandler;
        await LambdaBootstrapBuilder.Create(handler, new
        SourceGeneratorLambdaJsonSerializer<LambdaFunctionJsonSerializerContext>())
            .Build()
            .RunAsync();
    }

    /// <summary>
    /// A simple function that takes a string and does a ToUpper.
    ///
    /// To use this handler to respond to an AWS event, reference the appropriate
    /// package from
```

```

    /// https://github.com/aws/aws-lambda-dotnet#events
    /// and change the string input parameter to the desired event type. When the
    event type
    /// is changed, the handler type registered in the main method needs to be
    updated and the LambdaFunctionJsonSerializerContext
    /// defined below will need the JsonSerializable updated. If the return type
    and event type are different then the
    /// LambdaFunctionJsonSerializerContext must have two JsonSerializable
    attributes, one for each type.
    ///
    /// When using Native AOT extra testing with the deployed Lambda functions is
    required to ensure
    /// the libraries used in the Lambda function work correctly with Native AOT. If
    a runtime
    /// error occurs about missing types or methods the most likely solution will be
    to remove references to trim-unsafe
    /// code or configure trimming options. This sample defaults to partial TrimMode
    because currently the AWS
    /// SDK for .NET does not support trimming. This will result in a larger
    executable size, and still does not
    /// guarantee runtime trimming errors won't be hit.
    /// </summary>
    /// <param name="input"></param>
    /// <param name="context"></param>
    /// <returns></returns>
    public static string FunctionHandler(string input, ILambdaContext context)
    {
        return input.ToUpper();
    }
}

/// <summary>
/// This class is used to register the input event and return type for the
    FunctionHandler method with the System.Text.Json source generator.
/// There must be a JsonSerializable attribute for each type used as the input and
    return type or a runtime error will occur
/// from the JSON serializer unable to find the serialization information for
    unknown types.
/// </summary>
[JsonSerializable(typeof(string))]
public partial class LambdaFunctionJsonSerializerContext : JsonSerializerContext
{
    /// By using this partial class derived from JsonSerializerContext, we can
    generate reflection free JSON Serializer code at compile time

```

```
// which can deserialize our class and properties. However, we must attribute
this class to tell it what types to generate serialization code for.
// See https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-
text-json-source-generation
```

AOT nativo compila l'applicazione in un unico binario nativo. Il punto di ingresso di quel binario è il metodo `static Main`. All'interno di `static Main`, viene avviato il runtime Lambda e viene impostato il metodo `FunctionHandler`. Come parte del bootstrap di runtime, un serializzatore generato dal codice sorgente viene configurato utilizzando `new SourceGeneratorLambdaJsonSerializer<LambdaFunctionJsonSerializerContext>()`

2. Per distribuire l'applicazione in Lambda, assicurati che Docker sia in esecuzione nel tuo ambiente locale ed esegui il comando riportato.

```
dotnet lambda deploy-function
```

Dietro le quinte, la CLI globale .NET scarica un'immagine Docker di AL2023 e compila il codice dell'applicazione all'interno di un container in esecuzione. Il file binario compilato viene restituito al file system locale prima di essere implementato su Lambda.

3. Verifica la tua funzione eseguendo il comando riportato. Sostituisci `<FUNCTION_NAME>` con il nome scelto per la funzione nella procedura guidata di implementazione.

```
dotnet lambda invoke-function <FUNCTION_NAME> --payload "hello world"
```

La risposta della CLI include dettagli sulle prestazioni per l'avvio a freddo (durata di inizializzazione) e il runtime totale per l'invocazione della funzione.

4. Per eliminare le AWS risorse create seguendo i passaggi precedenti, esegui il comando seguente. Sostituisci `<FUNCTION_NAME>` con il nome scelto per la funzione nella procedura guidata di implementazione. Eliminando AWS le risorse che non utilizzi più, eviti che vengano addebitati costi non necessari al tuo. Account AWS

```
dotnet lambda delete-function <FUNCTION_NAME>
```

## Serializzazione

Per implementare le funzioni in Lambda utilizzando AOT nativo, il codice della funzione deve utilizzare la [serializzazione generata dal codice sorgente](#). Invece di utilizzare la riflessione in fase

di esecuzione per raccogliere i metadati necessari per accedere alle proprietà degli oggetti per la serializzazione, i generatori di codice sorgente generano file sorgente C# che vengono compilati durante la creazione dell'applicazione. Per configurare correttamente il serializzatore generato dal codice sorgente, assicurati di includere tutti gli oggetti di input e output utilizzati dalla funzione, nonché tutti i tipi personalizzati. Ad esempio, una funzione Lambda che riceve eventi da una REST API di Gateway API e restituisce un tipo `Product` personalizzato includerebbe un serializzatore definito come segue.

```
[JsonSerializable(typeof(APIGatewayProxyRequest))]  
[JsonSerializable(typeof(APIGatewayProxyResponse))]  
[JsonSerializable(typeof(Product))]  
public partial class CustomSerializer : JsonSerializerContext  
{  
}
```

## Rifinitura

L'AOT nativo rifinisce il codice dell'applicazione come parte della compilazione per garantire che il file binario sia il più piccolo possibile. .NET 8 for Lambda offre un supporto di rifinitura migliorato rispetto alle versioni precedenti di .NET. È stato aggiunto il supporto alle [librerie di runtime Lambda](#), all'[SDK AWS .NET](#) alle [annotazioni Lambda .NET](#) e a .NET 8 stesso.

Questi miglioramenti offrono la possibilità di eliminare gli avvisi di taglio in fase di compilazione, ma .NET non è mai completamente sicuro. Ciò significa che parti delle librerie su cui si basa la funzione potrebbero essere eliminate durante la fase di compilazione. Puoi gestire questo problema definendo `TrimmerRootAssemblies` come parte del tuo file `.csproj`, come mostrato nell'esempio seguente.

```
<ItemGroup>  
  <TrimmerRootAssembly Include="AWSSDK.Core" />  
  <TrimmerRootAssembly Include="AWSXRayRecorder.Core" />  
  <TrimmerRootAssembly Include="AWSXRayRecorder.Handlers.AwsSdk" />  
  <TrimmerRootAssembly Include="Amazon.Lambda.APIGatewayEvents" />  
  <TrimmerRootAssembly Include="bootstrap" />  
  <TrimmerRootAssembly Include="Shared" />  
</ItemGroup>
```

Tieni presente che quando ricevi un avviso di rifinitura, l'aggiunta della classe che genera l'avviso `TrimmerRootAssembly` potrebbe non risolvere il problema. Un avviso di rifinitura indica che la



classe sta provando ad accedere a un'altra classe che non può essere determinata fino al runtime. Per evitare errori di runtime, aggiungi questa seconda classe a `TrimmerRootAssembly`.

Per ulteriori informazioni sulla gestione degli avvisi di rifinitura, consulta [Introduzione agli avvisi di rifinitura](#) nella documentazione di Microsoft .NET.

## Risoluzione dei problemi

Errore: la compilazione Cross-OS nativa non è supportata.

La tua versione dello strumento globale .NET Core di Amazon.Lambda.Tools non è aggiornata. Esegui l'aggiornamento alla versione più recente e riprova.

Docker: il sistema operativo di immagini "linux" non può essere utilizzato su questa piattaforma.

Docker sul tuo sistema è configurato per utilizzare container Windows. Passa ai container Linux per eseguire l'ambiente della compilazione AOT nativa.

Per ulteriori informazioni sugli errori più comuni, vedere il repository [AWS NativeAOT for .NET su GitHub](#)

## Utilizzo dell'oggetto del contesto Lambda per recuperare le informazioni sulla funzione C#

Quando Lambda esegue la funzione, passa un oggetto Context al [gestore](#). Questo oggetto fornisce proprietà con informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

### Proprietà del contesto

- `FunctionName`: il nome della funzione Lambda.
- `FunctionVersion`: la [versione](#) della funzione.
- `InvokedFunctionArn`: l'Amazon Resource Name (ARN) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `MemoryLimitInMB`: la quantità di memoria allocata per la funzione.
- `AwsRequestId`: l'identificatore della richiesta di invocazione.
- `LogGroupName`: il gruppo di log per la funzione.
- `LogStreamName`: il flusso di log per l'istanza della funzione.
- `RemainingTime (TimeSpan)`: il numero di millisecondi rimasti prima del timeout dell'esecuzione.
- `Identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
- `ClientContext`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.
- `Logger` L'[oggetto logger](#) per la funzione.

Puoi utilizzare le informazioni riportate nell'oggetto `ILambdaContext` per generare informazioni sull'invocazione della funzione a scopo di monitoraggio. Il seguente codice fornisce un esempio di come aggiungere informazioni di contesto a un framework di registrazione strutturato. In questo esempio, la funzione aggiunge `AwsRequestId` agli output di log. La funzione utilizza anche la proprietà `RemainingTime` per annullare un'attività in transito se il timeout della funzione Lambda sta per essere raggiunto.

```
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))

namespace GetProductHandler;

public class Function
```

```
{
    private readonly IDatabaseRepository _repo;

    public Function()
    {
        this._repo = new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
request, ILambdaContext context)
    {
        Logger.AppendKey("AwsRequestId", context.AwsRequestId);

        var id = request.PathParameters["id"];

        using var cts = new CancellationTokenSource();

        try
        {
            cts.CancelAfter(context.RemainingTime.Add(TimeSpan.FromSeconds(-1)));

            var databaseRecord = await this._repo.GetById(id, cts.Token);

            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.OK,
                Body = JsonSerializer.Serialize(databaseRecord)
            };
        }
        catch (Exception ex)
        {
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.InternalServerError,
                Body = JsonSerializer.Serialize(new { error = ex.Message })
            };
        }
        finally
        {
            cts.Cancel();
        }
    }
}
```

## Registrare e monitorare le funzioni Lambda con C#

AWS Lambda monitora automaticamente le funzioni Lambda e invia le voci di registro ad Amazon. CloudWatch La tua funzione Lambda include un gruppo di CloudWatch log Logs e un flusso di log per ogni istanza della tua funzione. L'ambiente di runtime di Lambda invia al flusso di log i dettagli su ogni invocazione e altri output dal codice della funzione. Per ulteriori informazioni sui CloudWatch log, consulta [Invio dei log delle funzioni Lambda ai CloudWatch log](#).

### Sections

- [Creazione di una funzione che restituisce i registri](#)
- [Utilizzo dei controlli di registrazione avanzati di Lambda con .NET](#)
- [Strumenti e librerie di registrazione aggiuntivi](#)
- [Utilizzo di Powertools per AWS Lambda \(.NET\) e AWS SAM per la registrazione strutturata](#)
- [Visualizzazione dei log nella console Lambda](#)
- [Visualizzazione dei log nella console CloudWatch](#)
- [Visualizzazione dei log utilizzando AWS Command Line Interface \(AWS CLI\)](#)
- [Eliminazione dei log](#)

### Creazione di una funzione che restituisce i registri

Per generare i log dal codice della funzione, potete utilizzare l'oggetto [ILambdaLogger](#) on the context, i metodi della classe <https://docs.microsoft.com/en-us/dotnet/api/system.console> Console o qualsiasi libreria di registrazione che scrive su `stdout` o `stderr`.

Il runtime di .NET registra START, END e REPORT per ogni chiamata. La riga del report fornisce i seguenti dettagli.

### Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per l'invocazione.
- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.

- **Quantità max utilizzata** – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- **Durata Init** – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- **XRAY TraceId**: per le richieste tracciate, l'ID di traccia. [AWS X-Ray](#)
- **SegmentId**— Per le richieste tracciate, l'ID del X-Ray segmento.
- **Campionato** – Per le richieste tracciate, il risultato del campionamento.

## Utilizzo dei controlli di registrazione avanzati di Lambda con .NET

Per darti un maggiore controllo sul modo in cui i log delle tue funzioni vengono acquisiti, elaborati e utilizzati, puoi configurare le seguenti opzioni di registrazione per i runtime .NET supportati:

- **Formato di registro**: seleziona tra testo normale e formato JSON strutturato per i log della tua funzione
- **Livello di registro**: per i log in formato JSON, scegli il livello di dettaglio dei log a cui Lambda invia CloudWatch, ad esempio ERROR, DEBUG o INFO
- **Gruppo di log**: scegli il gruppo di CloudWatch log a cui la tua funzione invia i log

Per ulteriori informazioni su queste opzioni di registrazione e istruzioni su come configurare la funzione per utilizzarle, consulta la pagina [the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”](#).

Per utilizzare le opzioni del formato di log e del livello di log con le funzioni Lambda in .NET, consulta le istruzioni nelle sezioni seguenti.

## Utilizzo del formato di log JSON strutturato con .NET

Se selezioni JSON come formato di registro della tua funzione, Lambda invia l'output dei log utilizzando un JSON [ILambdaLogger](#) strutturato. Ogni oggetto di log JSON contiene almeno cinque coppie chiave-valore con le seguenti chiavi:

- **"timestamp"**: l'ora in cui è stato generato il messaggio di log
- **"level"**: il livello di log assegnato al messaggio
- **"requestId"**: l'ID di richiesta univoco dell'invocazione alla funzione

- "traceId": la variabile di ambiente `_X_AMZN_TRACE_ID`
- "message": il contenuto del messaggio di log

L'istanza `ILambdaLogger` può aggiungere ulteriori coppie chiave-valore, ad esempio durante la registrazione delle eccezioni. È inoltre possibile fornire parametri aggiuntivi personalizzati come descritto nella sezione [the section called “Customer-provided parametri di registro”](#).

#### Note

Se il codice utilizza già un'altra libreria di registrazione per produrre JSON-formatted registri, assicurati che il formato di registro della funzione sia impostato su testo normale. L'impostazione del formato di registro su JSON comporta una doppia codifica degli output del registro.

Il seguente comando di registrazione di esempio mostra come scrivere un messaggio di log con il livello `INFO`.

Example codice di registrazione .NET

```
context.Logger.LogInformation("Fetching cart from database");
```

Inoltre, puoi usare un metodo di log generico che utilizza il livello di log come argomento come mostrato nell'esempio seguente.

```
context.Logger.Log(LogLevel.Information, "Fetching cart from database");
```

L'output del log ottenuto da questi frammenti di codice di esempio verrà acquisito in Logs come segue: CloudWatch

Example Record di log JSON

```
{
  "timestamp": "2025-09-07T01:30:06.977Z",
  "level": "Information",
  "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
  "traceId": "1-6408af34-50f56f5b5677a7d763973804",
  "message": "Fetching cart from database"
```

```
}
```

### Note

Se configuri il formato di log della funzione per utilizzare testo semplice anziché JSON, il livello di log acquisito nel messaggio segue la convenzione di Microsoft di utilizzare un'etichetta di quattro caratteri. Ad esempio, un livello di log di Debug è rappresentato nel messaggio come `debug`.

Quando si configura la funzione per l'utilizzo di log in formato JSON, il livello di log acquisito nel log utilizza l'etichetta completa, come mostrato nel record di log JSON di esempio.

Se non assegni un livello all'output del log, Lambda gli assegna automaticamente il livello INFO.

### Registrazione delle eccezioni in JSON

Quando si utilizza la registrazione JSON strutturata con `ILambdaLogger`, è possibile registrare le eccezioni dei log nel codice come illustrato nell'esempio seguente.

### Example utilizzo della registrazione delle eccezioni

```
try
{
    connection.ExecuteQuery(query);
}
catch(Exception e)
{
    context.Logger.LogWarning(e, "Error executing query");
}
```

Il formato di log generato da questo codice è mostrato nell'esempio JSON seguente. Si noti che la proprietà `message` in JSON viene compilata utilizzando l'argomento del messaggio fornito nella chiamata `LogWarning`, mentre la proprietà `errorMessage` proviene dalla proprietà `Message` dell'eccezione stessa.

### Example Record di log JSON

```
{
  "timestamp": "2025-09-07T01:30:06.977Z",
  "level": "Warning",
```

```
{
  "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
  "traceId": "1-6408af34-50f56f5b5677a7d763973804",
  "message": "Error executing query",
  "errorType": "System.Data.SqlClient.SqlException",
  "errorMessage": "Connection closed",
  "stackTrace": ["<call exception.StackTrace>"]
}
```

Se il formato di registrazione della funzione è impostato su JSON, Lambda emette anche messaggi di JSON-formatted registro quando il codice genera un'eccezione non rilevata. Il frammento di codice e il messaggio di log di esempio seguenti mostrano come vengono registrate le eccezioni non rilevate.

### Example codice di eccezione

```
throw new ApplicationException("Invalid data");
```

### Example Record di log JSON

```
{
  "timestamp": "2025-09-07T01:30:06.977Z",
  "level": "Error",
  "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
  "traceId": "1-6408af34-50f56f5b5677a7d763973804",
  "message": "Invalid data",
  "errorType": "System.ApplicationException",
  "errorMessage": "Invalid data",
  "stackTrace": ["<call exception.StackTrace>"]
}
```

### Customer-provided parametri di registro

Con i messaggi di JSON-formatted registro, è possibile fornire parametri di registro aggiuntivi e includerli nel registromessage. Il seguente esempio di frammento di codice mostra un comando per aggiungere due parametri forniti dall'utente etichettati `retryAttempt` e `uri`. Nell'esempio, il valore di questi parametri deriva dagli argomenti `retryAttempt` e `uriDestination` passati al comando di registrazione.

### Example Comando di registrazione JSON con parametri aggiuntivi

```
context.Logger.LogInformation("Starting retry {retryAttempt} to make GET request to {uri}", retryAttempt, uriDestination);
```



L'output del messaggio di log di questo comando è mostrato nell'esempio JSON seguente.

### Example Record di log JSON

```
{
  "timestamp": "2025-09-07T01:30:06.977Z",
  "level": "Information",
  "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
  "traceId": "1-6408af34-50f56f5b5677a7d763973804",
  "message": "Starting retry 1 to make GET request to http://example.com/",
  "retryAttempt": 1,
  "uri": "http://example.com/"
}
```

#### Tip

Quando si specificano parametri aggiuntivi, è inoltre possibile utilizzare proprietà posizionali anziché nomi. A titolo illustrativo, il comando di registrazione nell'esempio precedente potrebbe essere scritto anche come segue:

```
context.Logger.LogInformation("Starting retry {0} to make GET request to {1}",
    retryAttempt, uriDestination);
```

Tieni presente che quando fornisci parametri di registrazione aggiuntivi, Lambda li acquisisce come proprietà di primo livello nel record di log JSON. Questo approccio è diverso da alcune popolari librerie di registrazione .NET come Serilog, che acquisiscono parametri aggiuntivi in un oggetto secondario separato.

Se l'argomento fornito per un parametro aggiuntivo è un oggetto complesso, per impostazione predefinita Lambda utilizza il metodo `ToString()` per fornire il valore. Per indicare che un argomento deve essere serializzato in JSON, utilizza il prefisso `@` come mostrato nel seguente frammento di codice. In questo esempio, `User` è un oggetto con proprietà `FirstName` e `LastName`.

### Example Comando di registrazione JSON con oggetto serializzato JSON

```
context.Logger.LogInformation("User {@user} logged in", User);
```

L'output del messaggio di log di questo comando è mostrato nell'esempio JSON seguente.

## Example Record di log JSON

```
{
  "timestamp": "2025-09-07T01:30:06.977Z",
  "level": "Information",
  "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
  "traceId": "1-6408af34-50f56f5b5677a7d763973804",
  "message": "User {@user} logged in",
  "user": {
    "FirstName": "John",
    "LastName": "Doe"
  }
}
```

Se l'argomento per un parametro aggiuntivo è un array o implementa `ICollection` o `IDictionary`, Lambda aggiunge l'argomento al messaggio di log JSON come matrice, come mostrato nell'esempio seguente di record di log JSON. In questo esempio, `{users}` accetta un argomento `ICollection` contenente istanze della proprietà `User` con lo stesso formato dell'esempio precedente. Lambda converte `ICollection` in un array, con ogni valore creato utilizzando il metodo `ToString`.

## Example Record di registro JSON con un argomento **ICollection**

```
{
  "timestamp": "2025-09-07T01:30:06.977Z",
  "level": "Information",
  "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
  "traceId": "1-6408af34-50f56f5b5677a7d763973804",
  "message": "{users} have joined the group",
  "users": [
    "Rosalez, Alejandro",
    "Stiles, John"
  ]
}
```

Puoi serializzare l'elenco in JSON anche utilizzando il prefisso `@` nel comando di registrazione. Nel seguente esempio di record di log JSON, la proprietà `users` è serializzata in JSON.

## Example Record di registro JSON con un argomento `ICollection` serializzato JSON

```
{
```

```
{
  "timestamp": "2025-09-07T01:30:06.977Z",
  "level": "Information",
  "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
  "traceId": "1-6408af34-50f56f5b5677a7d763973804",
  "message": "{@users} have joined the group",
  "users":
  [
    {
      "FirstName": "Alejandro",
      "LastName": "Rosalez"
    },
    {
      "FirstName": "John",
      "LastName": "Stiles"
    }
  ]
}
```

## Utilizzo del filtraggio a livello di log con .NET

Configurando il filtro a livello di registro, puoi scegliere di inviare solo i log con un determinato livello di dettaglio o inferiore a Logs. CloudWatch Per informazioni su come configurare il filtraggio a livello di log della funzione, consulta la pagina [the section called “Log-level filtraggio”](#).

Per AWS Lambda filtrare i messaggi di registro in base al livello di registro, puoi utilizzare i log in formato JSON o utilizzare i metodi.NET Console per generare i messaggi di registro. Per creare log in formato JSON, [configura il tipo di log della funzione su JSON](#) e usa l'istanza `ILambdaLogger`.

Con JSON-formatted i log, Lambda filtra gli output dei log utilizzando la coppia chiave-valore «level» nell'oggetto JSON descritto in: [the section called “Utilizzo del formato di log JSON strutturato con .NET”](#)

Se utilizzi i Console metodi.NET per scrivere messaggi CloudWatch nei log, Lambda applica i livelli di log ai tuoi messaggi come segue:

- `Console.WriteLine` metodo - Lambda applica un livello di log di INFO
- `Console.Error` metodo - Lambda applica un livello di log di ERROR

Quando configuri la tua funzione per utilizzare il filtro a livello di log, devi selezionare una delle seguenti opzioni per il livello di log che desideri che Lambda invii a Logs. CloudWatch Nota la

mappatura dei livelli di log utilizzati da Lambda con i livelli Microsoft standard utilizzati da .NET  
ILambdaLogger.

| Livello di log Lambda     | Livello Microsoft equivalente | Utilizzo standard                                                                               |
|---------------------------|-------------------------------|-------------------------------------------------------------------------------------------------|
| TRACE (dettaglio massimo) | Traccia                       | Le informazioni più dettagliate utilizzate per tracciare il percorso di esecuzione del codice   |
| DEBUG                     | Esegui il debug               | Informazioni dettagliate per il debug del sistema                                               |
| INFO                      | Informazioni                  | Messaggi che registrano il normale funzionamento della funzione                                 |
| WARN                      | Attenzione                    | Messaggi su potenziali errori che potrebbero portare a comportamenti imprevisti se non risolti  |
| ERRORE                    | Errore                        | Messaggi relativi a problemi che impediscono al codice di funzionare come previsto              |
| FATAL (dettaglio minimo)  | Critica                       | Messaggi relativi a errori gravi che causano l'interruzione del funzionamento dell'applicazione |

Lambda invia i log relativi al livello di dettaglio selezionato e inferiore a. CloudWatch Ad esempio, se si configura un livello di registro WARN, Lambda invia i log corrispondenti ai livelli WARN, ERROR e FATAL.

## Strumenti e librerie di registrazione aggiuntivi

[Powertools for AWS Lambda \(.NET\)](#) è un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori. L'[utilità di registrazione](#) fornisce un logger ottimizzato per Lambda che include informazioni aggiuntive sul contesto delle funzioni in tutte le funzioni con output strutturato come JSON. Utilizza l'utility per eseguire le seguenti operazioni:

- Acquisizione di campi essenziali dal contesto Lambda, avvii a freddo e output di registrazione della struttura come JSON
- Registrazione degli eventi di chiamata Lambda quando richiesto (disabilitata per impostazione predefinita)
- Stampa di tutti i log solo per una percentuale di chiamate tramite campionamento dei log (disabilitata per impostazione predefinita)
- Aggiunta di chiavi supplementari al log strutturato in qualsiasi momento
- Utilizzo di un formattatore di log personalizzato (Bring Your Own Formatter) per generare i log in una struttura compatibile con Logging RFC dell'organizzazione

### Utilizzo di Powertools per AWS Lambda (.NET) e AWS SAM per la registrazione strutturata

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Hello World C# di esempio con moduli [Powertools for AWS Lambda \(.NET\)](#) integrati utilizzando AWS SAM. Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a CloudWatch e AWS X-Ray. La funzione restituisce un messaggio `hello world`.

### Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- .NET 8
- [AWS CLI versione 2](#)
- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI. AWS SAM](#)

## Distribuisci un esempio AWS SAM applicazione

1. Inizializza l'applicazione utilizzando il modello Hello World. TypeScript

```
sam init --app-template hello-world-powertools-dotnet --name sam-app --package-type Zip --runtime dotnet6 --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi Enter.

### Note

Perché HelloWorldFunction potrebbe non avere l'autorizzazione definita, va bene? , assicurati di entrarey.

5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query 'Stacks[0].Outputs[?OutputKey=='HelloWorldApi'].OutputValue' --output text
```

6. Richiama l'endpoint dell'API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere i log per la funzione, esegui [sam logs](#). Per ulteriori informazioni, consulta l'argomento relativo all'[utilizzo dei log](#) nella Guida per sviluppatori AWS Serverless Application Model .

```
sam logs --stack-name sam-app
```

L'output del log ha la struttura seguente:

```
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8
2025-09-20T14:15:27.988000 INIT_START Runtime Version:
dotnet:6.v13 Runtime Version ARN: arn:aws:lambda:ap-
southeast-2::runtime:699f346a05dae24c58c45790bc4089f252bf17dae3997e79b17d939a288aa1ec
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2025-09-20T14:15:28.229000
START RequestId: bed25b38-d012-42e7-ba28-f272535fb80e Version: $LATEST
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2025-09-20T14:15:29.259000
2025-09-20T14:15:29.201Z bed25b38-d012-42e7-ba28-f272535fb80e info
{"_aws":{"Timestamp":1676902528962,"CloudWatchMetrics":[{"Namespace":"sam-
app-logging","Metrics":[{"Name":"ColdStart","Unit":"Count"}],"Dimensions":
[["FunctionName"],["Service"]]}]},"FunctionName":"sam-app-HelloWorldFunction-
haKIoVeose2p","Service":"PowertoolsHelloWorld","ColdStart":1}
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2025-09-20T14:15:30.479000
2025-09-20T14:15:30.479Z bed25b38-d012-42e7-ba28-f272535fb80e info
{"ColdStart":true,"XrayTraceId":"1-63f3807f-5dbcb9910c96f50742707542","CorrelationId":"d3d
a549-4d67b2fdc015","FunctionName":"sam-app-HelloWorldFunction-
haKIoVeose2p","FunctionVersion":"$LATEST","FunctionMemorySize":256,"FunctionArn":"arn:aws:lambda:ap-
southeast-2:123456789012:function:sam-app-HelloWorldFunction-
haKIoVeose2p","FunctionRequestId":"bed25b38-d012-42e7-ba28-
f272535fb80e","Timestamp":"2025-09-20T14:15:30.4602970Z","Level":"Information","Service":"PowertoolsHelloWorld API - HTTP 200"}
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2025-09-20T14:15:30.599000
2025-09-20T14:15:30.599Z bed25b38-d012-42e7-ba28-f272535fb80e info
{"_aws":{"Timestamp":1676902528922,"CloudWatchMetrics":[{"Namespace":"sam-
app-logging","Metrics":[{"Name":"ApiRequestCount","Unit":"Count"}],"Dimensions":
[["Service"]]}]},"Service":"PowertoolsHelloWorld","ApiRequestCount":1}
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2025-09-20T14:15:30.680000 END
RequestId: bed25b38-d012-42e7-ba28-f272535fb80e
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2025-09-20T14:15:30.680000
REPORT RequestId: bed25b38-d012-42e7-ba28-f272535fb80e Duration: 2450.99 ms
Billed Duration: 2692 ms Memory Size: 256 MB Max Memory Used: 74 MB Init
Duration: 240.05 ms
XRAY TraceId: 1-63f3807f-5dbcb9910c96f50742707542 SegmentId: 16b362cd5f52cba0
```

8. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

## Gestione della conservazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, elimina il gruppo di log o configura un periodo di conservazione dopo il quale i log CloudWatch vengono eliminati automaticamente. Per impostare la conservazione dei log, aggiungi quanto segue al tuo AWS SAM modello:

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

## Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor di codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, trovi l'output del registro nella sezione Dettagli.

## Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Aprire la pagina dei gruppi di [log](#) sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (/aws/lambda/***your-function-name***).



### 3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'[istanza della funzione](#). Un flusso di log viene visualizzato quando si aggiorna la funzione Lambda e quando vengono create istanze aggiuntive per gestire chiamate simultanee. Per trovare i log per una chiamata specifica, ti consigliamo di strumentare la tua funzione con AWS X-Ray. X-Ray registra i dettagli sulla richiesta e sul flusso di log nella traccia.

## Visualizzazione dei log utilizzando AWS Command Line Interface (AWS CLI)

AWS CLI È uno strumento open source che puoi utilizzare per interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della [AWS CLI versione 2](#).

È possibile utilizzare [AWS CLI](#) per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

## Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "LogResult":
    "U1RBUIQgUmVxdWVzdElkOiA4N2QwNDRIOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lva... ",
  "ExecutedVersion": "$LATEST"
}
```

### Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'base64 utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
```

```
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --decode
```

L'opzione `--cli-binary-format` è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
      Duration: 80 ms      Memory Size: 128 MB      Max Memory Used: 73 MB
```

L'utilità `base64` è disponibile su Linux, macOS e [Ubuntu su Windows](#). Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

Example Script `get-logs.sh`

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'opzione `--cli-binary-format` è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"/'/g' out
sleep 15
```

```
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

### Example(solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il comando seguente per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

### Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\n$LATEST\n",
\r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
```

```

        "ingestionTime": 1559763018353
    },
    {
        "timestamp": 1559763003218,
        "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
        "ingestionTime": 1559763018353
    },
    {
        "timestamp": 1559763003218,
        "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75 MB\t\n",
        "ingestionTime": 1559763018353
    }
],
"nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}

```

## Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o [configurare un periodo di conservazione](#) trascorso il quale i log vengono eliminati automaticamente.

## Strumentazione del codice C# in AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzarlo X-Ray per tracciare una richiesta mentre attraversa le risorse dell'applicazione, che possono includere funzioni Lambda e altri servizi. AWS

Per inviare dati di tracciamento a X-Ray, puoi utilizzare una delle tre librerie SDK:

- [AWS Distro for OpenTelemetry \(ADOT\)](#): una distribuzione sicura, pronta per la produzione e AWS supportata dell'SDK (OtEL). OpenTelemetry
- [SDK AWS X-Ray per .NET](#)— Un SDK per la generazione e l'invio di dati di traccia a X-Ray
- [Powertools for AWS Lambda \(.NET\)](#): un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori.

Ciascuno degli SDK offre modi per inviare i dati di telemetria al servizio. X-Ray Puoi quindi utilizzarlo X-Ray per visualizzare, filtrare e ottenere informazioni dettagliate sulle metriche delle prestazioni dell'applicazione per identificare problemi e opportunità di ottimizzazione.

### Important

The X-Ray e Powertools for AWS Lambda SDK fanno parte di una soluzione di strumentazione strettamente integrata offerta da AWS. I livelli Lambda ADOT fanno parte di uno standard di settore per la strumentazione di tracciamento che in generale raccoglie più dati, ma potrebbero non essere adatti a tutti i casi d'uso. È possibile implementare il tracciamento end-to-end utilizzando entrambe le soluzioni. X-Ray Per saperne di più sulla scelta tra di esse, consulta [Scelta tra AWS Distro for Open Telemetry e SDK. X-Ray](#)

## Sections

- [Usare Powertools per AWS Lambda \(.NET\) e AWS SAM per tracciare](#)
- [Utilizzo dell' X-Ray SDK per strumentare le tue funzioni.NET](#)
- [Attivazione del tracciamento con la console Lambda](#)
- [Attivazione del tracciamento con l'API Lambda](#)
- [Attivazione del tracciamento con CloudFormation](#)
- [Interpretazione e tracciamento X-Ray](#)

## Usare Powertools per AWS Lambda (.NET) e AWS SAM per tracciare

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Hello World C# di esempio con moduli [Powertools for AWS Lambda \(.NET\)](#) integrati utilizzando AWS SAM. Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a CloudWatch X-Ray. La funzione restituisce un messaggio hello world.

### Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- .NET 8
- [AWS CLI versione 2](#)
- [AWS SAM CLI versione 1.75 o successiva](#). Se disponi di una versione precedente della AWS SAM CLI, vedi [Aggiornamento della CLI. AWS SAM](#)

### Distribuisci un esempio AWS SAM applicazione

1. Inizializza l'applicazione utilizzando il modello Hello World. TypeScript

```
sam init --app-template hello-world-powertools-dotnet --name sam-app --package-type Zip --runtime dotnet6 --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi Enter.

 Note

Perché HelloWorldFunction potrebbe non avere l'autorizzazione definita, va bene? , assicurati di entrarey.

## 5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query  
'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

## 6. Richiama l'endpoint dell'API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere le tracce per la funzione, esegui [sam traces](#).

```
sam traces
```

L'output della traccia ha il seguente aspetto:

```
New XRay Service Graph  
Start time: 2023-02-20 23:05:16+08:00  
End time: 2023-02-20 23:05:16+08:00  
Reference Id: 0 - AWS::Lambda - sam-app-HelloWorldFunction-pNjub7mEoew - Edges:  
[1]  
  Summary_statistics:  
    - total requests: 1  
    - ok count(2XX): 1  
    - error count(4XX): 0  
    - fault count(5XX): 0  
    - total response time: 2.814  
Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-pNjub7mEoew  
- Edges: []  
  Summary_statistics:  
    - total requests: 1  
    - ok count(2XX): 1
```

```

- error count(4XX): 0
- fault count(5XX): 0
- total response time: 2.429
Reference Id: 2 - (Root) AWS::ApiGateway::Stage - sam-app/Prod - Edges: [0]
Summary_statistics:
- total requests: 1
- ok count(2XX): 1
- error count(4XX): 0
- fault count(5XX): 0
- total response time: 2.839
Reference Id: 3 - client - sam-app/Prod - Edges: [2]
Summary_statistics:
- total requests: 0
- ok count(2XX): 0
- error count(4XX): 0
- fault count(5XX): 0
- total response time: 0

XRay Event [revision 3] at (2023-02-20T23:05:16.521000) with id
(1-63f38c2c-270200bf1d292a442c8e8a00) and duration (2.877s)
- 2.839s - sam-app/Prod [HTTP: 200]
- 2.836s - Lambda [HTTP: 200]
- 2.814s - sam-app-HelloWorldFunction-pNjujb7mEoew [HTTP: 200]
- 2.429s - sam-app-HelloWorldFunction-pNjujb7mEoew
- 0.230s - Initialization
- 2.389s - Invocation
- 0.600s - ## FunctionHandler
- 0.517s - Get Calling IP
- 0.039s - Overhead

```

8. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

X-Ray non traccia tutte le richieste alla tua applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento sia efficiente, fornendo comunque un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non puoi configurare la frequenza di X-Ray campionamento per le tue funzioni.



## Utilizzo dell' X-Ray SDK per strumentare le tue funzioni.NET

È possibile utilizzare il codice funzione per registrare i metadati e tracciare le chiamate a valle. Per registrare i dettagli delle chiamate effettuate dalla funzione ad altre risorse e servizi, utilizza SDK AWS X-Ray per .NET. Per ottenere l'SDK, aggiungere i pacchetti AWSXRayRecorder al file di progetto.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AWSProjectType>Lambda</AWSProjectType>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="2.1.0" />
    <PackageReference Include="Amazon.Lambda.SQSEvents" Version="2.1.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="2.1.0" />
    <PackageReference Include="AWSSDK.Core" Version="3.7.103.24" />
    <PackageReference Include="AWSSDK.Lambda" Version="3.7.104.3" />
    <PackageReference Include="AWSXRayRecorder.Core" Version="2.13.0" />
    <PackageReference Include="AWSXRayRecorder.Handlers.AwsSdk" Version="2.11.0" />
  </ItemGroup>
</Project>
```

Esistono diversi pacchetti Nuget che forniscono la strumentazione automatica per AWS SDK, Entity Framework e richieste HTTP. Per visualizzare il set completo di opzioni di configurazione, consulta [SDK AWS X-Ray per .NET](#) nella Guida per gli sviluppatori di AWS X-Ray .

Dopo aver aggiunto i pacchetti Nuget desiderati, configura la strumentazione automatica. Una best practice consiste nell'eseguire questa configurazione al di fuori della funzione di gestione della funzione. Questo sfrutta il riutilizzo dell'ambiente di esecuzione per migliorare le prestazioni della funzione. Nel seguente esempio di codice, il `RegisterXRayForAllServices` metodo viene chiamato nel costruttore della funzione per aggiungere strumentazione per tutte le chiamate SDK. AWS

```
[assembly:
  LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))

namespace GetProductHandler;

public class Function
```

```
{
    private readonly IDatabaseRepository _repo;

    public Function()
    {
        // Add auto instrumentation for all AWS SDK calls
        // It is important to call this method before initializing any SDK clients
        AWSSDKHandler.RegisterXRayForAllServices();
        this._repo = new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
request)
    {
        var id = request.PathParameters["id"];

        var databaseRecord = await this._repo.GetById(id);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = JsonSerializer.Serialize(databaseRecord)
        };
    }
}
```

## Attivazione del tracciamento con la console Lambda

Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. In Strumenti di monitoraggio aggiuntivi scegli Modifica.
5. In CloudWatch Application Signals e AWS X-Ray, scegli Abilita per le tracce del servizio Lambda.
6. Scegli Save (Salva).

## Attivazione del tracciamento con l'API Lambda

Configura il tracciamento sulla tua funzione Lambda con AWS CLI or AWS SDK, utilizza le seguenti operazioni API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

Il seguente AWS CLI comando di esempio abilita la traccia attiva su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

## Attivazione del tracciamento con CloudFormation

Per attivare la traccia su una `AWS::Lambda::Function` risorsa in un CloudFormation modello, utilizzate la proprietà `TracingConfig`

Example [function-inline.yml](#) — Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Per una `AWS::Serverless::Function` risorsa AWS Serverless Application Model (AWS SAM), usa la `Tracing` proprietà.

Example [template.yml](#) — Configurazione di tracciamento

```
Resources:
  function:
```

Type: [AWS::Serverless::Function](#)

Properties:

**Tracing: Active**

...

## Interpretazione e tracciamento X-Ray

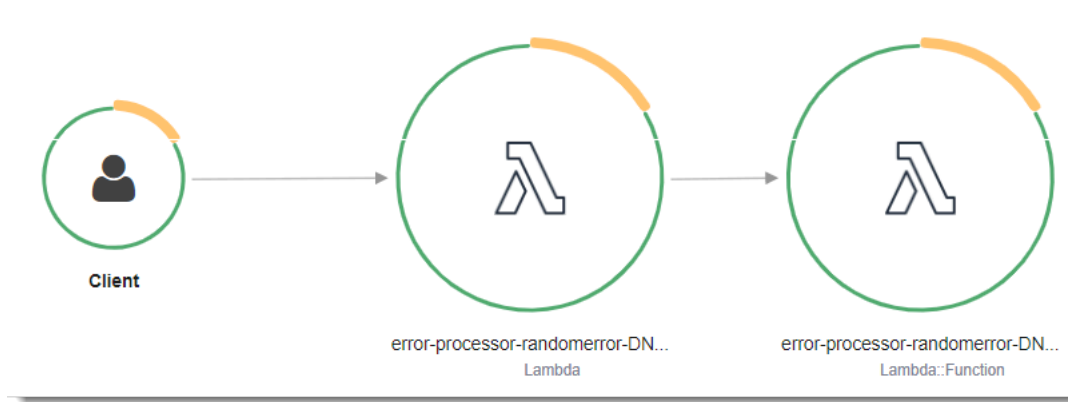
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al [ruolo di esecuzione](#) della funzione. Altrimenti, aggiungi la [AWSXRayDaemonWriteAccess](#) policy al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico del [X-Ray servizio](#) mostra le informazioni sull'applicazione e su tutti i suoi componenti. Il seguente esempio mostra un'applicazione con due funzioni. La funzione principale elabora gli eventi e talvolta restituisce errori. La seconda funzione in alto elabora gli errori che appaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon CloudWatch Logs.

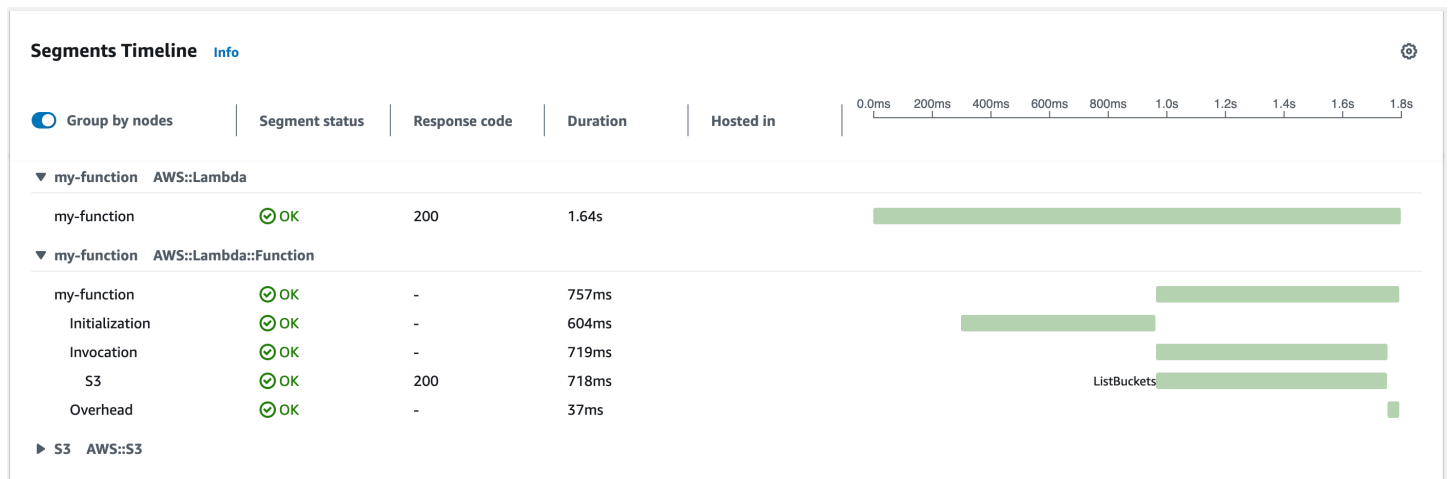


X-Ray non traccia tutte le richieste all'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento sia efficiente, fornendo comunque un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non puoi configurare la frequenza di X-Ray campionamento per le tue funzioni.

In X-Ray, una traccia registra le informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati `my-function`, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.

### Note

AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti.

Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regioni, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia nel nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- Inizializzazione – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il [codice di inizializzazione](#). Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.
- Chiamata: rappresenta il tempo impiegato per eseguire il codice del gestore.
- Overhead: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla struttura dei segmenti di funzioni vecchio e nuovo stile, consulta [the section called “Comprensione delle tracce X-Ray”](#).

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta [SDK AWS X-Ray per .NET](#) nella Guida per gli sviluppatori di AWS X-Ray .

#### Prezzi

Puoi utilizzare il X-Ray tracciamento gratuitamente ogni mese fino a un determinato limite come parte del piano gratuito. AWS Oltre tale soglia, vengono X-Ray addebitati i costi per l'archiviazione e il recupero delle tracce. Per ulteriori informazioni, consultare [Prezzi di AWS X-Ray](#).

## AWS Lambda test delle funzioni in C#

### Note

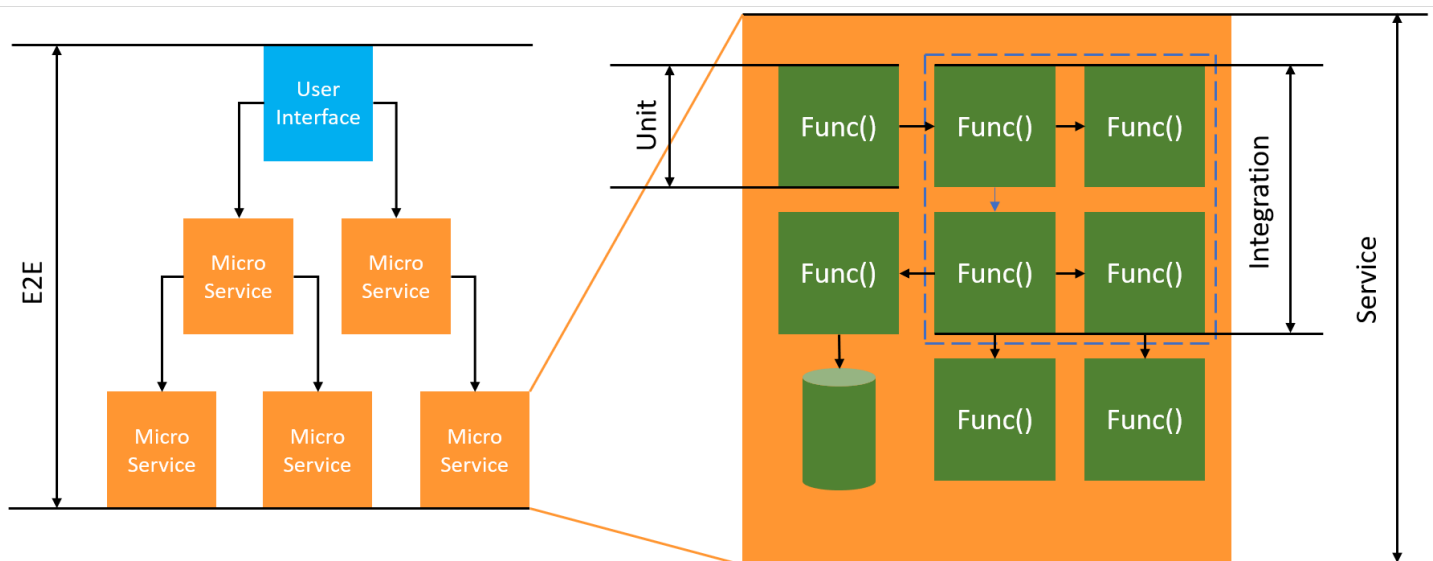
Consulta il capitolo sul [Test delle funzioni](#) per un'introduzione completa alle tecniche e alle best practice per testare soluzioni serverless.

Il test delle funzioni serverless utilizza tipi e tecniche di test tradizionali, ma è necessario considerare anche il test delle applicazioni serverless nel loro insieme. Cloud-based i test forniscono la misura più accurata della qualità sia delle funzioni che delle applicazioni serverless.

Un'architettura applicativa serverless include servizi gestiti che forniscono funzionalità applicative critiche tramite chiamate API. Pertanto, è fondamentale che il ciclo di sviluppo preveda test automatici in grado di verificare il corretto funzionamento dell'interazione tra le funzioni e i servizi.

Se non crei test basati sul cloud, potresti riscontrare problemi dovuti alle differenze tra l'ambiente locale e quello implementato. Il processo di integrazione continua dovrebbe eseguire test su un insieme di risorse con provisioning nel cloud prima di promuovere il codice nell'ambiente di implementazione successivo, come quello di controllo qualità (QA), staging o produzione.

Continua a leggere questa breve guida per scoprire le strategie di test per le applicazioni serverless oppure visita il [repository Serverless Test Samples](#) per approfondire esempi pratici, specifici per il linguaggio e il runtime selezionati.



Per i test serverless, è comunque necessario scrivere test unitari, di integrazione e end-to-end.

- **Test unitari:** vengono eseguiti su un blocco di codice isolato. Ad esempio, possono essere utilizzati per verificare la logica aziendale per calcolare le spese di spedizione in base a un articolo e a una destinazione specifici.
- **Test di integrazione:** coinvolgono due o più componenti o servizi che interagiscono, in genere in un ambiente cloud. Ad esempio, possono essere utilizzati per verificare che una funzione elabori gli eventi di una coda.
- **End-to-end test:** test che verificano il comportamento in un'intera applicazione. Ad esempio, possono essere utilizzati per verificare che l'infrastruttura sia configurata correttamente e che gli eventi fluiscano tra i servizi come previsto per registrare l'ordine di un cliente.

## Test delle applicazioni serverless

In genere si utilizzano una combinazione di approcci per testare il codice dell'applicazione serverless, tra cui test nel cloud, test con simulazioni e, occasionalmente, test con emulatori.

### Test nel cloud

I test nel cloud sono utili per tutte le fasi di test, inclusi i test unitari, i test di integrazione e i test end-to-end. Esegui test su codice implementato nel cloud e interagisci con servizi basati su cloud. Questo approccio fornisce la misura più accurata della qualità del codice.

Un modo conveniente per eseguire il debug di una funzione Lambda nel cloud è con un evento di test nella console. Un evento di test è un input JSON per la funzione. Se la funzione non richiede input, l'evento può essere un documento JSON ( `{}` ) vuoto. La console fornisce eventi di esempio per una varietà di integrazioni di servizi. Dopo aver creato un evento nella console, puoi condividerlo con il tuo team per rendere i test più semplici e coerenti.

#### Note

[Testare una funzione nella console](#) è un modo rapido per iniziare, ma l'automazione dei cicli di test garantisce la qualità delle applicazioni e la velocità di sviluppo.

## Strumenti di test

Per accelerare il ciclo di sviluppo, esistono diversi strumenti e tecniche che è possibile utilizzare durante il test delle funzioni. Ad esempio, [AWS SAM Accelerate](#) e [AWS CDK CDK watch mode](#) riducono entrambi il tempo necessario per aggiornare gli ambienti cloud.



Il modo in cui definisci il codice della funzione Lambda semplifica l'aggiunta di test unitari. Per inizializzare la classe, Lambda richiede un costruttore pubblico senza parametri. L'introduzione di un secondo costruttore interno consente di controllare le dipendenze utilizzate dall'applicazione.

```
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))

namespace GetProductHandler;

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function(): this(null)
    {
    }

    internal Function(IDatabaseRepository repo)
    {
        this._repo = repo ?? new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
request)
    {
        var id = request.PathParameters["id"];

        var databaseRecord = await this._repo.GetById(id);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = JsonSerializer.Serialize(databaseRecord)
        };
    }
}
```

Per scrivere un test per questa funzione, puoi inizializzare una nuova istanza della classe `Function` e passare un'implementazione simulata di `IDatabaseRepository`. Gli esempi seguenti utilizzano `XUnit`, `Moq` e `FluentAssertions` per scrivere un semplice test che assicuri che `FunctionHandler` restituisca un codice di stato 200.

```
using Xunit;
using Moq;
using FluentAssertions;

public class FunctionTests
{
    [Fact]
    public async Task TestLambdaHandler_WhenInputIsValid_ShouldReturn200StatusCode()
    {
        // Arrange
        var mockDatabaseRepository = new Mock<IDatabaseRepository>();

        var functionUnderTest = new Function(mockDatabaseRepository.Object);

        // Act
        var response = await functionUnderTest.FunctionHandler(new
        APIGatewayProxyRequest());

        // Assert
        response.StatusCode.Should().Be(200);
    }
}
```

Per esempi più dettagliati, inclusi esempi di test asincroni, consultate il repository .NET testing samples su <https://github.com/aws-samples/serverless-test-samples/tree/main/dotnet-test-samples> GitHub

## Creazione di funzioni Lambda con PowerShell

Le sezioni seguenti spiegano come si applicano i modelli di programmazione e i concetti fondamentali comuni quando si crea il codice di una funzione Lambda in PowerShell.

Per informazioni sull'oggetto di PowerShell contesto, vedere [the section called “Contesto”](#).

Lambda fornisce le seguenti applicazioni di esempio per PowerShell:

- [blank-powershell](#): una PowerShell funzione che mostra l'uso della registrazione, delle variabili di ambiente e dell'SDK. AWS

Prima di iniziare, devi prima configurare un ambiente di sviluppo. PowerShell Per istruzioni su come eseguire questa operazione, consultare [Configurazione di un ambiente di sviluppo PowerShell](#).

Per informazioni su come utilizzare il AWSLambdaPSCore modulo per scaricare PowerShell progetti di esempio dai modelli, creare pacchetti PowerShell di distribuzione e distribuire PowerShell funzioni nel AWS cloud, consulta. [Distribuzione delle funzioni Lambda di PowerShell con gli archivi di file .zip](#)

Lambda fornisce i runtime seguenti per i linguaggi .NET:

| Nome                      | Identificatore | Sistema operativo | Data di ritiro   | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|---------------------------|----------------|-------------------|------------------|---------------------------|-------------------------------|
| .NET 10                   | dotnet10       | Amazon Linux 2023 | 14 novembre 2028 | 14 dicembre 2028          | 15 gennaio 2029               |
| .NET 9 (solo contenitore) | dotnet9        | Amazon Linux 2023 | 10 novembre 2026 | Non programmato           | Non programmato               |
| .NET 8                    | dotnet8        | Amazon Linux 2023 | 10 novembre 2026 | 1 febbraio 2027           | 3 marzo 2027                  |

## Argomenti

- [Configurazione di un ambiente di sviluppo PowerShell](#)
- [Distribuzione delle funzioni Lambda di PowerShell con gli archivi di file .zip](#)
- [Definire l'handler della funzione Lambda in PowerShell](#)
- [Utilizzo dell'oggetto contestuale Lambda per recuperare le informazioni sulla funzione PowerShell](#)
- [Registrare e monitorare le funzioni Lambda con Powershell](#)

## Configurazione di un ambiente di sviluppo PowerShell

Lambda fornisce un insieme di strumenti e librerie per il runtime PowerShell. Per le istruzioni di installazione, consulta [Lambda Tools for Powershell](#) su GitHub.

Il modulo AWSLambdaPSCore include i seguenti nuovi cmdlet per aiutare a creare e pubblicare funzioni PowerShell Lambda.

- `Get-AWSPowerShellLambdaTemplate` – Restituisce un elenco di modelli di base.
- `New-AWSPowerShellLambda` – Crea uno script PowerShell iniziale basato su un modello.
- `Publish-AWSPowerShellLambda` – Pubblica un dato script PowerShell su Lambda.
- `New-AWSPowerShellLambdaPackage` – Crea un pacchetto di implementazione Lambda che può essere utilizzato in un sistema CI/CD per l'implementazione.

## Distribuzione delle funzioni Lambda di PowerShell con gli archivi di file .zip

Un pacchetto di distribuzione per il runtime PowerShell contiene lo script PowerShell, i moduli PowerShell necessari per lo script PowerShell e gli assembly necessari per ospitare PowerShell Core.

### Creazione di una funzione Lambda

Per iniziare a scrivere e invocare uno script PowerShell con , è possibile utilizzare il cmdlet `New-AWSPowerShell11Lambda` per creare un script di inizio in base a un modello. È possibile utilizzare il cmdlet `Publish-AWSPowerShell11Lambda` per distribuire lo script in Lambda. Quindi è possibile testare lo script utilizzando la riga di comando o la console Lambda.

Per creare un nuovo script PowerShell, caricarlo e testarlo, effettuare le seguenti operazioni:

1. Per visualizzare l'elenco dei modelli disponibili, esegui il comando seguente:

```
PS C:\> Get-AWSPowerShell11LambdaTemplate

Template          Description
-----
Basic             Bare bones script
CodeCommitTrigger Script to process AWS CodeCommit Triggers
...
```

2. Per creare uno script di esempio in base al modello Basic, esegui il comando seguente:

```
New-AWSPowerShell11Lambda -ScriptName MyFirstPSScript -Template Basic
```

Un nuovo file con nome `MyFirstPSScript.ps1` viene creato in una nuova sottodirectory della directory corrente. Il nome della directory viene determinato in base al parametro `-ScriptName`. Puoi utilizzare il parametro `-Directory` per scegliere una directory alternativa.

Puoi vedere che il nuovo file ha il seguente contenuto:

```
# PowerShell script file to run as a Lambda function
#
# When executing in Lambda the following variables are predefined.
# $LambdaInput - A PSObject that contains the Lambda function input data.
# $LambdaContext - An Amazon.Lambda.Core.ILambdaContext object that contains
information about the currently running Lambda environment.
```

```
#  
# The last item in the PowerShell pipeline is returned as the result of the Lambda  
# function.  
#  
# To include PowerShell modules with your Lambda function, like the  
# AWSPowerShell.NetCore module, add a "#Requires" statement  
# indicating the module and version.  
  
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}  
  
# Uncomment to send the input to CloudWatch Logs  
# Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
```

3. Per vedere come i messaggi di log dello script PowerShell vengono inviati ad Amazon CloudWatch Logs, rimuovi il commento dalla riga `Write-Host` dello script di esempio.

Per dimostrare come puoi restituire i dati dalle tue funzioni Lambda, aggiungi una nuova riga alla fine dello script con `$PSVersionTable`. In tal modo `$PSVersionTable` viene aggiunto alla pipeline PowerShell. Al completamento dello script di PowerShell, l'ultimo oggetto nella pipeline PowerShell sono i dati restituiti per la funzione Lambda. `$PSVersionTable` è una variabile globale PowerShell che fornisce inoltre informazioni sull'ambiente di esecuzione.

Dopo aver apportato le modifiche, le ultime due righe dello script di esempio sono come riportato di seguito:

```
Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)  
$PSVersionTable
```

4. Dopo la modifica del file `MyFirstPSScript.ps1`, modifica la directory sul percorso dello script. Quindi esegui il seguente comando per pubblicare lo script in Lambda:

```
Publish-AWSPowerShellLambda -ScriptPath .\MyFirstPSScript.ps1 -Name  
MyFirstPSScript -Region us-east-2
```

Nota che il parametro `-Name` specifica il nome della funzione Lambda che viene visualizzato nella console Lambda. Puoi utilizzare questa funzione per invocare manualmente lo script.

5. Invoca la funzione utilizzando il comando AWS Command Line Interface (AWS CLI) `invoke`.

```
> aws lambda invoke --function-name MyFirstPSScript out
```



## Definire l'handler della funzione Lambda in PowerShell

Quando una funzione Lambda viene invocata, il gestore Lambda richiama lo script PowerShell.

Quando lo script PowerShell viene richiamato, le seguenti variabili sono predefinite:

- ***\$LambdaInput*** – PSObject contenente l'input per il gestore. L'input può essere costituito da dati dell'evento, pubblicati da un'origine eventi, o un input personalizzato che fornisci, ad esempio una stringa o qualsiasi oggetto dati personalizzato.
- ***\$LambdaContext*** – Un oggetto Amazon.Lambda.Core.ILambdaContext che è possibile utilizzare per accedere alle informazioni sulla chiamata corrente, come il nome della funzione corrente, il limite di memoria, il runtime rimanente e la registrazione.

A titolo illustrativo, considera l'esempio di codice PowerShell riportato di seguito.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}  
Write-Host 'Function Name:' $LambdaContext.FunctionName
```

Questo script restituisce la proprietà `FunctionName` ottenuta dalla variabile `$LambdaContext`.

### Note

Sei tenuto a usare l'istruzione `#Requires` all'interno dei tuoi script PowerShell per indicare i moduli da cui dipendono gli script. Questa istruzione svolge due attività importanti. 1) Comunica ad altri sviluppatori quali moduli lo script utilizza e 2) identifica i moduli dipendenti richiesti dagli strumenti AWS PowerShell per creare un pacchetto con lo script, come parte della distribuzione. Per ulteriori informazioni sull'istruzione `#Requires` in PowerShell, consulta la pagina relativa alle [informazioni sui requisiti](#). Per ulteriori dettagli sui pacchetti di distribuzione PowerShell, consulta [Distribuzione delle funzioni Lambda di PowerShell con gli archivi di file .zip](#).

Quando la funzione Lambda PowerShell usa i cmdlet PowerShell AWS, assicurati di impostare un'istruzione `#Requires` che fa riferimento al modulo `AWSPowerShell.NetCore` che supporta PowerShell Core e non al modulo `AWSPowerShell` che supporta solo Windows PowerShell. Inoltre, assicurati di utilizzare la versione 3.3.270.0 o più recente di `AWSPowerShell.NetCore` che ottimizza il processo di importazione dei cmdlet. Se usi una versione precedente, sperimenterai partenze a freddo più lunghe. Per ulteriori informazioni, consulta [Tools for PowerShell AWS](#).



## Restituzione dei dati

Alcune chiamate Lambda hanno lo scopo di restituire i dati al loro chiamante. Ad esempio, se una chiamata era in risposta a una richiesta web proveniente da API Gateway, la funzione Lambda deve restituire la risposta. Per PowerShell Lambda, l'ultimo oggetto aggiunto alla pipeline PowerShell costituisce i dati restituiti dalla chiamata Lambda. Se l'oggetto è una stringa, i dati vengono restituiti così come sono. In caso contrario, l'oggetto viene convertito in formato JSON utilizzando il cmdlet `ConvertTo-Json`.

Ad esempio, considera la seguente istruzione PowerShell che aggiunge `$PSVersionTable` alla pipeline PowerShell:

```
$PSVersionTable
```

Al termine dello script di PowerShell, l'ultimo oggetto nella pipeline PowerShell sono i dati restituiti per la funzione Lambda. `$PSVersionTable` è una variabile globale PowerShell che fornisce inoltre informazioni sull'ambiente di esecuzione.

## Utilizzo dell'oggetto contestuale Lambda per recuperare le informazioni sulla funzione PowerShell

Quando Lambda esegue la tua funzione, passa le informazioni di contesto rendendo una variabile `$LambdaContext` disponibile al [gestore](#). Questa variabile fornisce i metodi e le proprietà con informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

### Proprietà del contesto

- `FunctionName`: il nome della funzione Lambda.
- `FunctionVersion`: la [versione](#) della funzione.
- `InvokedFunctionArn`: l'Amazon Resource Name (ARN) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `MemoryLimitInMB`: la quantità di memoria allocata per la funzione.
- `AwsRequestId`: l'identificatore della richiesta di invocazione.
- `LogGroupName`: il gruppo di log per la funzione.
- `LogStreamName`: il flusso di log per l'istanza della funzione.
- `RemainingTime`: il numero di millisecondi rimasti prima del timeout dell'esecuzione.
- `Identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
- `ClientContext`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.
- `Logger`: l'[oggetto logger](#) per la funzione.

Lo snippet di codice PowerShell mostra una funzione semplice dell'handler che stampa alcune informazioni relative al contesto.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host 'Function name:' $LambdaContext.FunctionName
Write-Host 'Remaining milliseconds:' $LambdaContext.RemainingTime.TotalMilliseconds
Write-Host 'Log group name:' $LambdaContext.LogGroupName
Write-Host 'Log stream name:' $LambdaContext.LogStreamName
```

## Registrare e monitorare le funzioni Lambda con Powershell

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon. CloudWatch La tua funzione Lambda include un gruppo di CloudWatch log Logs e un flusso di log per ogni istanza della tua funzione. L'ambiente del runtime Lambda invia i dettagli su ogni richiamo al flusso di log e inoltra i log e l'output del codice della funzione. Per ulteriori informazioni, consulta [Invio dei log delle funzioni Lambda ai CloudWatch log](#).

Questa pagina descrive come produrre un output di log dal codice della funzione Lambda e accedere ai log utilizzando la AWS Command Line Interface console Lambda o la console. CloudWatch

### Sections

- [Creazione di una funzione che restituisce i registri](#)
- [Visualizzazione dei log nella console Lambda](#)
- [Visualizzazione dei log nella console CloudWatch](#)
- [Visualizzazione dei log utilizzando AWS Command Line Interface \(AWS CLI\)](#)
- [Eliminazione dei log](#)

### Creazione di una funzione che restituisce i registri

Per generare i log dal codice della funzione, puoi utilizzare i cmdlet on [Microsoft.PowerShell.Utility](#) o qualsiasi modulo di registrazione che scrive su o. stdout stderr Nell'esempio seguente viene utilizzato Write-Host.

### Example [function/Handler.ps1](#) — Registrazione

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host `## Environment variables
Write-Host AWS_LAMBDA_FUNCTION_VERSION=$Env:AWS_LAMBDA_FUNCTION_VERSION
Write-Host AWS_LAMBDA_LOG_GROUP_NAME=$Env:AWS_LAMBDA_LOG_GROUP_NAME
Write-Host AWS_LAMBDA_LOG_STREAM_NAME=$Env:AWS_LAMBDA_LOG_STREAM_NAME
Write-Host AWS_EXECUTION_ENV=$Env:AWS_EXECUTION_ENV
Write-Host AWS_LAMBDA_FUNCTION_NAME=$Env:AWS_LAMBDA_FUNCTION_NAME
Write-Host PATH=$Env:PATH
Write-Host `## Event
Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 3)
```

## Example Formato dei log

```

START RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed Version: $LATEST
Importing module ./Modules/AWSPowerShell.NetCore/3.3.618.0/AWSPowerShell.NetCore.psd1
[Information] - ## Environment variables
[Information] - AWS_LAMBDA_FUNCTION_VERSION=$LATEST
[Information] - AWS_LAMBDA_LOG_GROUP_NAME=/aws/lambda/blank-powershell-
function-18CIXMPLHFAJJ
[Information] - AWS_LAMBDA_LOG_STREAM_NAME=2020/04/01/
[$LATEST]53c5xmpl52d64ed3a744724d9c201089
[Information] - AWS_EXECUTION_ENV=AWS_Lambda_dotnet6_powershell_1.0.0
[Information] - AWS_LAMBDA_FUNCTION_NAME=blank-powershell-function-18CIXMPLHFAJJ
[Information] - PATH=/var/lang/bin:/usr/local/bin:/usr/bin:/bin:/opt/bin
[Information] - ## Event
[Information] -
{
  "Records": [
    {
      "messageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
      "receiptHandle": "MessageReceiptHandle",
      "body": "Hello from SQS!",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1523232000000",
        "SenderId": "123456789012",
        "ApproximateFirstReceiveTimestamp": "1523232000001"
      },
      ...
    }
  ]
}
END RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed
REPORT RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed Duration: 3906.38 ms Billed
Duration: 9867 ms Memory Size: 512 MB Max Memory Used: 367 MB Init Duration: 5960.19
ms
XRAY TraceId: 1-5e843da6-733cxmple7d0c3c020510040 SegmentId: 3913xmpl20999446 Sampled:
true

```

Il runtime di .NET registra START, END e REPORT per ogni chiamata. La riga del report fornisce i seguenti dettagli.

### Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per l'invocazione.

- **Durata** – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- **Durata fatturata** – La quantità di tempo fatturata per la chiamata.
- **Dimensioni memoria** – La quantità di memoria allocata per la funzione.
- **Quantità max utilizzata** – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- **Durata Init** – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- **XRAY Traceld**: per le richieste tracciate, l'ID di traccia. [AWS X-Ray](#)
- **SegmentId**— Per le richieste tracciate, l'ID del X-Ray segmento.
- **Campionato** – Per le richieste tracciate, il risultato del campionamento.

## Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor di codice incorporato, i log sono contenuti nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, trovi l'output del registro nella sezione **Dettagli**.

## Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Aprire la pagina dei gruppi di [log](#) sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (`/aws/lambda/your-function-name`).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'[istanza della funzione](#). Un flusso di log viene visualizzato quando si aggiorna la funzione Lambda e quando vengono create istanze aggiuntive per gestire chiamate

simultanee. Per trovare i log per una chiamata specifica, ti consigliamo di strumentare la tua funzione con AWS X-Ray. X-Ray registra i dettagli sulla richiesta e sul flusso di log nella traccia.

## Visualizzazione dei log utilizzando AWS Command Line Interface (AWS CLI)

AWS CLI È uno strumento open source che puoi utilizzare per interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della [AWS CLI versione 2](#).

È possibile utilizzare [AWS CLI](#) per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

## Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "LogResult":
    "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lva... ",
  "ExecutedVersion": "$LATEST"
}
```

## Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'base64 utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'cli-binary-format opzione è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-`

base64-out. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0""ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed
Duration: 80 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

L'utilità base64 è disponibile su Linux, macOS e [Ubuntu su Windows](#). Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

### Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'opzione `cli-binary-format` è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

### Example(solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il comando seguente per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\n$LATEST\n",
\r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    },
    {
```



```

        "timestamp": 1559763003218,
        "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
MB\t\n",
        "ingestionTime": 1559763018353
    }
],
"nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}

```

## Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o [configurare un periodo di conservazione](#) trascorso il quale i log vengono eliminati automaticamente.

## Compilazione di funzioni Lambda con Rust

Poiché Rust viene compilato in codice nativo, non è necessario un runtime dedicato per eseguire il codice Rust su Lambda. Utilizza invece il client di runtime [Rust](#) per creare il tuo progetto localmente, quindi distribiscilo su Lambda utilizzando un [OS-only runtime](#). Quando usi un OS-only runtime, Lambda mantiene automaticamente aggiornato il sistema operativo con le ultime patch. Per informazioni su come lavorare con gli eventi HTTP, consulta [the section called “Eventi HTTP”](#). Per informazioni sull'uso dei livelli con Rust, consulta [the section called “Livelli”](#).

### Strumenti e librerie per Rust

- [AWS SDK per Rust](#): l'AWS SDK per Rust fornisce le API Rust per interagire con i servizi di infrastruttura di Amazon Web Services.
- [Client di runtime Rust per Lambda](#): il client di runtime Rust semplifica l'esecuzione delle funzioni Lambda scritte in Rust.
- [Cargo Lambda](#): si tratta di un'estensione open source di terze parti dello strumento da riga di comando Cargo che semplifica la creazione e l'implementazione di funzioni Rust Lambda.
- [HTTP Lambda](#): questa libreria fornisce un wrapper per lavorare con gli eventi HTTP.
- [Estensione Lambda](#): questa libreria fornisce supporto per scrivere estensioni Lambda con Rust.
- [AWS Lambda Eventi](#): questa libreria fornisce definizioni di tipo per le comuni integrazioni di sorgenti di eventi.

## Esempi di applicazioni Lambda per Rust

- [Funzione Lambda di base](#): una funzione Rust che mostra come elaborare gli eventi di base.
- [Funzione Lambda con gestione degli errori](#): una funzione Rust che mostra come gestire gli errori Rust personalizzati in Lambda.
- [Funzione Lambda con risorse condivise](#): un progetto Rust che inizializza le risorse condivise prima della creazione della funzione Lambda.
- [Eventi HTTP Lambda](#): una funzione Rust che gestisce gli eventi HTTP.
- [Eventi HTTP Lambda con intestazioni CORS](#): una funzione Rust che utilizza Tower per inserire le intestazioni CORS.
- [REST API Lambda](#): una REST API che utilizza Axum e Diesel per connettersi a un database PostgreSQL.
- [Demo Serverless Rust](#): un progetto Rust che mostra l'uso delle librerie Rust, della registrazione, delle variabili di ambiente e dell'SDK di Lambda. AWS
- [Estensione Lambda di base](#): un'estensione Rust che mostra come elaborare gli eventi di estensione di base.
- [Estensione Amazon Data Firehose per i log Lambda](#): un'estensione Rust che mostra come inviare log Lambda a Firehose.

## Argomenti

- [Definire i gestori di funzioni Lambda in Rust](#)
- [Utilizzo dell'oggetto contestuale Lambda per recuperare le informazioni sulla funzione Rust](#)
- [Elaborazione di eventi HTTP con Rust](#)
- [Implementazione di funzioni Lambda per Go con gli archivi di file .zip](#)
- [Utilizzo dei livelli per le funzioni Lambda in Go](#)
- [Registrare e monitorare le funzioni Lambda con Rust](#)

## Definire i gestori di funzioni Lambda in Rust

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come lavorare con i gestori di funzioni Lambda in Rust, tra cui l'inizializzazione del progetto, le convenzioni di denominazione e le best practice. Questa pagina include anche un esempio di funzione Rust Lambda che acquisisce informazioni su un ordine, produce una ricevuta in un file di testo e lo inserisce in un bucket Amazon Simple Storage Service (S3). Per informazioni su come distribuire una funzione dopo averla scritta, consulta [the section called “Implementazione di archivi di file .zip”](#).

### Argomenti

- [Configurazione del progetto Rust handler](#)
- [Esempio di codice della funzione Rust Lambda](#)
- [Definizioni di classe valide per i gestori Rust](#)
- [Convenzioni di denominazione dei gestori](#)
- [Definizione e accesso all'oggetto evento di input](#)
- [Accesso e utilizzo dell'oggetto contestuale Lambda](#)
- [Utilizzo di AWS SDK per Rust nel tuo gestore](#)
- [Accesso alle variabili d'ambiente](#)
- [Utilizzo dello stato condiviso](#)
- [Best practice di codice per funzioni Lambda in Rust](#)

### Configurazione del progetto Rust handler

Quando si lavora con le funzioni Lambda in Rust, il processo prevede la scrittura del codice, la compilazione e la distribuzione degli artefatti compilati su Lambda. Il modo più semplice per configurare un progetto di gestore Lambda in Rust è usare Runtime for Rust. [AWS Lambda](#) Nonostante il nome, AWS Lambda Runtime for Rust non è un runtime gestito nello stesso senso in cui lo è in Lambda for Python, Java o. Node.js. Invece, AWS Lambda Runtime for Rust è un crate (`lambda_runtime`) che supporta la scrittura di funzioni Lambda in Rust e l'interfacciamento con l'ambiente di esecuzione. AWS Lambda

Usa il seguente comando per installare [Cargo Lambda](#), un'estensione open source di terze parti dello strumento da riga di comando Cargo che semplifica la creazione e l'implementazione di funzioni Rust Lambda:

```
cargo install cargo-lambda
```

Una volta completata l'installazione `cargo-lambda`, usa il seguente comando per inizializzare un nuovo progetto di gestore di funzioni Rust Lambda:

```
cargo lambda new example-rust
```

Quando esegui questo comando, l'interfaccia a riga di comando (CLI) ti pone un paio di domande sulla tua funzione Lambda:

- Funzione HTTP: se intendi richiamare la tua funzione tramite [API Gateway](#) o l'URL di una [funzione](#), rispondi Sì. Altrimenti, rispondi No. Nel codice di esempio in questa pagina, invochiamo la nostra funzione con un evento JSON personalizzato, quindi rispondiamo No.
- Tipo di evento: se intendi utilizzare una forma di evento predefinita per richiamare la funzione, seleziona il tipo di evento previsto corretto. Altrimenti, lascia vuota questa opzione. Nel codice di esempio in questa pagina, invochiamo la nostra funzione con un evento JSON personalizzato, quindi lasciamo vuota questa opzione.

Dopo che il comando è stato eseguito correttamente, accedi alla directory principale del tuo progetto:

```
cd example-rust
```

Questo comando genera un file `generic_handler.rs` e un file `main.rs` nella directory `src`. `generic_handler.rs` Può essere utilizzato per personalizzare un gestore di eventi generico. Il `main.rs` file contiene la logica principale dell'applicazione. Il `Cargo.toml` file contiene i metadati sul pacchetto ed elenca le sue dipendenze esterne.

Esempio di codice della funzione Rust Lambda

Il seguente esempio di codice della funzione Rust Lambda acquisisce informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3.

Example **funzione Lambda** `main.rs`

```
use aws_sdk_s3::{Client, primitives::ByteStream};
```

```

use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use serde::{Deserialize, Serialize};
use serde_json::Value;
use std::env;

#[derive(Deserialize, Serialize)]
struct Order {
    order_id: String,
    amount: f64,
    item: String,
}

async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error> {
    let payload = event.payload;

    // Deserialize the incoming event into Order struct
    let order: Order = serde_json::from_value(payload)?;

    let bucket_name = env::var("RECEIPT_BUCKET")
        .map_err(|_| "RECEIPT_BUCKET environment variable is not set")?;

    let receipt_content = format!(
        "OrderID: {}\nAmount: {:.2}\nItem: {}",
        order.order_id, order.amount, order.item
    );
    let key = format!("receipts/{}.txt", order.order_id);

    let config =
aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
    let s3_client = Client::new(&config);

    upload_receipt_to_s3(&s3_client, &bucket_name, &key, &receipt_content).await?;

    Ok("Success".to_string())
}

async fn upload_receipt_to_s3(
    client: &Client,
    bucket_name: &str,
    key: &str,
    content: &str,
) -> Result<(), Error> {
    client
        .put_object()

```

```

        .bucket(bucket_name)
        .key(key)
        .body(ByteStream::from(content.as_bytes().to_vec())) // Fixed conversion
        .content_type("text/plain")
        .send()
        .await?;

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(function_handler)).await
}

```

Questo file `main.rs` contiene le sezioni seguenti:

- `usedichiarazioni`: usate per importare le casse e i metodi Rust richiesti dalla tua funzione Lambda.
- `#[derive(Deserialize, Serialize)]`: definisce la forma dell'evento di input previsto in questa struttura Rust.
- `async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error>`: questo è il metodo dell'handler principale, che contiene la logica principale dell'applicazione.
- `async fn upload_receipt_to_s3 (...)`: Questo è un metodo di supporto a cui fa riferimento il metodo principale. `function_handler`
- `#[tokio::main]`: Questa è una macro che segna il punto di ingresso di un programma Rust. Imposta anche un runtime [Tokio](#), che consente al `main()` metodo di utilizzare `async/await` essere eseguito in modo asincrono.
- `async fn main() -> Result<(), Error>`: La `main()` funzione è il punto di ingresso del codice. Al suo interno, specifichiamo `function_handler` come metodo di gestione principale.

File di esempio `Cargo.toml`

Il `Cargo.toml` file seguente accompagna questa funzione.

```

[package]
name = "example-rust"
version = "0.1.0"
edition = "2024"

```

```
[dependencies]
aws-config = "1.5.18"
aws-sdk-s3 = "1.78.0"
lambda_runtime = "0.13.0"
serde = { version = "1", features = ["derive"] }
serde_json = "1"
tokio = { version = "1", features = ["full"] }
```

Affinché questa funzione funzioni correttamente, il suo ruolo di [esecuzione](#) deve consentire l'`s3:PutObject` azione. Inoltre, assicuratevi di definire la variabile di ambiente `RECEIPT_BUCKET`. Dopo una chiamata riuscita, il bucket Amazon S3 dovrebbe contenere un file di ricevuta.

Definizioni di classe valide per i gestori Rust

Nella maggior parte dei casi, le firme dei gestori Lambda definite in Rust hanno il seguente formato:

```
async fn function_handler(event: LambdaEvent<T>) -> Result<U, Error>
```

Per questo gestore:

- Il nome di questo gestore è. `function_handler`
- L'input singolare per il gestore è `event` ed è di tipo. `LambdaEvent<T>`
  - `LambdaEvent` è un involucro che proviene dalla cassa. `lambda_runtime` L'uso di questo wrapper consente di accedere all'oggetto `context`, che include Lambda-specific metadati come l'ID della richiesta dell'invocazione.
  - Tè il tipo di evento deserializzato. Ad esempio, questo può essere `serde_json::Value`, il che consente al gestore di accettare qualsiasi input JSON generico. In alternativa, può essere un tipo come `ApiGatewayProxyRequest` se la funzione prevedesse un tipo di input specifico e predefinito.
- Il tipo restituito dal gestore è. `Result<U, Error>`
  - Uè il tipo di output deserializzato. Udeve implementare il `serde::Serialize` tratto in modo che Lambda possa convertire il valore restituito in JSON. Ad esempio, `U` può essere un tipo semplice come `String` o `serde_json::Value`, o una struttura personalizzata purché sia implementata. `Serialize` Quando il codice raggiunge un'istruzione `Ok (U)`, ciò indica che l'esecuzione è riuscita e la funzione restituisce un valore di tipo `U`.
  - Quando il codice rileva un errore (ovvero `Err (Error)`), la funzione registra l'errore in Amazon CloudWatch e restituisce una risposta di errore di tipo. `Error`

Nel nostro esempio, la firma del gestore ha il seguente aspetto:

```
async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error>
```

Altre firme del gestore valide possono contenere quanto segue:

- Omissione del `LambdaEvent` wrapper: se si omette `LambdaEvent`, si perde l'accesso all'oggetto di contesto `Lambda` all'interno della funzione. Di seguito è riportato un esempio di questo tipo di firma:

```
async fn handler(event: serde_json::Value) -> Result<String, Error>
```

- Utilizzo del tipo di unità come input: per Rust, è possibile utilizzare il tipo di unità per rappresentare un input vuoto. Questo è comunemente usato per funzioni con invocazioni periodiche e pianificate. Di seguito è riportato un esempio di questo tipo di firma:

```
async fn handler(_: ()) -> Result<Value, Error>
```

## Convenzioni di denominazione dei gestori

I gestori Lambda in Rust non hanno rigide restrizioni di denominazione. Sebbene sia possibile utilizzare qualsiasi nome per il gestore, i nomi delle funzioni in Rust sono generalmente in `snake_case`.

Per applicazioni più piccole, come in questo esempio, puoi utilizzare un singolo `main.rs` file per contenere tutto il codice. Per progetti di grandi dimensioni, `main.rs` dovrebbe contenere il punto di ingresso alla funzione, ma è possibile aggiungere file aggiuntivi per separare il codice in moduli logici. Ad esempio, potresti avere la seguente struttura di file:

```
/example-rust
### src/
#   ### main.rs           # Entry point
#   ### handler.rs       # Contains main handler
#   ### services.rs      # [Optional] Back-end service calls
#   ### models.rs        # [Optional] Data models
### Cargo.toml
```

## Definizione e accesso all'oggetto evento di input

JSON è il formato di input più comune e standard per le funzioni Lambda. In questo esempio, la funzione prevede un input simile a quanto segue:



```
{
  "order_id": "12345",
  "amount": 199.99,
  "item": "Wireless Headphones"
}
```

In Rust, puoi definire la forma dell'evento di input previsto in una struttura. In questo esempio, definiamo la seguente struttura per rappresentare una `Order`:

```
#[derive(Deserialize, Serialize)]
struct Order {
    order_id: String,
    amount: f64,
    item: String,
}
```

Questa struttura corrisponde alla forma di input prevista. In questo esempio, la `#[derive(Deserialize, Serialize)]` macro genera automaticamente codice per la serializzazione e la deserializzazione. Ciò significa che possiamo deserializzare il tipo JSON di input generico nella nostra struttura utilizzando il metodo `serde_json::from_value()`. Ciò è illustrato nelle prime righe dell'handler:

```
async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error> {
    let payload = event.payload;

    // Deserialize the incoming event into Order struct
    let order: Order = serde_json::from_value(payload)?;
    ...
}
```

È quindi possibile accedere ai campi dell'oggetto. Ad esempio, `order.order_id` recupera il valore di `order_id` dall'input originale.

### Pre-defined tipi di eventi di input

Nella `aws_lambda_events` cassa sono disponibili molti tipi di eventi di input predefiniti. Ad esempio, se intendi richiamare la tua funzione con API Gateway, inclusa la seguente importazione:

```
use aws_lambda_events::event::apigw::ApiGatewayProxyRequest;
```

Quindi, assicurati che il tuo gestore principale utilizzi la seguente firma:

```
async fn handler(event: LambdaEvent<ApiGatewayProxyRequest>) -> Result<String, Error> {  
    let body = event.payload.body.unwrap_or_default();  
    ...  
}
```

Fai riferimento alla cassa [aws\\_lambda\\_events](#) per ulteriori informazioni su altri tipi di eventi di input predefiniti.

### Accesso e utilizzo dell'oggetto contestuale Lambda

L'[oggetto contesto](#): contiene informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. In Rust, il wrapper include l'oggetto context. LambdaEvent Ad esempio, puoi utilizzare l'oggetto context per recuperare l'ID della richiesta dell'invocazione corrente con il codice seguente:

```
async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error> {  
    let request_id = event.context.request_id;  
    ...  
}
```

Per ulteriori informazioni sulla copia di oggetti, consulta la sezione [the section called “Contesto”](#).

### Utilizzo di AWS SDK per Rust nel tuo gestore

Spesso, utilizzerai le funzioni Lambda per interagire o apportare aggiornamenti ad altre AWS risorse. Il modo più semplice per interfacciarsi con queste risorse è usare la [AWS SDK per Rust](#).

Per aggiungere dipendenze SDK alla tua funzione, aggiungile nel tuo file. Cargo.toml Ti consigliamo di aggiungere solo le librerie necessarie per la tua funzione. Nel codice di esempio precedente, abbiamo usato la `aws_sdk_s3::Client`. Nel Cargo.toml file, puoi aggiungere questa dipendenza aggiungendo la riga seguente nella `[dependencies]` sezione:

```
aws-sdk-s3 = "1.78.0"
```

#### Note

Questa potrebbe non essere la versione più recente. Scegli la versione appropriata per la tua applicazione.

Quindi, importa le dipendenze direttamente nel tuo codice:

```
use aws_sdk_s3::{Client, primitives::ByteStream};
```

Il codice di esempio inizializza quindi un client Amazon S3 come segue:

```
let config = aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let s3_client = Client::new(&config);
```

Dopo aver inizializzato il client SDK, puoi utilizzarlo per interagire con altri servizi. AWS Il codice di esempio richiama l'API PutObject Amazon S3 nella funzione upload\_receipt\_to\_s3 helper.

### Accesso alle variabili d'ambiente

Nel codice dell'handler, puoi fare riferimento a qualsiasi [variabile di ambiente](#) utilizzando il metodo `env::var`. In questo esempio, facciamo riferimento alla variabile di RECEIPT\_BUCKET ambiente definita utilizzando la seguente riga di codice:

```
let bucket_name = env::var("RECEIPT_BUCKET")
    .map_err(|_| "RECEIPT_BUCKET environment variable is not set")?;
```

### Utilizzo dello stato condiviso

È possibile dichiarare delle variabili condivise indipendenti dal codice del gestore della funzione Lambda. Queste variabili possono aiutarti a caricare le informazioni sullo stato durante i [Fase di init](#), prima che la funzione riceva eventi. Ad esempio, puoi modificare il codice in questa pagina per utilizzare lo stato condiviso durante l'inizializzazione del client Amazon S3 aggiornando la funzione e la firma del main gestore:

```
async fn function_handler(client: &Client, event: LambdaEvent<Value>) -> Result<String,
Error> {
    ...
    upload_receipt_to_s3(client, &bucket_name, &key, &receipt_content).await?;
    ...
}

...

#[tokio::main]
```

```
async fn main() -> Result<(), Error> {
    let shared_config = aws_config::from_env().load().await;
    let client = Client::new(&shared_config);
    let shared_client = &client;
    lambda_runtime::run(service_fn(move |event: LambdaEvent<Request>| async move {
        handler(&shared_client, event).await
    })))
    .await
```

## Best practice di codice per funzioni Lambda in Rust

Segui le linee guida riportate nell'elenco seguente per utilizzare le best practice di codifica durante la creazione delle funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. Ciò semplifica la creazione di una funzione più testabile a livello di unità.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del [contesto di esecuzione](#).
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. Ciò riduce il tempo necessario per scaricare e decomprimere il pacchetto di distribuzione prima dell'invocazione.

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione.

Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedi [Riutilizzo delle connessioni con in. Keep-Alive Node.js](#)

Utilizzare [le variabili di ambiente](#) per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su 0 per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionali alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha una dipendenza su queste API non pubbliche. Consulta il [riferimento all'API](#) per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta [Come posso rendere idempotente la mia funzione Lambda?](#).

## Utilizzo dell'oggetto contestuale Lambda per recuperare le informazioni sulla funzione Rust

Quando Lambda esegue la funzione, aggiunge un oggetto di contesto a `LambdaEvent` quello ricevuto dal [gestore](#). Questo oggetto fornisce proprietà con informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

### Proprietà del contesto

- `request_id`: l'ID della AWS richiesta generato dal servizio Lambda.
- `deadline`: la scadenza di esecuzione per la chiamata corrente in millisecondi.
- `invoked_function_arn`: il nome della risorsa Amazon (ARN) della funzione Lambda da richiamare.
- `xray_trace_id`: L'ID di AWS X-Ray traccia per la chiamata corrente.
- `client_content`: l'oggetto contestuale del client inviato dall'SDK AWS mobile. Questo campo è vuoto a meno che la funzione non venga richiamata utilizzando un SDK AWS mobile.
- `identity`: l'identità Amazon Cognito che ha richiamato la funzione. Questo campo è vuoto a meno che la richiesta di chiamata a Lambda non sia APIs stata effettuata utilizzando AWS credenziali emesse dai pool di identità di Amazon Cognito.
- `env_config`: la configurazione della funzione Lambda in base alle variabili di ambiente locali. Questa proprietà include informazioni come il nome della funzione, l'allocazione della memoria, la versione e i flussi di log.

### Accesso alle informazioni relative al contesto di invocazione

Le funzioni Lambda hanno accesso ai metadata relativi al loro ambiente e alla richiesta di invocazione. L'oggetto `LambdaEvent` ricevuto dal gestore delle funzioni include i metadati `context`:

```
use lambda_runtime::{service_fn, LambdaEvent, Error};
use serde_json::{json, Value};

async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    let invoked_function_arn = event.context.invoked_function_arn;
    Ok(json!({ "message": format!("Hello, this is function {invoked_function_arn}!") }))
}

#[tokio::main]
```

```
async fn main() -> Result<(), Error> {  
    lambda_runtime::run(service_fn(handler)).await  
}
```

## Elaborazione di eventi HTTP con Rust

Le API di Gateway Amazon API, gli Application Load Balancer e gli [URL delle funzioni](#) possono inviare eventi HTTP a Lambda. Puoi utilizzare la cassa [aws\\_lambda\\_events](#) di crates.io per elaborare gli eventi da queste origini.

### Example- Gestione della richiesta proxy di Gateway API

Tenere presente quanto segue:

- `use aws_lambda_events::apigw::{ApiGatewayProxyRequest, ApiGatewayProxyResponse};` la cassa [aws\\_lambda\\_events](#) include molti eventi Lambda. Per ridurre i tempi di compilazione, utilizza i flag delle funzionalità per attivare gli eventi di cui hai bisogno. Esempio: `aws_lambda_events = { version = "0.8.3", default-features = false, features = ["apigw"] }`.
- `use http::HeaderMap;` questa importazione richiede l'aggiunta della cassa [http](#) alle dipendenze.

```
use aws_lambda_events::apigw::{ApiGatewayProxyRequest, ApiGatewayProxyResponse};
use http::HeaderMap;
use lambda_runtime::{service_fn, Error, LambdaEvent};

async fn handler(
    _event: LambdaEvent<ApiGatewayProxyRequest>,
) -> Result<ApiGatewayProxyResponse, Error> {
    let mut headers = HeaderMap::new();
    headers.insert("content-type", "text/html".parse().unwrap());
    let resp = ApiGatewayProxyResponse {
        status_code: 200,
        multi_value_headers: headers.clone(),
        is_base64_encoded: false,
        body: Some("Hello AWS Lambda HTTP request".into()),
        headers,
    };
    Ok(resp)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_runtime::run(service_fn(handler)).await
}
```



```
}
```

Il client di runtime [Rust per Lambda fornisce](#) anche un'astrazione su questi tipi di eventi che puoi utilizzare con i tipi HTTP nativi, indipendentemente dal servizio che invia gli eventi. Il codice seguente è equivalente all'esempio precedente e funziona immediatamente con gli URL delle funzioni Lambda, gli Application Load Balancer e Gateway API.

#### Note

La cassa [lambda\\_http](#) utilizza la cassa [lambda\\_runtime](#) sottostante. Non è necessario eseguire l'importazione di `lambda_runtime` separatamente.

### Example- Gestione delle richieste HTTP

```
use lambda_http::{service_fn, Error, IntoResponse, Request, RequestExt, Response};

async fn handler(event: Request) -> Result<impl IntoResponse, Error> {
    let resp = Response::builder()
        .status(200)
        .header("content-type", "text/html")
        .body("Hello AWS Lambda HTTP request")
        .map_err(Box::new)?;
    Ok(resp)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_http::run(service_fn(handler)).await
}
```

Per un altro esempio di utilizzo `lambda_http`, consulta l'esempio [di codice](#) `http-axum` nel repository Labs. AWS GitHub

### Eventi HTTP Lambda di esempio per Rust

- [Eventi HTTP Lambda](#): una funzione Rust che gestisce gli eventi HTTP.
- [Eventi HTTP Lambda con intestazioni CORS](#): una funzione Rust che utilizza Tower per inserire le intestazioni CORS.

- [Eventi HTTP Lambda con risorse condivise](#): una funzione Rust che utilizza risorse condivise inizializzate prima della creazione del gestore della funzione.

## Implementazione di funzioni Lambda per Go con gli archivi di file .zip

Questa pagina descrive come compilare la funzione Rust e quindi distribuire il binario compilato su Cargo [Lambda AWS Lambda](#). Mostra anche come distribuire il binario compilato con AWS Command Line Interface e la AWS Serverless Application Model CLI.

### Sections

- [Prerequisiti](#)
- [Compilazione di funzioni Rust su macOS, Windows o Linux](#)
- [Implementazione del file binario di funzioni Rust con Cargo Lambda](#)
- [Richiamo della funzione Rust con Cargo Lambda](#)

### Prerequisiti

- [Rust](#)
- [AWS CLI versione 2](#)

### Compilazione di funzioni Rust su macOS, Windows o Linux

I passaggi seguenti mostrano come creare il progetto per la prima funzione Lambda con Rust e compilarlo con Cargo Lambda, un'estensione open source di terze parti dello [strumento da riga di comando Cargo che semplifica la creazione e l'implementazione di funzioni Rust Lambda](#).

1. Installa [Cargo Lambda](#), un'estensione open source di terze parti per lo strumento da riga di comando Cargo che semplifica la creazione e l'implementazione di funzioni Rust Lambda:

```
cargo install cargo-lambda
```

Per altre opzioni di installazione, consulta [Installazione](#) nella documentazione di Cargo Lambda.

2. Crea la struttura del pacchetto. Questo comando crea un codice di funzione di base in `src/main.rs`. A fini di test, puoi utilizzare questo codice o sostituirlo con il tuo codice personalizzato.

```
cargo lambda new my-function
```

3. All'interno della directory principale del pacchetto, esegui il sottocomando [build](#) per compilare il codice nella funzione.

```
cargo lambda build --release
```

(Facoltativo) Se vuoi usare AWS Graviton2 su Lambda, aggiungi il `--arm64` flag per compilare il codice per ARM. CPUs

```
cargo lambda build --release --arm64
```

4. Prima di distribuire la funzione Rust, configura le credenziali sulla tua macchina. AWS

```
aws configure
```

## Implementazione del file binario di funzioni Rust con Cargo Lambda

Utilizza il sottocomando [deploy](#) per implementare il file binario compilato su Lambda. Questo comando crea un [ruolo di esecuzione](#) e quindi crea la funzione Lambda. Per specificare un ruolo di esecuzione esistente, utilizza il [flag --iam-role](#).

```
cargo lambda deploy my-function
```

## Distribuzione del file binario della funzione Rust con AWS CLI

Puoi anche distribuire il tuo file binario con. AWS CLI

1. Per compilare il pacchetto di implementazione .zip, utilizza il sottocomando [build](#).

```
cargo lambda build --release --output-format zip
```

2. Per implementare il pacchetto .zip su Lambda, eseguire il comando [create-function](#).

- Per `--runtime`, specificare `provided.al2023`. Questo è un [runtime solo per il sistema operativo](#). I runtime solo per il sistema operativo vengono utilizzati per distribuire file binari compilati e runtime personalizzati su Lambda.
- Per `--role`, specifica l'ARN del [ruolo di esecuzione](#).

```
aws lambda create-function \  
  --function-name my-function \  
  --runtime provided.al2023 \  
  --role arn:aws:iam::123456789012:role/lambda-role
```

```
--role arn:aws:iam::111122223333:role/lambda-role \  
--handler rust.handler \  
--zip-file fileb://target/lambda/my-function/bootstrap.zip
```

## Distribuzione del binario della funzione Rust con la CLI AWS SAM

Puoi anche distribuire il tuo file binario con la AWS SAM CLI.

1. Crea un AWS SAM modello con la definizione delle risorse e delle proprietà. Per Runtime, specificare `provided.al2023`. Questo è un [runtime solo per il sistema operativo](#). I runtime solo per il sistema operativo vengono utilizzati per distribuire file binari compilati e runtime personalizzati su Lambda.

Per ulteriori informazioni sulla distribuzione delle funzioni Lambda AWS SAM utilizzando, [AWS::Serverless::Function](#) consulta la Guida per AWS Serverless Application Model gli sviluppatori.

### Example Definizione di risorse e proprietà SAM per un file binario Rust

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Description: SAM template for Rust binaries  
Resources:  
  RustFunction:  
    Type: AWS::Serverless::Function  
    Properties:  
      CodeUri: target/lambda/my-function/  
      Handler: rust.handler  
      Runtime: provided.al2023  
Outputs:  
  RustFunction:  
    Description: "Lambda Function ARN"  
    Value: !GetAtt RustFunction.Arn
```

2. Utilizza il sottocomando [build](#) per compilare la funzione.

```
cargo lambda build --release
```

3. Utilizza il comando [sam deploy](#) per implementare la funzione su Lambda.

```
sam deploy --guided
```

Per ulteriori informazioni sulla creazione di funzioni Rust con la AWS SAM CLI, consulta [Creazione di funzioni Rust Lambda con Cargo Lambda nella Developer Guide](#).AWS Serverless Application Model

Richiamo della funzione Rust con Cargo Lambda

Utilizza il sottocomando [invoke](#) per testare la tua funzione con un payload.

```
cargo lambda invoke --remote --data-ascii '{"command": "Hello world"}' my-function
```

Invocare la funzione Rust con AWS CLI

Puoi anche usare il AWS CLI per richiamare la funzione.

```
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --  
payload '{"command": "Hello world"}' /tmp/out.txt
```

L'`cli-binary-format` opzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

## Utilizzo dei livelli per le funzioni Lambda in Go

Consigliamo di non utilizzare i livelli per gestire le dipendenze per le funzioni Lambda scritte in Go. Questo perché le funzioni Lambda in Go vengono compilate in un unico eseguibile, che viene fornito a Lambda quando si distribuisce la funzione. Questo eseguibile contiene il codice di funzione compilato, insieme a tutte le sue dipendenze. L'uso dei livelli non solo complica questo processo, ma comporta anche un aumento dei tempi di avvio a freddo, poiché le funzioni devono caricare manualmente assieme aggiuntivi in memoria durante la fase di inizializzazione.

Per utilizzare dipendenze esterne con i gestori Go, includile direttamente nel pacchetto di implementazione. In questo modo, semplifichi il processo di implementazione e sfrutti anche le ottimizzazioni integrate del compilatore Go. Per un esempio di come importare e utilizzare una dipendenza come l'SDK AWS per Go nella tua funzione, consulta [the section called “Gestore”](#).

## Registrazione e monitoraggio delle funzioni Lambda con Rust

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon CloudWatch. La tua funzione Lambda include un gruppo di CloudWatch log e un flusso di log per ogni istanza della tua funzione. L'ambiente del runtime Lambda invia i dettagli su ogni richiamo al flusso di log e inoltra i log e l'output del codice della funzione.

Per ulteriori informazioni, consulta [Invio dei log delle funzioni Lambda ai CloudWatch log](#). Per informazioni sulla configurazione dei formati di log, consulta [Configurazione dei formati di log JSON e testo normale](#). In questa pagina viene descritto come generare l'output del log dal codice della funzione Lambda.

### Creazione di una funzione che scrive i log

Per generare i log dal codice della funzione, è possibile utilizzare qualsiasi funzione di registrazione che scriva in `stdout` o `stderr`, come la macro `println!`. L'esempio seguente utilizza `println!` per stampare un messaggio all'avvio del gestore della funzione e prima che finisca.

```
use lambda_runtime::{service_fn, LambdaEvent, Error};
use serde_json::{json, Value};
async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    println!("Rust function invoked");
    let payload = event.payload;
    let first_name = payload["firstName"].as_str().unwrap_or("world");
    println!("Rust function responds to {}", &first_name);
    Ok(json!({ "message": format!("Hello, {}!", first_name) }))
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_runtime::run(service_fn(handler)).await
}
```

### Registrazione avanzata con la cassa Tracing

[Tracing](#) è un framework che consente di strumentare i programmi Rust per raccogliere informazioni diagnostiche strutturate e basate sugli eventi. Questo framework fornisce utilità per personalizzare i livelli e i formati di output di registrazione, come la creazione di messaggi di log JSON strutturati. Per utilizzare questo framework, è necessario inizializzare un `subscriber` prima di implementare il gestore della funzione. Dopodiché, è possibile utilizzare macro di tracciamento come `debug`, `info` e `error` per specificare il livello di registrazione desiderato per ogni scenario.



## Example- Utilizzo della cassa Tracing

Tenere presente quanto segue:

- `tracing_subscriber::fmt().json()`: quando questa opzione è inclusa, i log vengono formattati in JSON. Per utilizzare questa opzione, è necessario includere la funzionalità `json` nella dipendenza `tracing-subscriber` (ad esempio, `tracing-subscriber = { version = "0.3.11", features = ["json"] }`).
- `#[tracing::instrument(skip(event), fields(req_id = %event.context.request_id))]`: questa annotazione genera un intervallo ogni volta che viene richiamato il gestore. L'intervallo aggiunge l'ID della richiesta su ciascuna riga di log.
- `{ %first_name }`: questo costrutto aggiunge il campo `first_name` alla riga di log in cui viene utilizzato. Il valore di questo campo corrisponde alla variabile con lo stesso nome.

```
use lambda_runtime::{service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
#[tracing::instrument(skip(event), fields(req_id = %event.context.request_id))]
async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    tracing::info!("Rust function invoked");
    let payload = event.payload;
    let first_name = payload["firstName"].as_str().unwrap_or("world");
    tracing::info!({ %first_name }, "Rust function responds to event");
    Ok(json!({ "message": format!("Hello, {first_name}!") }))
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt().json()
        .with_max_level(tracing::Level::INFO)
        // this needs to be set to remove duplicated information in the log.
        .with_current_span(false)
        // this needs to be set to false, otherwise ANSI color codes will
        // show up in a confusing manner in CloudWatch logs.
        .with_ansi(false)
        // disabling time is handy because CloudWatch will add the ingestion time.
        .without_time()
        // remove the name of the function from every log entry
        .with_target(false)
        .init();
    lambda_runtime::run(service_fn(handler)).await
}
```

```
}
```

Quando viene richiamata questa funzione Rust, stampa due righe di log simili alle seguenti:

```
{"level":"INFO","fields":{"message":"Rust function invoked"},"spans":  
[{"req_id":"45daaaa7-1a72-470c-9a62-e79860044bb5","name":"handler"}]}  
{"level":"INFO","fields":{"message":"Rust function responds to  
event","first_name":"David"},"spans":[{"req_id":"45daaaa7-1a72-470c-9a62-  
e79860044bb5","name":"handler"}]}
```

## Runtime Lambda

Lambda supporta più lingue attraverso l'uso di runtime. Un runtime fornisce un ambiente specifico del linguaggio che inoltra gli eventi di chiamata, le informazioni di contesto e le risposte tra Lambda e la funzione. Puoi utilizzare i runtime forniti da Lambda o puoi crearne uno personalizzato. Per informazioni sui runtime open source, vedere [the section called “Repository open source”](#)

Lambda è indipendente dal runtime scelto dall'utente. Per funzioni semplici, interpreta linguaggi come Python e Node.js offrono le prestazioni più veloci. Per le funzioni con calcoli più complessi, i linguaggi compilati come Java sono spesso più lenti da inizializzare ma vengono eseguiti rapidamente nell'handler Lambda. La scelta del runtime è influenzata anche dalle preferenze degli sviluppatori e dalla familiarità del linguaggio.

Ogni versione del linguaggio di programmazione principale ha un runtime separato, con un identificatore di runtime univoco, ad esempio `nodejs24.x` o `python3.14`. Per configurare una funzione in modo da utilizzare una nuova versione principale del linguaggio, è necessario modificare l'identificatore di runtime. Poiché AWS Lambda non è possibile garantire la retrocompatibilità tra le versioni principali, si tratta di un'operazione orientata al cliente.

Per una [funzione definita come immagine di container](#), scegli un runtime e la distribuzione Linux quando crei l'immagine di container. Per modificare il runtime, è necessario creare una nuova immagine di container.

Quando si utilizza un archivio di file con estensione `.zip` per il pacchetto di implementazione, è necessario scegliere un runtime quando si crea la funzione. Per modificare il runtime, è possibile [aggiornare la configurazione della funzione](#). Il runtime è associato a una delle distribuzioni Amazon Linux. L'ambiente di esecuzione sottostante fornisce ulteriori librerie e [variabili di ambiente](#) alle quali è possibile accedere dal codice della funzione.

Lambda richiama la funzione in un [ambiente di esecuzione](#). Un ambiente di esecuzione fornisce un ambiente di runtime sicuro e isolato che gestisce le risorse necessarie per eseguire la funzione. Lambda riutilizza l'ambiente di esecuzione da una chiamata precedente, se disponibile, oppure può creare un nuovo ambiente di esecuzione.

Per usare altri linguaggi in Lambda, come [Go](#) o [Rust](#), usa un runtime. [OS-only](#) L'ambiente di esecuzione Lambda offre un'[interfaccia di runtime](#) per ricevere gli eventi di chiamata e inviare le risposte. Puoi implementare altri linguaggi implementando un [runtime personalizzato](#) insieme al codice della funzione o in un [livello](#).

## Runtime supportati

Nella tabella seguente sono elencati i runtime Lambda supportati e le date di ritiro previste. Una volta ritirato un runtime, è ancora possibile creare e aggiornare le funzioni per un periodo limitato. Per ulteriori informazioni, consulta [the section called “Utilizzo del runtime dopo la deprecazione”](#). La tabella fornisce le date attualmente previste per la deprecazione del runtime, in base al nostro [the section called “Policy di deprecazione del runtime”](#) Tali date vengono indicate a scopo di pianificazione e sono soggette a modifiche.

### Note

I runtime Node.js 26 e Python 3.15 sono in anteprima pubblica. I runtime di anteprima non sono coperti dal Lambda SLA o dal supporto tecnico e non devono essere utilizzati per i carichi di lavoro di produzione.

Lo scopo dei runtime di anteprima è raccogliere feedback da clienti, partner e comunità linguistiche a monte prima della disponibilità generale, mentre sono ancora possibili modifiche sostanziali. Puoi fornire un feedback tramite i GitHub numeri dedicati per [Node.js 26](#) e [Python 3.15](#). Per ulteriori informazioni, consulta questo [post del blog](#).

### Important

La fine del ciclo di vita di Amazon Linux 2 è prevista per il 30 giugno 2026. I runtime Lambda e le immagini di base dei contenitori per Java 8 (AL2), Java 11 (AL2), Java 17 (AL2), Python 3.10, Python 3.11 e provided.al2 continueranno a ricevere patch per problemi di sicurezza importanti e [critici di](#) Amazon Linux 2, oltre alle patch di runtime del linguaggio, fino alle date di deprecazione mostrate nella tabella seguente.

Consigliamo ai clienti di eseguire l'aggiornamento a un runtime basato su Amazon Linux 2023 il prima possibile. Per i clienti che effettuano l'aggiornamento a Java 21 o Java 25, puoi utilizzare [AWS Transform custom](#) per assisterti con questi aggiornamenti. Per i clienti che non sono in grado di aggiornare la propria versione Java, forniamo runtime basati su Amazon Linux 2023 per Java 8, Java 11 e Java 17 che seguono la pianificazione di deprecazione di AL2023.

| Nome        | Identificatore | Sistema operativo | Data di ritiro  | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|-------------|----------------|-------------------|-----------------|---------------------------|-------------------------------|
| Node.js 26  | nodejs26.x     | Amazon Linux 2023 | Non programmato | Non programmato           | Non programmato               |
| Node.js 24  | nodejs24.x     | Amazon Linux 2023 | 30 aprile 2028  | 1 giugno 2028             | 1 luglio 2028                 |
| Node.js 22  | nodejs22.x     | Amazon Linux 2023 | 30 aprile 2027  | 1 giugno 2027             | 1 luglio 2027                 |
| Python 3.15 | python3.15     | Amazon Linux 2023 | Non programmato | Non programmato           | Non programmato               |
| Python 3.14 | python3.14     | Amazon Linux 2023 | 30 giugno 2029  | 31 luglio 2029            | 31 agosto 2029                |
| Python 3.13 | python3.13     | Amazon Linux 2023 | 30 giugno 2029  | 31 luglio 2029            | 31 agosto 2029                |
| Python 3.12 | python3.12     | Amazon Linux 2023 | 31 ottobre 2028 | 30 novembre 2028          | 10 gennaio 2029               |
| Python 3.11 | python3.11     | Amazon Linux 2    | 30 giugno 2027  | 31 luglio 2027            | 31 agosto 2027                |
| Python 3.10 | python3.10     | Amazon Linux 2    | 31 ottobre 2026 | 1 febbraio 2027           | 3 marzo 2027                  |

| Nome                      | Identificatore | Sistema operativo | Data di ritiro   | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|---------------------------|----------------|-------------------|------------------|---------------------------|-------------------------------|
| Java 25                   | java25         | Amazon Linux 2023 | 30 giugno 2029   | 31 luglio 2029            | 31 agosto 2029                |
| Java 21                   | java21         | Amazon Linux 2023 | 30 giugno 2029   | 31 luglio 2029            | 31 agosto 2029                |
| Java 17                   | java17.a12023  | Amazon Linux 2023 | 30 giugno 2029   | 31 luglio 2029            | 31 agosto 2029                |
| Java 11                   | java11.a12023  | Amazon Linux 2023 | 30 giugno 2029   | 31 luglio 2029            | 31 agosto 2029                |
| Java 8                    | java8.a12023   | Amazon Linux 2023 | 30 giugno 2029   | 31 luglio 2029            | 31 agosto 2029                |
| Java 17                   | java17         | Amazon Linux 2    | 30 giugno 2027   | 31 luglio 2027            | 31 agosto 2027                |
| Java 11                   | java11         | Amazon Linux 2    | 30 giugno 2027   | 31 luglio 2027            | 31 agosto 2027                |
| Java 8                    | java8.a12      | Amazon Linux 2    | 30 giugno 2027   | 31 luglio 2027            | 31 agosto 2027                |
| .NET 10                   | dotnet10       | Amazon Linux 2023 | 14 novembre 2028 | 14 dicembre 2028          | 15 gennaio 2029               |
| .NET 9 (solo contenitore) | dotnet9        | Amazon Linux 2023 | 10 novembre 2026 | Non programmato           | Non programmato               |
| .NET 8                    | dotnet8        | Amazon Linux 2023 | 10 novembre 2026 | 1 febbraio 2027           | 3 marzo 2027                  |
| Ruby 4.0                  | ruby4.0        | Amazon Linux 2023 | 31 marzo 2029    | 30 aprile 2029            | 31 maggio 2029                |

| Nome            | Identificatore  | Sistema operativo | Data di ritiro | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|-----------------|-----------------|-------------------|----------------|---------------------------|-------------------------------|
| Ruby 3.4        | ruby3.4         | Amazon Linux 2023 | 31 marzo 2028  | 30 aprile 2028            | 31 maggio 2028                |
| Ruby 3.3        | ruby3.3         | Amazon Linux 2023 | 31 marzo 2027  | 30 aprile 2027            | 31 maggio 2027                |
| OS-only Runtime | provided.al2023 | Amazon Linux 2023 | 30 giugno 2029 | 31 luglio 2029            | 31 agosto 2029                |

#### Note

Per le nuove regioni, Lambda non supporterà i runtime per cui è previsto il ritiro entro i prossimi 6 mesi.

Lambda mantiene aggiornati i runtime gestiti e le immagini di base dei container corrispondenti con patch e supporto per le versioni minori. Per ulteriori informazioni, consulta [Aggiornamenti di runtime Lambda](#).

Per interagire in modo programmatico con altre Servizi AWS risorse della tua funzione Lambda, puoi utilizzare uno degli SDK. AWS I Node.js runtime Python e Ruby includono una versione dell'SDK. AWS Tuttavia, per mantenere il pieno controllo delle dipendenze e massimizzare la [compatibilità con le versioni precedenti](#) durante gli aggiornamenti automatici del runtime, ti consigliamo di includere sempre i moduli SDK utilizzati dal codice (insieme alle eventuali dipendenze) nel pacchetto di distribuzione della funzione o in un livello Lambda. [???](#)

Ti consigliamo di utilizzare la versione SDK inclusa nel runtime solo quando non puoi includere pacchetti aggiuntivi nella distribuzione. Ad esempio, quando crei la tua funzione utilizzando l'editor di codice della console Lambda o utilizzando il codice funzione in linea in un modello. CloudFormation

Lambda aggiorna periodicamente le versioni degli AWS SDK inclusi nei runtime Python Node.js e Ruby. Per determinare la versione dell' AWS SDK inclusa nel runtime che stai utilizzando, consulta le seguenti sezioni:

- [Runtime-included Versioni SDK \(\) Node.js](#)
- [Runtime-included Versioni SDK \(Python\)](#)
- [Runtime-included Versioni SDK \(Ruby\)](#)

Lambda continua a supportare il linguaggio di programmazione Go dopo il ritiro del runtime Go 1.x. Per ulteriori informazioni, consulta [Migrazione AWS Lambda delle funzioni dal Go1.x runtime al runtime personalizzato su Amazon Linux 2 nel blog di](#) Compute. AWS

Tutti i runtime Lambda supportati supportano sia le architetture x86\_64 che le architetture arm64.

## Nuovi rilasci di runtime

Lambda fornisce runtime gestiti per le nuove versioni di linguaggio solo quando il rilascio raggiunge la fase di supporto a lungo termine (LTS) del ciclo di rilascio del linguaggio. Ad esempio, per il ciclo di [Node.js rilascio](#), quando il rilascio raggiunge la fase Active LTS.

Prima che raggiunga la fase di supporto a lungo termine, il rilascio rimane in fase di sviluppo e può ancora essere soggetto a modifiche sostanziali. Lambda applica in automatico gli aggiornamenti di runtime per impostazione predefinita, quindi le modifiche sostanziali a una versione di runtime potrebbe impedire alle funzioni di funzionare come previsto.

Lambda non fornisce runtime gestiti per le versioni di linguaggio che non sono pianificate per il rilascio LTS.

L'elenco seguente mostra il mese di lancio previsto per i runtime Lambda futuri. Queste date sono solo indicative e soggette a modifica.

- Node.js 26 - novembre 2026
- Python 3.15 - novembre 2026

## Policy di deprecazione del runtime

I runtime Lambda per gli archivi di file .zip si basano su una combinazione di sistema operativo, linguaggio di programmazione e librerie software soggette a manutenzione e aggiornamenti di sicurezza. La politica di deprecazione standard di Lambda prevede la deprecazione di un runtime quando uno dei componenti principali del runtime raggiunge la fine del supporto a lungo termine della comunità (LTS) e gli aggiornamenti di sicurezza non sono più disponibili. In genere si tratta del

runtime del linguaggio, anche se in alcuni casi un runtime può diventare obsoleto perché il sistema operativo (OS) raggiunge la fine dell'LTS.

Dopo che un runtime è diventato obsoleto, non AWS può più applicare patch di sicurezza o aggiornamenti a quel runtime e le funzioni che utilizzano quel runtime non sono più idonee per il supporto tecnico. Tali runtime obsoleti vengono forniti «così come sono», senza alcuna garanzia e possono contenere bug, errori, difetti o altre vulnerabilità.

Per ulteriori informazioni sulla gestione degli aggiornamenti di runtime e della deprecazione, consulta le sezioni seguenti e Gestire gli aggiornamenti del runtime nel blog di Compute. [AWS Lambda](#) AWS

#### Important

Occasionalmente, Lambda ritarda la deprecazione di un runtime Lambda per un periodo limitato oltre la data di fine del supporto della versione di linguaggio supportata dal runtime. Durante questo periodo, Lambda applica le patch di sicurezza soltanto al sistema operativo runtime. Lambda non applica le patch di sicurezza ai runtime dei linguaggi di programmazione dopo la data di fine del supporto.

## Modello di responsabilità condivisa

Lambda è responsabile della cura e della pubblicazione degli aggiornamenti di sicurezza per tutti i runtime gestiti e le immagini dei container supportati. Per impostazione predefinita, Lambda applica automaticamente questi aggiornamenti alle funzioni che utilizzano runtime gestiti. Se l'impostazione predefinita per l'aggiornamento automatico del runtime è stata modificata, consulta il [modello di responsabilità condivisa dei controlli di gestione del runtime](#). Per le funzioni implementate utilizzando immagini di container, l'utente sarà responsabile della ricostruzione dell'immagine di container della funzione dall'immagine di base più recente e della nuova implementazione dell'immagine di container.

Quando un runtime è obsoleto, la responsabilità di Lambda per l'aggiornamento del runtime gestito e delle immagini di base del contenitore cessa. L'utente è responsabile dell'aggiornamento delle funzioni per utilizzare un runtime o un'immagine di base supportata.

In tutti i casi, l'utente è responsabile dell'applicazione degli aggiornamenti al codice della funzione, comprese le sue dipendenze. Le tue responsabilità nell'ambito del modello di responsabilità condivisa sono riassunte nella tabella seguente.



| Fase del ciclo di vita di runtime          | Responsabilità di Lambda                                                                                                                                                                                                                                                                                        | Le tue responsabilità                                                                                                                                                                                                                                     |
|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Runtime gestito supportato                 | <p>Fornisci aggiornamenti di runtime regolari con patch di sicurezza e altri aggiornamenti.</p> <p>Applica automaticamente gli aggiornamenti di runtime per impostazione predefinita (vedi <a href="#">the section called “Modalità di aggiornamento del runtime”</a> per i comportamenti non predefiniti).</p> | <p>Aggiorna il codice della funzione, comprese le dipendenze, per risolvere eventuali vulnerabilità di sicurezza.</p>                                                                                                                                     |
| Immagine di container supportata           | <p>Fornisci aggiornamenti di runtime regolari all'immagine di base del container con patch di sicurezza e altri aggiornamenti.</p>                                                                                                                                                                              | <p>Aggiorna il codice della funzione, comprese le dipendenze, per risolvere eventuali vulnerabilità di sicurezza.</p> <p>Ricostruisci e reimplementa regolarmente l'immagine di container utilizzando l'immagine di base più recente.</p>                 |
| Runtime gestito prossimo alla deprecazione | <p>Informa i clienti prima della deprecazione del runtime tramite documentazione, e-mail e Dashboard Health Trusted Advisor</p> <p>La responsabilità per gli aggiornamenti del runtime termina quando il runtime diventa obsoleto.</p>                                                                          | <p>Monitora la documentazione di Lambda Dashboard Health, invia e-mail o informazioni sulla deprecazione in fase Trusted Advisor di esecuzione.</p> <p>Aggiorna le funzioni a un runtime supportato prima che il runtime precedente diventi obsoleto.</p> |

| Fase del ciclo di vita di runtime                       | Responsabilità di Lambda                                                                                                                                                                                                                      | Le tue responsabilità                                                                                                                                                  |
|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Immagine di container che si avvicina alla deprecazione | <p>Le notifiche di deprecazione non sono disponibili per le funzioni che utilizzano immagini di container.</p> <p>La responsabilità per gli aggiornamenti dell'immagine di base del container termina quando il runtime diventa obsoleto.</p> | Fai attenzione alle pianificazioni di deprecazione e alle funzioni di aggiornamento a un'immagine di base supportata prima che l'immagine precedente diventi obsoleta. |

## Utilizzo del runtime dopo la deprecazione

Dopo che un runtime è diventato obsoleto, AWS potrebbe non applicare più patch di sicurezza o aggiornamenti a quel runtime e le funzioni che utilizzano quel runtime non sono più idonee al supporto tecnico. Sebbene sia possibile continuare a richiamare le funzioni a tempo indeterminato, si consiglia AWS vivamente di eseguire la migrazione a un runtime supportato. I runtime obsoleti vengono forniti «così come sono», senza alcuna garanzia e possono contenere bug, errori, difetti o altre vulnerabilità. Le funzioni che utilizzano un runtime obsoleto possono inoltre presentare un peggioramento delle prestazioni o altri problemi, come la scadenza del certificato, che possono impedirne il corretto funzionamento.

È possibile aggiornare una funzione per utilizzare un runtime supportato più recente in qualsiasi momento dopo che un runtime è diventato obsoleto. Si consiglia di testare la funzione con il nuovo runtime prima di applicare le modifiche negli ambienti di produzione. Non sarà possibile ripristinare il runtime obsoleto dopo il blocco degli aggiornamenti delle funzioni. Consigliamo di utilizzare le [versioni](#) e gli [alias](#) delle funzioni per consentire una implementazione sicura con rollback.

La sequenza temporale seguente descrive cosa succede quando un runtime viene dichiarato obsoleto:

| Fase del ciclo di vita di runtime     | Quando                                     | What                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------------------------|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Periodo di preavviso di deprezzamento | Almeno 180 giorni prima della deprecazione | <ul style="list-style-type: none"> <li>AWS invia notifiche tramite e-mail e Dashboard Health agli account che dispongono di funzioni che utilizzano questo runtime nella loro \$LATEST versione.</li> <li>Le funzioni interessate sono elencate anche nella scheda Modifiche <a href="#">Dashboard Health pianificate</a> e nella verifica dei runtime <a href="#">AWS Trusted Advisor obsoleti</a>.</li> </ul> |
| Raggiunta obsolescenza                | Data di ritiro                             | <ul style="list-style-type: none"> <li>AWS potrebbe non applicare più aggiornamenti di sicurezza o altri aggiornamenti.</li> <li>Le funzioni non sono più idonee al supporto tecnico.</li> <li>Non è più possibile creare o aggiornare funzioni utilizzando il runtime obsoleto nella console Lambda. Puoi continuare a creare e aggiornare le funzioni tramite AWS CLI,, AWS SAM o. CloudFormation</li> </ul>  |
| Blocco creazione funzioni             | Almeno 30 giorni dopo la deprecazione      | <ul style="list-style-type: none"> <li>Lambda inizia a bloccare la creazione di nuove funzioni.</li> <li>Puoi continuare ad aggiornare il codice e la configurazione per le funzioni esistenti tramite AWS CLI, AWS SAM, o.. CloudFormation</li> </ul>                                                                                                                                                          |
| Blocco aggiornamento funzioni         | Almeno 60 giorni dopo la deprecazione      | <ul style="list-style-type: none"> <li>Lambda inizia a bloccare l'aggiornamento del codice e della configurazione per le funzioni esistenti.</li> <li>Puoi comunque aggiornare la configurazione della funzione a un runtime supportato. Tuttavia, il ripristino del runtime obsoleto potrebbe essere bloccato.</li> </ul>                                                                                      |

### Note

Per alcuni runtime, AWS sta posticipando le date block-function-create e block-function-update oltre i soliti 30 e 60 giorni dopo la deprecazione. AWS ha apportato questa modifica in risposta al feedback dei clienti per darti più tempo per aggiornare le tue funzioni. Fai riferimento alle tabelle riportate in [the section called “Runtime supportati”](#) e [the section called “Runtime obsoleti”](#) per vedere le date dei runtime. Lambda non inizierà a bloccare le creazioni o gli aggiornamenti delle funzioni prima delle date indicate in queste tabelle.

## Ricezione di notifiche di ritiro del runtime

Quando un runtime si avvicina alla data di deprecazione, Lambda invia un avviso via email se alcune funzioni utilizzano quel runtime. Account AWS Le notifiche vengono visualizzate anche nei formati e in. Dashboard Health AWS Trusted Advisor

- Ricezione di notifiche e-mail:

Lambda ti invia un avviso e-mail almeno 180 giorni prima che un runtime venga ritirato. Questa e-mail elenca le versioni \$LATEST di tutte le funzioni che utilizzano il runtime. Per visualizzare un elenco completo delle versioni delle funzioni interessate, usa Trusted Advisor o consulta [the section called “Ottenere dati sulle funzioni in base al runtime”](#).

Lambda invia una notifica via email al contatto principale Account AWS del tuo account. Per informazioni sulla visualizzazione o l'aggiornamento degli indirizzi e-mail nel tuo account, consulta la pagina [Aggiornamento delle informazioni di contatto](#) della Guida generale di AWS .

- Ricezione di notifiche tramite: Dashboard Health

Dashboard Health Visualizza una notifica almeno 180 giorni prima che un runtime diventi obsoleto. Le notifiche compaiono nella pagina Stato del tuo account, in [Altre notifiche](#). La scheda Risorse interessate della notifica elenca le versioni \$LATEST di tutte le funzioni che utilizzano il runtime.

### Note

Per visualizzare un elenco completo e aggiornato delle versioni delle funzioni interessate, usa Trusted Advisor o consulta. [the section called “Ottenere dati sulle funzioni in base al runtime”](#)

Dashboard Health le notifiche scadono 90 giorni dopo che il runtime interessato è diventato obsoleto.

- Utilizzando AWS Trusted Advisor

Trusted Advisor visualizza una notifica almeno 180 giorni prima che un runtime diventi obsoleto. Le notifiche compaiono nella pagina [Sicurezza](#). Un elenco delle funzioni interessate viene visualizzato in Funzioni AWS Lambda che utilizzano runtime ritirati. Questo elenco di funzioni mostra sia le versioni \$LATEST sia le versioni pubblicate e si aggiorna in automatico per riflettere lo stato attuale delle funzioni.

Puoi attivare le notifiche e-mail Trusted Advisor settimanali dalla [pagina](#) Preferenze della Trusted Advisor console.

## Runtime obsoleti

Per i seguenti runtime è stata raggiunta la fine del supporto:

| Nome            | Identificatore | Sistema operativo | Data di ritiro   | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|-----------------|----------------|-------------------|------------------|---------------------------|-------------------------------|
| OS-only Runtime | provided.al2   | Amazon Linux 2    | 31 luglio 2026   | 1 febbraio 2027           | 3 marzo 2027                  |
| Node.js 20      | nodejs20.x     | Amazon Linux 2023 | 30 aprile 2026   | 1 febbraio 2027           | 3 marzo 2027                  |
| Ruby 3.2        | ruby3.2        | Amazon Linux 2    | 31 marzo 2026    | 1 febbraio 2027           | 3 marzo 2027                  |
| Python 3.9      | python3.9      | Amazon Linux 2    | 15 dicembre 2025 | 1 febbraio 2027           | 3 marzo 2027                  |
| Node.js 18      | nodejs18.x     | Amazon Linux 2    | 1 settembre 2025 | 1 febbraio 2027           | 3 marzo 2027                  |

| Nome                    | Identificatore | Sistema operativo | Data di ritiro   | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|-------------------------|----------------|-------------------|------------------|---------------------------|-------------------------------|
| .NET 6                  | dotnet6        | Amazon Linux 2    | 20 dicembre 2024 | 1 febbraio 2027           | 3 marzo 2027                  |
| Python 3.8              | python3.8      | Amazon Linux 2    | 14 ottobre 2024  | 1 febbraio 2027           | 3 marzo 2027                  |
| Node.js 16              | nodejs16.x     | Amazon Linux 2    | 12 giugno 2024   | 1 febbraio 2027           | 3 marzo 2027                  |
| .NET 7 (solo container) | dotnet7        | Amazon Linux 2    | 14 maggio 2024   | N/A                       | N/A                           |
| Java 8                  | java8          | Amazon Linux      | 8 gennaio 2024   | 8 febbraio 2024           | 3 marzo 2027                  |
| Go 1.x                  | go1.x          | Amazon Linux      | 8 gennaio 2024   | 8 febbraio 2024           | 3 marzo 2027                  |
| OS-only Runtime         | provided       | Amazon Linux      | 8 gennaio 2024   | 8 febbraio 2024           | 3 marzo 2027                  |
| Ruby 2.7                | ruby2.7        | Amazon Linux 2    | 7 dicembre 2023  | 9 gennaio 2024            | 3 marzo 2027                  |
| Node.js 14              | nodejs14.x     | Amazon Linux 2    | 4 dicembre 2023  | 9 gennaio 2024            | 3 marzo 2027                  |
| Python 3.7              | python3.7      | Amazon Linux      | 4 dicembre 2023  | 9 gennaio 2024            | 3 marzo 2027                  |
| .NET Core 3.1           | dotnetcore3.1  | Amazon Linux 2    | 3 aprile 2023    | 3 aprile 2023             | 3 maggio 2023                 |
| Node.js 12              | nodejs12.x     | Amazon Linux 2    | 31 marzo 2023    | 31 marzo 2023             | 30 aprile 2023                |

| Nome                    | Identificatore | Sistema operativo | Data di ritiro | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|-------------------------|----------------|-------------------|----------------|---------------------------|-------------------------------|
| Python 3.6              | python3.6      | Amazon Linux      | 18 luglio 2022 | 18 luglio 2022            | 29 agosto 2022                |
| .NET 5 (solo container) | dotnet5.0      | Amazon Linux 2    | 10 maggio 2022 | N/A                       | N/A                           |
| .NET Core 2.1           | dotnetcore2.1  | Amazon Linux      | 5 gennaio 2022 | 5 gennaio 2022            | 13 aprile 2022                |
| Node.js 10              | nodejs10.x     | Amazon Linux 2    | 30 luglio 2021 | 30 luglio 2021            | 14 febbraio 2022              |
| Ruby 2.5                | ruby2.5        | Amazon Linux      | 30 luglio 2021 | 30 luglio 2021            | 31 marzo 2022                 |
| Python 2.7              | python2.7      | Amazon Linux      | 15 luglio 2021 | 15 luglio 2021            | 30 maggio 2022                |
| Node.js 8,10            | nodejs8.10     | Amazon Linux      | 6 marzo 2020   | 4 febbraio 2020           | 6 marzo 2020                  |
| Node.js 4.3             | nodejs4.3      | Amazon Linux      | 5 marzo 2020   | 3 febbraio 2020           | 5 marzo 2020                  |
| Node.js 4.3 bordo       | nodejs4.3-edge | Amazon Linux      | 5 marzo 2020   | 31 marzo 2019             | 30 aprile 2019                |
| Node.js 6.10            | nodejs6.10     | Amazon Linux      | 12 agosto 2019 | 12 luglio 2019            | 12 agosto 2019                |
| .NET Core 1.0           | dotnetcore1.0  | Amazon Linux      | 27 giugno 2019 | 30 giugno 2019            | 30 luglio 2019                |
| .NET Core 2.0           | dotnetcore2.0  | Amazon Linux      | 30 maggio 2019 | 30 aprile 2019            | 30 maggio 2019                |

| Nome         | Identificatore | Sistema operativo | Data di ritiro | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|--------------|----------------|-------------------|----------------|---------------------------|-------------------------------|
| Node.js 0,10 | nodejs         | Amazon Linux      | 30 agosto 2016 | 30 settembre 2016         | 31 ottobre 2016               |

In quasi tutti i casi, la data di fine del ciclo di vita del sistema operativo o della versione di un linguaggio è nota in anticipo. I collegamenti riportati di seguito forniscono le pianificazioni di fine utilizzo per ogni linguaggio supportato da Lambda come runtime gestito.

#### Policy di supporto su linguaggio e framework

- Node.js— github.com <https://github.com/nodejs/Release#release-schedule>
- Python – [devguide.python.org](http://devguide.python.org)
- Ruby – [www.ruby-lang.org](http://www.ruby-lang.org)
- Java – [www.oracle.com](http://www.oracle.com) e [Domande frequenti su Corretto](#)
- Go – [golang.org](http://golang.org)
- .NET – [dotnet.microsoft.com](http://dotnet.microsoft.com)



## Informazioni su come Lambda gestisce gli aggiornamenti delle versioni di runtime

Lambda mantiene aggiornato ogni runtime gestito con aggiornamenti di sicurezza, correzioni di bug, nuove funzionalità, miglioramenti delle prestazioni e supporto per versioni minori. Questi aggiornamenti di runtime vengono pubblicati come versioni di runtime. Lambda applica gli aggiornamenti di runtime alle funzioni migrando la funzione da una versione di runtime precedente a una nuova versione di runtime.

Per impostazione predefinita, Lambda applica automaticamente gli aggiornamenti di runtime alle funzioni che utilizzano runtime gestiti. Con gli aggiornamenti automatici del runtime, Lambda si assume l'onere operativo dell'applicazione di patch alle versioni di runtime. Per la maggior parte dei clienti, gli aggiornamenti automatici sono la scelta ideale. È possibile modificare questo comportamento predefinito [configurando le impostazioni di gestione del runtime](#).

Lambda pubblica inoltre ogni nuova versione di runtime come immagine di container. Per aggiornare le versioni di runtime per le funzioni basate su container, è necessario [creare una nuova immagine di container](#) dall'immagine di base aggiornata e implementare nuovamente la funzione.

Ogni versione di runtime è associata a un numero di versione e a un nome della risorsa Amazon (ARN). I numeri di versione di runtime utilizzano uno schema numerico definito da Lambda indipendente dai numeri di versione utilizzati dal linguaggio di programmazione. I numeri delle versioni di runtime non sono sempre sequenziali. Ad esempio, la versione 42 potrebbe essere seguita dalla versione 45. L'ARN della versione di runtime è un identificatore univoco per ogni versione di runtime. È possibile visualizzare l'ARN della versione di runtime corrente della funzione nella console Lambda o nella [riga INIT\\_START dei log della funzione](#). Per informazioni sulla gestione degli aggiornamenti di runtime nelle applicazioni critiche per la salute, vedere [the section called "Permissions"](#).

Le versioni di runtime non devono essere confuse con gli identificatori di runtime. Ogni runtime ha un identificatore di runtime univoco, ad esempio `python3.14` o `nodejs24.x`. Questi corrispondono a ciascuna delle principali versioni del linguaggio di programmazione. Le versioni di runtime descrivono la versione patch di un singolo runtime.

### Note

L'ARN per lo stesso numero di versione di runtime può variare Regioni AWS a seconda delle architetture della CPU.

## Argomenti

- [Compatibilità con le versioni precedenti](#)
- [Modalità di aggiornamento del runtime](#)
- [Two-phase implementazione della versione in fase di esecuzione](#)
- [Configurazione delle impostazioni di gestione del runtime Lambda](#)
- [Rollback di una versione di runtime Lambda](#)
- [Identificazione delle modifiche alla versione di runtime di Lambda](#)
- [Informazioni sul modello di responsabilità condivisa per la gestione del runtime Lambda](#)
- [Controllo delle autorizzazioni di aggiornamento del runtime Lambda per applicazioni ad alta conformità](#)

## Compatibilità con le versioni precedenti

Lambda fornisce aggiornamenti di runtime compatibili con le versioni precedenti delle funzioni esistenti. Tuttavia, come nel caso delle patch software, ci sono rari casi in cui un aggiornamento del runtime può influire negativamente su una funzione esistente. Ad esempio, le patch di sicurezza possono evidenziare un problema di fondo di una funzione esistente che dipende dal comportamento precedente e non sicuro.

Quando si crea e si implementa una funzione, è importante capire come gestire le dipendenze per evitare potenziali incompatibilità con un futuro aggiornamento di runtime. Ad esempio, supponiamo che la tua funzione dipenda dal pacchetto A, che a sua volta dipende dal pacchetto B. Entrambi i pacchetti sono inclusi nel runtime Lambda (ad esempio, potrebbero essere parti dell'SDK o delle sue dipendenze o parti delle librerie di sistema di runtime).

Considerare i seguenti scenari:

| Implementazione                                                                                                                 | Compatibile con l'applicazione di patch | Motivo                                                                       |
|---------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------|------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>• Pacchetto A: utilizzo dal runtime</li><li>• Pacchetto B: utilizzo dal runtime</li></ul> | Sì                                      | I futuri aggiornamenti di runtime dei pacchetti A e B sono retrocompatibili. |

| Implementazione                                                                                                                                    | Compatibile con l'applicazione di patch | Motivo                                                                                                                                                                                                                                                                                |
|----------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>Pacchetto A: nel pacchetto di distribuzione</li> <li>Pacchetto B: nel pacchetto di distribuzione</li> </ul> | Sì                                      | La distribuzione ha la precedenza, quindi i futuri aggiornamenti di runtime ai pacchetti A e B non hanno alcun effetto.                                                                                                                                                               |
| <ul style="list-style-type: none"> <li>Pacchetto A: nel pacchetto di distribuzione</li> <li>Pacchetto B: utilizzo dal runtime</li> </ul>           | Sì*                                     | <p>I futuri aggiornamenti di runtime del pacchetto B sono retrocompatibili.</p> <p>*Se A e B sono strettamente associati, possono verificarsi problemi di compatibilità. Ad esempio, i botocore pacchetti boto3 e nell' AWS SDK per Python dovrebbero essere distribuiti insieme.</p> |
| <ul style="list-style-type: none"> <li>Pacchetto A: utilizzo dal runtime</li> <li>Pacchetto B: nel pacchetto di distribuzione</li> </ul>           | No                                      | I futuri aggiornamenti di runtime del pacchetto A potrebbero richiedere una versione aggiornata del pacchetto B. Tuttavia, la versione distribuita del pacchetto B ha la precedenza e potrebbe non essere compatibile con le versioni successive del pacchetto A.                     |

Per mantenere la compatibilità con i futuri aggiornamenti di runtime, segui queste best practice:

- Quando possibile, impacchetta tutte le dipendenze: includi tutte le librerie richieste, incluso l' AWS SDK e le relative dipendenze, nel pacchetto di distribuzione. Ciò garantisce un set di componenti stabile e compatibile.

- Usa gli SDK forniti in fase di esecuzione con parsimonia: affidati all'SDK fornito in fase di esecuzione solo quando non puoi includere pacchetti aggiuntivi (ad esempio, quando usi l'editor di codice della console Lambda o il codice in linea in un modello). AWS CloudFormation
- Evita di sovrascrivere le librerie di sistema: non distribuire librerie personalizzate del sistema operativo che potrebbero entrare in conflitto con i futuri aggiornamenti di runtime.

## Modalità di aggiornamento del runtime

Lambda fornisce aggiornamenti di runtime compatibili con le versioni precedenti delle funzioni esistenti. Tuttavia, come nel caso delle patch software, ci sono rari casi in cui un aggiornamento del runtime può influire negativamente su una funzione esistente. Ad esempio, le patch di sicurezza possono evidenziare un problema di fondo di una funzione esistente che dipende dal comportamento precedente e non sicuro. I controlli di gestione del runtime Lambda aiutano a ridurre il rischio di impatto sui carichi di lavoro nel raro caso di incompatibilità delle versioni di runtime. Per ogni [versione della funzione](#) (\$LATEST o versione pubblicata), è possibile scegliere una delle seguenti modalità di aggiornamento del runtime:

- Auto (default) (Automatico [impostazione predefinita]): esegue automaticamente l'aggiornamento alla versione di runtime più recente e sicura tramite una [Two-phase implementazione della versione in fase di esecuzione](#). Consigliamo questa modalità alla maggior parte dei clienti in modo da beneficiare sempre degli aggiornamenti di runtime.
- Aggiornamento delle funzioni: aggiorna alla versione di runtime più recente e sicura quando aggiorni la funzione. Quando aggiorni la funzione, Lambda aggiorna il runtime della funzione alla versione più recente e sicura. Questo approccio sincronizza gli aggiornamenti del runtime con le implementazioni delle funzioni, dando all'utente il controllo su quando Lambda applica gli aggiornamenti del runtime. Con questa modalità, è possibile rilevare e mitigare tempestivamente le rare incompatibilità degli aggiornamenti del runtime. Quando si utilizza questa modalità, è necessario aggiornare regolarmente le funzioni in modo da mantenerne aggiornato il runtime.
- Manuale: aggiorna manualmente la versione del runtime. Specifica una versione di runtime nella configurazione della funzione. La funzione utilizzerà questa versione di runtime a tempo indeterminato.

Nel raro caso in cui una nuova versione di runtime non sia compatibile con una funzione esistente, ciò consente di ripristinare la funzione a una versione di runtime precedente. Si consiglia di non utilizzare la modalità Manual (Manuale) per cercare di ottenere la coerenza del runtime tra le varie implementazioni. Per ulteriori informazioni, consulta [Rollback di una versione di runtime Lambda](#).

La responsabilità dell'applicazione degli aggiornamenti di runtime alle funzioni varia in base alla modalità di aggiornamento del runtime scelta. Per ulteriori informazioni, consulta [Informazioni sul modello di responsabilità condivisa per la gestione del runtime Lambda](#).

## Two-phase implementazione della versione in fase di esecuzione

Lambda introduce nuove versioni di runtime nel seguente ordine:

1. Nella prima fase Lambda utilizza la nuova versione di runtime ogni volta che crei o aggiorni una funzione. Una funzione viene aggiornata quando si chiamano le operazioni [UpdateFunctionCode](#) o [UpdateFunctionConfiguration](#) API.
2. Nella seconda fase, Lambda aggiorna qualsiasi funzione che utilizza la modalità di aggiornamento del runtime Auto e che non è già stata aggiornata alla nuova versione di runtime.

La durata complessiva del processo di rollout varia in base a diversi fattori, tra cui la gravità di eventuali patch di sicurezza incluse nell'aggiornamento del runtime.

Se state sviluppando e implementando attivamente le vostre funzioni, molto probabilmente sceglierete nuove versioni di runtime durante la prima fase. Ciò sincronizza gli aggiornamenti del runtime con gli aggiornamenti delle funzioni. Nel raro caso in cui l'ultima versione di runtime abbia un impatto negativo sull'applicazione, questo approccio offre l'opportunità di intraprendere tempestivamente azioni correttive. Le funzioni che non sono in fase di sviluppo attivo ricevono comunque i vantaggi operativi degli aggiornamenti automatici del runtime durante la seconda fase.

Questo approccio non influisce sulle funzioni impostate sulla modalità Function update (Aggiornamento della funzione) o Manual (Manuale). Le funzioni che utilizzano la modalità Function update (Aggiornamento della funzione) ricevono gli aggiornamenti di runtime più recenti solo quando vengono create o aggiornate. Le funzioni che utilizzano la modalità Manual (Manuale) non ricevono aggiornamenti di runtime.

Lambda pubblica nuove versioni di runtime in modo graduale e continuo nelle Regioni AWS. Se le funzioni sono impostate sulle modalità di aggiornamento automatico o funzionale, è possibile che le funzioni distribuite contemporaneamente in regioni diverse o in momenti diversi nella stessa regione utilizzino versioni di runtime diverse. I clienti che richiedono una coerenza delle versioni di runtime garantita nei propri ambienti devono [utilizzare le immagini di container per implementare le proprie funzioni Lambda](#). La modalità Manuale è concepita come mitigazione temporanea per consentire il rollback della versione di runtime nel raro caso in cui una versione di runtime non sia compatibile con la tua funzione.

## Configurazione delle impostazioni di gestione del runtime Lambda

È possibile configurare le impostazioni di gestione del runtime utilizzando la console Lambda o AWS Command Line Interface (AWS CLI).

### Note

È possibile configurare le impostazioni di gestione del runtime separatamente per ciascuna [versione della funzione](#).

### Configurazione del modo in cui Lambda aggiorna la versione di runtime (console)

1. Aprire la pagina [Functions](#) (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Nella scheda Code (Codice), in Runtime settings (Impostazioni di runtime), scegli Edit runtime management configuration (Modifica configurazione di gestione del runtime).
4. In Runtime management configuration (Configurazione di gestione del runtime), scegli una delle seguenti opzioni:
  - Per aggiornare automaticamente la funzione all'ultima versione di runtime, scegli Auto (Automatico).
  - Per fare in modo che la funzione venga aggiornata all'ultima versione di runtime quando si modifica la funzione, scegli Function update (Aggiornamento della funzione).
  - Perché la funzione venga aggiornata all'ultima versione di runtime solo quando si modifica l'ARN della versione di runtime, scegli Manual (Manuale). L'ARN della versione di runtime è disponibile in Configurazione della gestione del runtime. L'ARN è disponibile anche nella riga INIT\_START dei log della funzione.

Per ulteriori informazioni su queste opzioni, consulta [Modalità di aggiornamento del runtime](#).

5. Selezionare Salva.

### Configurazione del modo in cui Lambda aggiorna la versione di runtime (AWS CLI)

Per configurare la gestione del runtime per una funzione, esegui il comando [put-runtime-management-config](#) della AWS CLI. Quando si utilizza la modalità Manual, è necessario fornire anche l'ARN della versione di runtime.

```
aws lambda put-runtime-management-config \  
  --function-name my-function \  
  --update-runtime-on Manual \  
  --runtime-version-arn arn:aws:lambda:us-east-2::runtime:8eeff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1
```

Verrà visualizzato un output simile al seguente:

```
{  
  "UpdateRuntimeOn": "Manual",  
  "FunctionArn": "arn:aws:lambda:us-east-2:111122223333:function:my-function",  
  "RuntimeVersionArn": "arn:aws:lambda:us-east-2::runtime:8eeff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1"  
}
```

## Rollback di una versione di runtime Lambda

Nel raro caso in cui una nuova versione di runtime non sia compatibile con una funzione esistente, è possibile ripristinare la versione di runtime a una versione precedente. Ciò mantiene l'applicazione funzionante e riduce al minimo le interruzioni, dando il tempo necessario per porre rimedio all'incompatibilità prima di tornare alla versione di runtime più recente.

Lambda non impone un limite per il tempo che è possibile utilizzare una particolare versione di runtime. Tuttavia, consigliamo vivamente di eseguire sempre l'aggiornamento all'ultima versione di runtime il prima possibile per beneficiare delle patch di sicurezza, dei miglioramenti delle prestazioni e delle funzionalità più recenti. Lambda offre la possibilità di ripristinare una versione di runtime precedente solo come mitigazione temporanea nel raro caso in cui si verifichi un problema di compatibilità con gli aggiornamenti del runtime. Le funzioni che utilizzano una versione di runtime precedente per un periodo prolungato potrebbero presentare prestazioni ridotte o problemi, come la scadenza del certificato, che possono impedirne il corretto funzionamento.

Puoi eseguire il rollback di una versione di runtime nei seguenti modi:

- [Utilizzo della modalità manuale per l'aggiornamento del runtime](#)
- [Utilizzo delle versioni pubblicate della funzione](#)

Per ulteriori informazioni, consulta [Introduzione ai controlli AWS Lambda di gestione del runtime nel blog di Compute AWS](#).

## Rollback di una versione di runtime tramite la modalità di aggiornamento del runtime Manual (Manuale)

Se utilizzi la modalità di aggiornamento della versione di runtime Auto (Automatico) o stai utilizzando la versione di runtime `$LATEST`, puoi ripristinare la versione di runtime precedente utilizzando la modalità Manual (Manuale). Per la [versione della funzione](#) che desideri ripristinare, modifica la modalità di aggiornamento della versione di runtime in Manual (Manuale) e specifica l'ARN della versione di runtime precedente. Per ulteriori informazioni su come trovare l'ARN della versione di runtime precedente, consulta [Identificazione delle modifiche alla versione di runtime di Lambda](#).

### Note

Se la versione `$LATEST` della funzione è configurata per utilizzare la modalità Manual (Manuale), non è possibile modificare l'architettura della CPU o la versione di runtime utilizzata dalla funzione. Per apportare queste modifiche, è necessario passare alla modalità di aggiornamento Auto (Automatico) o Function update (Aggiornamento della funzione).

## Rollback di una versione di runtime utilizzando le versioni pubblicate della funzione

Le [versioni delle funzioni](#) pubblicate sono un'istantanea immutabile del codice e della configurazione della funzione `$LATEST` al momento della loro creazione. In modalità Auto (Automatico), Lambda aggiorna automaticamente la versione di runtime delle versioni delle funzioni pubblicate durante la seconda fase del rollout della versione di runtime. In modalità Function update (Aggiornamento della funzione), Lambda non aggiorna la versione di runtime delle versioni delle funzioni pubblicate.

Le versioni delle funzioni pubblicate che utilizzano la modalità Function update (Aggiornamento della funzione) creano quindi un'istantanea statica del codice della funzione, della configurazione e della versione di runtime. Utilizzando la modalità Function update (Aggiornamento della funzione) con le versioni delle funzioni, è possibile sincronizzare gli aggiornamenti di runtime con le varie implementazioni. È possibile coordinare il rollback del codice, della configurazione e delle versioni di runtime anche reindirizzando il traffico verso una versione della funzione pubblicata in precedenza.

È possibile integrare questo approccio nell'integrazione continua e nella distribuzione continua (CI/CD) per il rollback completamente automatico nel raro caso di incompatibilità degli aggiornamenti in fase di esecuzione. Quando si utilizza questo approccio, è necessario aggiornare regolarmente la funzione e pubblicare nuove versioni delle funzioni per avere gli ultimi aggiornamenti di runtime. Per ulteriori informazioni, consulta [Informazioni sul modello di responsabilità condivisa per la gestione del runtime Lambda](#).



## Identificazione delle modifiche alla versione di runtime di Lambda

Il numero [di versione di runtime](#) e l'ARN vengono registrati nella riga di INIT\_START log, che Lambda invia a CloudWatch Logs ogni volta che crea un nuovo ambiente di esecuzione. ??? Poiché l'ambiente di esecuzione utilizza la stessa versione di runtime per tutte le chiamate di funzione, Lambda emette la riga di log INIT\_START solo quando esegue la fase di inizializzazione. Non emette questa riga di log per ogni chiamata di funzione. Lambda invia la riga di CloudWatch log a Logs, ma non è visibile nella console.

### Note

I numeri delle versioni di runtime non sono sempre sequenziali. Ad esempio, la versione 42 potrebbe essere seguita dalla versione 45.

### Example Riga di log INIT\_START di esempio

```
INIT_START Runtime Version: python:3.13.v14    Runtime Version ARN: arn:aws:lambda:eu-south-1::runtime:7b620fc2e66107a1046b140b9d320295811af3ad5d4c6a011fad1fa65127e9e6I
```

Anziché lavorare direttamente con i log, puoi utilizzare [Amazon CloudWatch Contributor Insights](#) per identificare le transizioni tra le versioni di runtime. La regola seguente conta le versioni di runtime distinte di ciascuna riga di log INIT\_START. Per utilizzare la regola, sostituisci il nome del gruppo di log di esempio /aws/lambda/\* con il prefisso appropriato per la funzione o il gruppo di funzioni.

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "AggregateOn": "Count",
  "Contribution": {
    "Filters": [
      {
        "Match": "eventType",
        "In": [
          "INIT_START"
        ]
      }
    ]
  },
}
```

```

    "Keys": [
      "runtimeVersion",
      "runtimeVersionArn"
    ],
  },
  "LogFormat": "CLF",
  "LogGroupNames": [
    "/aws/lambda/*"
  ],
  "Fields": {
    "1": "eventType",
    "4": "runtimeVersion",
    "8": "runtimeVersionArn"
  }
}

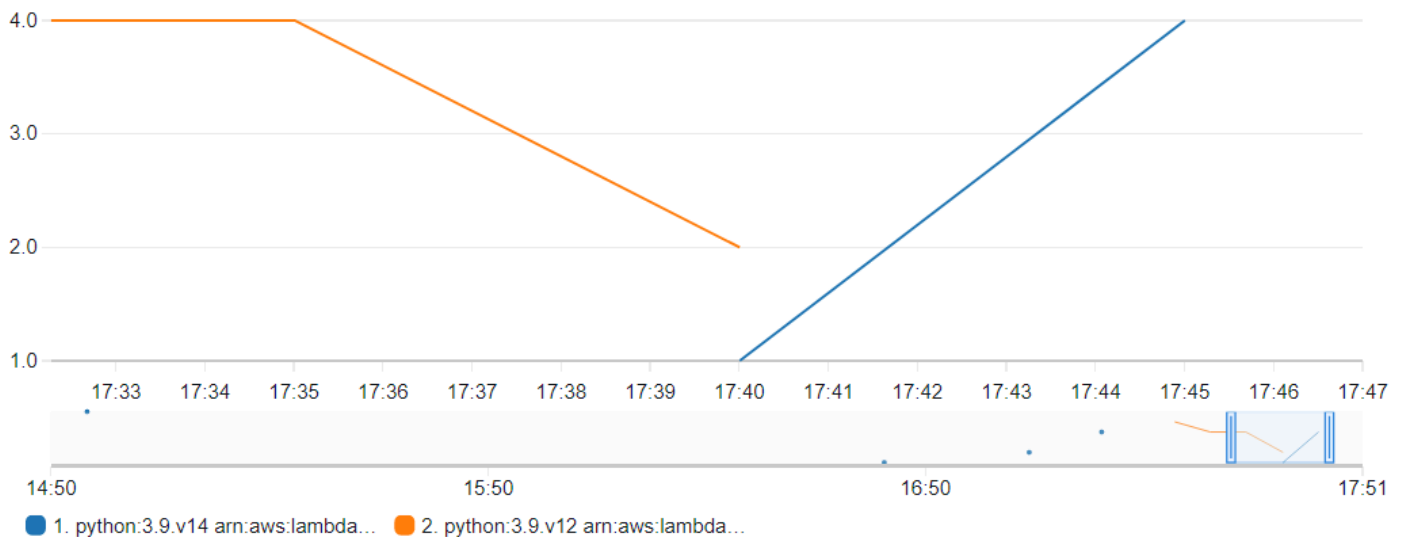
```

Il seguente rapporto di CloudWatch Contributor Insights mostra un esempio di transizione di versione in fase di esecuzione come illustrato dalla regola precedente. La linea arancione mostra l'inizializzazione dell'ambiente di esecuzione per la versione di runtime precedente (python:3.13.v12) e la linea blu mostra l'inizializzazione dell'ambiente di esecuzione per la nuova versione di runtime (python:3.13.v14).

#### Top 2 of 2 unique contributors



2 unique contributors • No unit



## Informazioni sul modello di responsabilità condivisa per la gestione del runtime Lambda

Lambda è responsabile della cura e della pubblicazione degli aggiornamenti di sicurezza per tutti i runtime gestiti e le immagini dei container supportati. La responsabilità di aggiornare le funzioni esistenti per utilizzare la versione di runtime più recente varia a seconda della modalità di aggiornamento del runtime utilizzata.

Lambda è responsabile dell'applicazione degli aggiornamenti di runtime a tutte le funzioni configurate per l'uso la modalità di aggiornamento del runtime Auto (Automatico).

Per le funzioni configurate con la modalità di aggiornamento del runtime Function update (Aggiornamento della funzione), l'utente è responsabile dell'aggiornamento regolare della funzione. Lambda è responsabile dell'applicazione degli aggiornamenti di runtime quando si effettuano tali aggiornamenti. Se non aggiorni la funzione, Lambda non aggiorna il runtime. Se non aggiorni regolarmente la tua funzione, ti consigliamo vivamente di configurarla per gli aggiornamenti automatici del runtime in modo da continuare a ricevere aggiornamenti di sicurezza.

Per le funzioni configurate per utilizzare la modalità di aggiornamento del runtime Manual (Manuale), l'utente è responsabile dell'aggiornamento della funzione per l'uso della versione di runtime più recente. Si consiglia vivamente di utilizzare questa modalità solo per ripristinare la versione di runtime come mitigazione temporanea nel raro caso di incompatibilità degli aggiornamenti del runtime. Ti consigliamo inoltre di passare alla modalità Auto (Automatico) il più rapidamente possibile per ridurre al minimo il tempo in cui le tue funzioni non vengono aggiornate.

Se [per implementare le funzioni, vengono utilizzate le immagini di container](#), Lambda è responsabile della pubblicazione delle immagini di base aggiornate. In questo caso, l'utente sarà responsabile della ricostruzione dell'immagine di container della funzione dall'immagine di base più recente e della nuova implementazione dell'immagine di container.

Ciò è riassunto nella seguente tabella:

| Deployment mode<br>(Modalità distribuzione)                              | Responsabilità di Lambda                                                                                                               | Responsabilità del cliente                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Runtime gestito, modalità Auto (Automatico)                              | <p>Pubblica nuove versioni di runtime contenenti le patch più recenti.</p> <p>Applica le patch di runtime alle funzioni esistenti.</p> | <p>Torna a una versione di runtime precedente nel raro caso in cui si verifichi un problema di compatibilità con l'aggiornamento del runtime. Segui le migliori pratiche per la <a href="#">compatibilità con le versioni precedenti</a>.</p>                                                                                                                                                                                                        |
| Runtime gestito, modalità Function update (Aggiornamento della funzione) | <p>Pubblica nuove versioni di runtime contenenti le patch più recenti.</p>                                                             | <p>Aggiorna regolarmente le funzioni per utilizzare la versione di runtime più recente.</p> <p>Passa una funzione alla modalità Auto (Automatico) quando questa non viene aggiornata regolarmente.</p> <p>Torna a una versione di runtime precedente nel raro caso in cui si verifichi un problema di compatibilità con l'aggiornamento del runtime. Segui le migliori pratiche per la compatibilità con le <a href="#">versioni precedenti</a>.</p> |
| Runtime gestito, modalità Manual (Manuale)                               | <p>Pubblica nuove versioni di runtime contenenti le patch più recenti.</p>                                                             | <p>Utilizza questa modalità solo per il rollback temporaneo del runtime nel raro caso in cui si verifichi un problema di compatibilità con gli aggiornamenti del runtime.</p> <p>Passa le funzioni alla modalità di aggiornamento Auto (Automatico) o Function update (Aggiornamento della funzione) e alla versione di runtime più recente il più presto possibile.</p>                                                                             |
| Immagine di container                                                    | <p>Pubblica nuove immagini di container contenenti le patch più recenti.</p>                                                           | <p>Implementa nuovamente le funzioni in maniera regolare utilizzando l'immagine di base del</p>                                                                                                                                                                                                                                                                                                                                                      |

| Deployment mode<br>(Modalità distribuzione) | Responsabilità di Lambda | Responsabilità del cliente                                 |
|---------------------------------------------|--------------------------|------------------------------------------------------------|
|                                             |                          | container più recente per utilizzare le patch più recenti. |

Per ulteriori informazioni sulla responsabilità condivisa con AWS, consulta [Modello di responsabilità condivisa](#).

## Controllo delle autorizzazioni di aggiornamento del runtime Lambda per applicazioni ad alta conformità

Per soddisfare i requisiti di patch, i clienti Lambda in genere si affidano agli aggiornamenti automatici del runtime. Se la tua applicazione è soggetta a severi requisiti di aggiornamento delle patch, potresti voler limitare l'uso delle versioni di runtime precedenti. Puoi limitare i controlli di gestione del runtime di Lambda utilizzando AWS Identity and Access Management (IAM) per negare agli utenti del tuo AWS account l'accesso alle operazioni API. [PutRuntimeManagementConfig](#) Questa operazione viene utilizzata per scegliere la modalità di aggiornamento del runtime per una funzione.

Se si nega l'accesso a questa operazione, tutte le funzioni passeranno automaticamente alla modalità Auto (Automatico). È possibile applicare questa limitazione in tutta l'organizzazione utilizzando una [policy di controllo dei servizi](#). Nel caso in cui sia necessario ripristinare una funzione a una versione di runtime precedente, è possibile concedere un'eccezione alla policy caso per caso.

## Recuperare dati sulle funzioni Lambda che utilizzano un runtime obsoleto

Quando un runtime Lambda si avvicina alla deprecazione, Lambda avvisa l'utente tramite e-mail e fornisce notifiche istantanee. Dashboard Health Trusted Advisor Queste e-mail e notifiche elencano le versioni \$LATEST delle funzioni che utilizzano il runtime. Per elencare tutte le versioni delle funzioni che utilizzano un determinato runtime, puoi utilizzare AWS Command Line Interface (AWS CLI) o uno degli SDK. AWS

Se disponi di un numero elevato di funzioni che utilizzano un runtime che è considerato obsoleto, puoi anche utilizzare gli AWS CLI o AWS SDK per aiutarti a dare priorità agli aggiornamenti delle funzioni richiamate più comunemente.

Fai riferimento alle sezioni seguenti per scoprire come utilizzare gli AWS SDK AWS CLI e per raccogliere dati sulle funzioni che utilizzano un determinato runtime.

### Elenco delle versioni delle funzioni che utilizzano un determinato runtime

Per utilizzare l' AWS CLI elenco di tutte le versioni delle funzioni che utilizzano un determinato runtime, esegui il comando seguente. Sostituiscilo `RUNTIME_IDENTIFIER` con il nome del runtime obsoleto e scegline uno tuo. Regione AWS Per elencare solo le versioni della funzione \$LATEST, ometti `--function-version ALL` dal comando.

```
aws lambda list-functions --function-version ALL --region us-east-1 --output text --  
query "Functions[?Runtime=='RUNTIME_IDENTIFIER'].FunctionArn"
```

#### Tip

Il comando di esempio elenca le funzioni nella `us-east-1` regione per una particolare area. Account AWS Dovrai ripetere questo comando per ogni regione in cui il tuo account ha delle funzioni e per ciascuna delle tue. Account AWS

Puoi anche elencare le funzioni che utilizzano un determinato runtime utilizzando uno degli AWS SDK. Il codice di esempio seguente utilizza V3 AWS SDK per JavaScript e AWS SDK per Python (Boto3) per restituire un elenco degli ARN della funzione per le funzioni che utilizzano un determinato runtime. Il codice di esempio restituisce anche il gruppo di CloudWatch log per ciascuna delle funzioni elencate. È possibile utilizzare questo gruppo di log per trovare la data dell'ultima invocazione per la funzione. Per ulteriori informazioni, consulta la sezione [the section called “Identificazione delle funzioni richiamate più di frequente e più di recente”](#) seguente.

## Node.js

Example JavaScript codice per elencare le funzioni utilizzando un particolare runtime

```
import { LambdaClient, ListFunctionsCommand } from "@aws-sdk/client-lambda";
const lambdaClient = new LambdaClient();

const command = new ListFunctionsCommand({
  FunctionVersion: "ALL",
  MaxItems: 50
});
const response = await lambdaClient.send(command);

for (const f of response.Functions){
  if (f.Runtime == '<your_runtime>'){ // Use the runtime id, e.g. 'nodejs24.x' or 'python3.14'
    console.log(f.FunctionArn);
    // get the CloudWatch log group of the function to
    // use later for finding the last invocation date
    console.log(f.LoggingConfig.LogGroup);
  }
}
// If your account has more functions than the specified
// MaxItems, use the returned pagination token in the
// next request with the 'Marker' parameter
if ('NextMarker' in response){
  let paginationToken = response.NextMarker;
}
```

## Python

Example Codice Python per elencare le funzioni utilizzando un particolare runtime

```
import boto3
from botocore.exceptions import ClientError

def list_lambda_functions(target_runtime):

    lambda_client = boto3.client('lambda')

    response = lambda_client.list_functions(
        FunctionVersion='ALL',
        MaxItems=50
    )
```

```
if not response['Functions']:
    print("No Lambda functions found")
else:
    for function in response['Functions']:
        if function['PackageType']=='Zip' and function['Runtime'] ==
target_runtime:
        print(function['FunctionArn'])
        # Print the CloudWatch log group of the function
        # to use later for finding last invocation date
        print(function['LoggingConfig']['LogGroup'])

    if 'NextMarker' in response:
        pagination_token = response['NextMarker']

if __name__ == "__main__":
    # Replace python3.12 with the appropriate runtime ID for your Lambda functions
    list_lambda_functions('python3.12')
```

Per saperne di più sull'utilizzo di un AWS SDK per elencare le funzioni utilizzando l'[ListFunctions](#) azione, consulta la documentazione [SDK](#) per il tuo linguaggio di programmazione preferito.

Puoi anche utilizzare la funzione Queries AWS Config avanzate per elencare tutte le tue funzioni che utilizzano un runtime interessato. Questa query restituisce solo le versioni \$LATEST della funzione, ma è possibile aggregare le query per elencare le funzioni in tutte le regioni e più Account AWS con un solo comando. Per ulteriori informazioni, consulta [Interrogare lo stato di configurazione corrente delle AWS Auto Scaling risorse](#) nella Guida per gli sviluppatori. AWS Config

## Identificazione delle funzioni richiamate più di frequente e più di recente

Se Account AWS contiene funzioni che utilizzano un runtime che è diventato obsoleto, potresti voler dare priorità all'aggiornamento delle funzioni richiamate di frequente o delle funzioni richiamate di recente.

Se disponi solo di poche funzioni, puoi utilizzare la console CloudWatch Logs per raccogliere queste informazioni esaminando i flussi di log delle tue funzioni. Per ulteriori informazioni, consulta [Visualizzare i dati di registro inviati a CloudWatch Logs](#).

Per vedere il numero di chiamate recenti di funzioni, puoi anche utilizzare le informazioni sulle CloudWatch metriche mostrate nella console Lambda. Per visualizzare queste informazioni, completa le seguenti operazioni:



1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Seleziona la funzione per la quale desideri visualizzare le statistiche sull'invocazione.
3. Selezionare la scheda Monitor (Monitora).
4. Imposta il periodo di tempo per cui desideri visualizzare le statistiche utilizzando il selettore dell'intervallo di date. Le invocazioni recenti vengono visualizzate nel riquadro Invocazioni.

Per gli account con un numero maggiore di funzioni, può essere più efficiente raccogliere questi dati a livello di codice utilizzando uno AWS CLI o uno degli AWS SDK utilizzando le azioni e API.

[DescribeLogStreams](#) [GetMetricStatistics](#)

Gli esempi seguenti forniscono frammenti di codice che utilizzano V3 AWS SDK per JavaScript e the AWS SDK per Python (Boto3) per identificare la data dell'ultimo richiamo per una particolare funzione e per determinare il numero di richiami per una particolare funzione negli ultimi 14 giorni.

Node.js

Example JavaScript codice per trovare l'ora dell'ultima chiamata per una funzione

```
import { CloudWatchLogsClient, DescribeLogStreamsCommand } from "@aws-sdk/client-cloudwatch-logs";
const cloudWatchLogsClient = new CloudWatchLogsClient();
const command = new DescribeLogStreamsCommand({
  logGroupName: '<your_log_group_name>',
  orderBy: 'LastEventTime',
  descending: true,
  limit: 1
});
try {
  const response = await cloudWatchLogsClient.send(command);
  const lastEventTimestamp = response.logStreams.length > 0 ?
    response.logStreams[0].lastEventTimestamp : null;
  // Convert the UNIX timestamp to a human-readable format for display
  const date = new Date(lastEventTimestamp).toLocaleDateString();
  const time = new Date(lastEventTimestamp).toLocaleTimeString();
  console.log(`${date} ${time}`);
} catch (e){
  console.error('Log group not found.')
}
```

## Python

Example Codice Python per trovare l'ora dell'ultima invocazione per una funzione

```
import boto3
from datetime import datetime

cloudwatch_logs_client = boto3.client('logs')

response = cloudwatch_logs_client.describe_log_streams(
    logGroupName='<your_log_group_name>',
    orderBy='LastEventTime',
    descending=True,
    limit=1
)

try:
    if len(response['logStreams']) > 0:
        last_event_timestamp = response['logStreams'][0]['lastEventTimestamp']
        print(datetime.fromtimestamp(last_event_timestamp/1000)) # Convert timestamp
        from ms to seconds
    else:
        last_event_timestamp = None
except:
    print('Log group not found')
```

### Tip

Puoi trovare il nome del gruppo di log della tua funzione utilizzando l'operazione [ListFunctions](#) API. Per un esempio su come eseguire questa operazione, consulta [the section called “Elenco delle versioni delle funzioni che utilizzano un determinato runtime”](#).

## Node.js

Example JavaScript codice per trovare il numero di chiamate negli ultimi 14 giorni

```
import { CloudWatchClient, GetMetricStatisticsCommand } from "@aws-sdk/client-cloudwatch";
const cloudWatchClient = new CloudWatchClient();
const command = new GetMetricStatisticsCommand({
```

```

    Namespace: 'AWS/Lambda',
    MetricName: 'Invocations',
    StartTime: new Date(Date.now()-86400*1000*14), // 14 days ago
    EndTime: new Date(Date.now()),
    Period: 86400 * 14, // 14 days.
    Statistics: ['Sum'],
    Dimensions: [{
        Name: 'FunctionName',
        Value: '<your_function_name>'
    }]
  });
const response = await cloudWatchClient.send(command);
const invokesInLast14Days = response.Datapoints.length > 0 ?
    response.Datapoints[0].Sum : 0;

console.log('Number of invocations: ' + invokesInLast14Days);

```

## Python

Example Codice Python per trovare il numero di invocazioni negli ultimi 14 giorni

```

import boto3
from datetime import datetime, timedelta

cloudwatch_client = boto3.client('cloudwatch')

response = cloudwatch_client.get_metric_statistics(
    Namespace='AWS/Lambda',
    MetricName='Invocations',
    Dimensions=[
        {
            'Name': 'FunctionName',
            'Value': '<your_function_name>'
        },
    ],
    StartTime=datetime.now() - timedelta(days=14),
    EndTime=datetime.now(),
    Period=86400 * 14, # 14 days
    Statistics=[
        'Sum'
    ]
)

```

```
if len(response['Datapoints']) > 0:
    invokes_in_last_14_days = int(response['Datapoints'][0]['Sum'])
else:
    invokes_in_last_14_days = 0

print(f'Number of invocations: {invokes_in_last_14_days}')
```

## Modifica dell'ambiente di runtime

È possibile utilizzare le [estensioni interne](#) per modificare il processo di runtime. Le estensioni interne non sono processi separati, sono eseguiti nell'ambito del processo di runtime.

Lambda fornisce [variabili di ambiente](#) specifiche della lingua che è possibile impostare per aggiungere opzioni e strumenti al runtime. Lambda fornisce anche [Script wrapper](#), che consentono a Lambda di delegare l'avvio del runtime allo script. È possibile creare uno script wrapper per personalizzare il comportamento di avvio del runtime.

### Language-specific variabili di ambiente

Lambda supporta metodi di sola configurazione per abilitare il precaricamento del codice durante l'inizializzazione della funzione tramite le seguenti variabili di ambiente specifiche della lingua:

- `JAVA_TOOL_OPTIONS`: su Java, Lambda supporta questa variabile di ambiente per impostare ulteriori variabili della riga di comando in Lambda. Questa variabile di ambiente consente di specificare l'inizializzazione degli strumenti, in particolare l'avvio di agenti del linguaggio di programmazione nativo o Java utilizzando le opzioni `agentlib` o `javaagent`. Per ulteriori informazioni, consulta la sezione [Variabili di ambiente `JAVA\_TOOL\_OPTIONS`](#).
- `NODE_OPTIONS`— Disponibile in [Node.js runtime](#).
- `DOTNET_STARTUP_HOOKS` – Su .NET Core 3.1 e versioni successive, questa variabile di ambiente specifica un percorso ad un assembly (dll) che Lambda può utilizzare.

L'utilizzo di variabili di ambiente specifiche della lingua è il modo preferito per impostare le proprietà di avvio.

### Script wrapper

È possibile creare uno script wrapper per personalizzare il comportamento di avvio runtime della funzione Lambda. Con uno script wrapper, è possibile impostare parametri di configurazione che non possono essere impostati tramite variabili di ambiente specifiche della lingua.

#### Note

Le chiamate potrebbero fallire se lo script wrapper non avvia correttamente il processo di runtime.

Gli script wrapper sono supportati su tutti i [runtime Lambda nativi](#). Gli script wrapper non sono supportati su [OS-only tempi di esecuzione](#) (la famiglia di runtime provided).

Quando si utilizza uno script wrapper per la funzione, Lambda avvia il runtime utilizzando lo script. Lambda invia allo script il percorso all'interprete e tutti gli argomenti originali per l'avvio del runtime standard. Lo script può estendere o trasformare il comportamento di avvio del programma. Ad esempio, lo script può iniettare e modificare argomenti, impostare variabili di ambiente o acquisire metriche, errori e altre informazioni diagnostiche.

È possibile specificare lo script impostando il valore della variabile di ambiente `AWS_LAMBDA_EXEC_WRAPPER` come percorso del file system di un file binario o di uno script eseguibile.

Esempio: creare e utilizzare uno script wrapper come layer Lambda

Nell'esempio seguente, si crea uno script wrapper per avviare l'interprete Python con l'opzione `-X importtime`. Quando si esegue la funzione, Lambda genera una voce di log per mostrare la durata del tempo di importazione per ogni importazione.

Per creare e utilizzare uno script wrapper come livello

1. Crea una directory per il livello:

```
mkdir -p python-wrapper-layer/bin  
cd python-wrapper-layer/bin
```

2. Nella bin directory, incollate il codice seguente in un nuovo file denominato `importtime_wrapper`. Questo è lo script wrapper.

```
#!/bin/bash  
  
# the path to the interpreter and all of the originally intended arguments  
args=("$@")  
  
# the extra options to pass to the interpreter  
extra_args=("-X" "importtime")  
  
# insert the extra options  
args=("${args[@]:0:$#-1}" "${extra_args[@]}" "${args[@]: -1}")  
  
# start the runtime with the extra options
```

```
exec "${args[@]}"
```

3. Assegnare autorizzazioni eseguibili a tutti gli script.

```
chmod +x importtime_wrapper
```

4. Crea un file .zip per il livello:

```
cd ..  
zip -r ../python-wrapper-layer.zip .
```

5. Verifica che il file .zip abbia la seguente struttura di directory:

```
python-wrapper-layer.zip  
# bin  
    # importtime_wrapper
```

6. [Create un livello](#) utilizzando il pacchetto.zip.
7. Crea una funzione utilizzando la console Lambda.
  - a. Aprire la [console Lambda](#).
  - b. Scegli Crea funzione.
  - c. Immetti un nome per la funzione.
  - d. Per Runtime, scegli l'ultimo runtime supportato di Python.
  - e. Scegli Crea funzione.
8. Aggiungi il livello alla tua funzione.
  - a. Scegliere la funzione, quindi scegliere Codice se non è già selezionato.
  - b. Scorri verso il basso fino alla sezione Livelli, quindi scegli Aggiungi un livello.
  - c. Per Layer source, seleziona Livelli personalizzati, quindi scegli il tuo livello dall'elenco a discesa Livelli personalizzati.
  - d. In Version (Versione), selezionare 1.
  - e. Scegliere Aggiungi.
9. Aggiungi la variabile di ambiente wrapper.
  - a. Scegli Configurazione, quindi scegli Variabili di ambiente.
  - b. ~~In Environment variables (Variabili di ambiente), scegliere Edit (Modifica)~~

- c. Scegli Add environment variable (Aggiungi variabile d'ambiente).
  - d. In Chiave, inserire `AWS_LAMBDA_EXEC_WRAPPER`.
  - e. Per Value, inserisci `/opt/bin/importtime_wrapper` (`/opt/+` la struttura delle cartelle del tuo livello.zip).
  - f. Scegli Save (Salva).
10. Provate lo script wrapper.
- a. Seleziona la scheda Test.
  - b. In Evento di prova, scegli Test. Non è necessario creare un evento di test: l'evento predefinito funziona.
  - c. Scorri verso il basso fino a Log output. Poiché lo script wrapper ha avviato l'interprete Python con l'opzione `-X importtime`, i log mostrano il tempo richiesto per ogni importazione. Ad esempio:

```

532 |          collections
import time:      63 |          63 |          _functools
import time:    1053 |        3646 |          functools
import time:    2163 |        7499 |          enum
import time:     100 |         100 |          _sre
import time:     446 |         446 |          re._constants
import time:     691 |        1136 |          re._parser
import time:     378 |         378 |          re._casefix
import time:     670 |        2283 |          re._compiler
import time:     416 |         416 |          copyreg

```

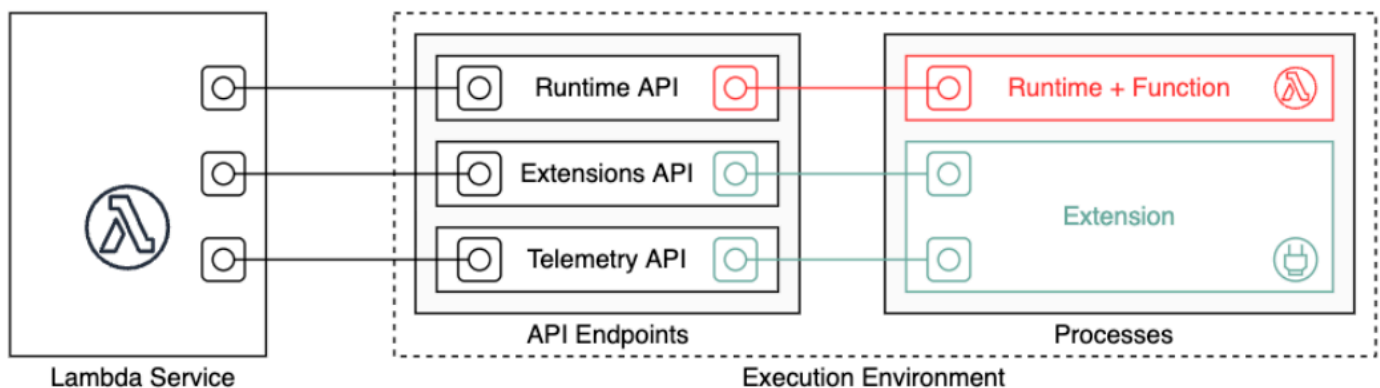


## Utilizzo dell'API di runtime Lambda per runtime personalizzati

AWS Lambda fornisce un'API HTTP per runtime [personalizzati per](#) ricevere eventi di chiamata da Lambda e inviare i dati di risposta all'interno dell'ambiente di esecuzione Lambda. [???](#) In questa sezione è riportato il riferimento API per l'API di runtime Lambda.

### Le istanze gestite Lambda supportano richieste simultanee

Le istanze gestite Lambda utilizzano la stessa API di runtime delle funzioni Lambda (impostazione predefinita). La differenza fondamentale è che le istanze gestite possono accettare `/response` richieste simultanee `/next` e fino al limite configurato. `AWS_LAMBDA_MAX_CONCURRENCY` Ciò consente di elaborare più chiamate contemporaneamente all'interno di un unico ambiente di esecuzione. Per ulteriori informazioni sulle istanze gestite, vedere. [Comprendere l'ambiente di esecuzione delle istanze gestite Lambda](#)



La specifica OpenAPI per l'API runtime versione 2018-06-01 è disponibile qui: [runtime-api.zip](#)

Per creare un URL di richiesta API, i runtime ottengono l'endpoint API dalla variabile d'ambiente `AWS_LAMBDA_RUNTIME_API`, aggiungere la versione dell'API e aggiungere il percorso della risorsa desiderato.

### Example Richiesta

```
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next"
```

### Metodi API

- [Chiamata successiva](#)
- [Risposta all'invocazione](#)
- [Errore di inizializzazione](#)
- [Errore della chiamata](#)
- [After-Restore \(applicabile solo per SnapStart\)](#)
- [Errore di ripristino \(applicabile solo per\) SnapStart](#)

## Chiamata successiva

Percorso – `/runtime/invocation/next`

Metodo – GET

Il runtime invia questo messaggio a Lambda per richiedere un evento di chiamata. Il corpo della risposta contiene il payload della chiamata che è un documento JSON contenente i dati dell'evento dal trigger della funzione. Le intestazioni della risposta contengono dati aggiuntivi sulla chiamata.

Intestazioni di risposta

- `Lambda-Runtime-Aws-Request-Id`— L'evento che ha attivato l'invocazione della funzione. Le fonti di eventi forniscono gli ID delle richieste oppure Lambda li genera automaticamente al momento dell'inserimento. Un singolo ID di richiesta potrebbe comportare più tentativi di richiamo. Utilizzalo nel percorso dell'URL quando invii la risposta o l'errore.

Ad esempio, `8476a536-e9f4-11e8-9739-2dfe598c3fcd`.

- `Lambda-Runtime-Deadline-Ms` – La data del timeout della funzione in millisecondi Unix.

Ad esempio, `1542409706888`.

- `Lambda-Runtime-Invoked-Function-Arn` – L'ARN della funzione Lambda, la versione o l'alias specificato nella chiamata.

Ad esempio, `arn:aws:lambda:us-east-2:123456789012:function:custom-runtime`.

- `Lambda-Runtime-Trace-Id` – L'[intestazione di tracciamento AWS X-Ray](#).

Ad esempio, `Root=1-5bef4de7-ad49b0e87f6ef6c87fc2e700;Parent=9a9197af755a6419;Sampled=1`.

- `Lambda-Runtime-Client-Context`— Per le chiamate da AWS Mobile SDK, dati sull'applicazione client e sul dispositivo.

- **Lambda-Runtime-Cognito-Identity**— Per le chiamate dall'SDK per AWS dispositivi mobili, dati sul provider di identità Amazon Cognito.
- **Lambda-Runtime-Invocation-Id**— Un identificatore univoco per questo tentativo di chiamata.

Non impostare un timeout sulla richiesta GET in quanto la risposta potrebbe essere ritardata. Tra il momento in cui Lambda avvia il runtime e il momento in cui il runtime ha un evento da restituire, il processo di runtime potrebbe rimanere bloccato per diversi secondi.

Un ID di richiesta (**Lambda-Runtime-Aws-Request-Id**) identifica un evento univoco. Gli ID delle richieste sono forniti dalle fonti di eventi o generati automaticamente da Lambda al momento dell'inserimento. Utilizza l'ID della richiesta nel percorso dell'URL quando invii la risposta o l'errore.

Un ID di chiamata (**Lambda-Runtime-Invocation-Id**) rappresenta un singolo tentativo di chiamata per un evento. Un singolo ID di richiesta può comportare più tentativi di chiamata, ciascuno con il proprio ID di chiamata univoco. Lambda utilizza ogni ID di chiamata esattamente una volta e non lo riutilizza mai. Riattiva questo valore e chiama. `/response` `/error` L'intestazione è opzionale per la retrocompatibilità con i runtime esistenti: la sua omissione non comporta un rifiuto. Lambda rifiuta solo `400 InvalidInvocationId` quando l'intestazione è presente, ma il suo valore non corrisponde alla chiamata attiva.

L'intestazione di traccia contiene l'ID di traccia, l'ID dell'elemento padre e la selezione per il campionamento. Se la richiesta viene campionata, la richiesta è stata campionata da Lambda o da un servizio upstream. Il runtime deve impostare `_X_AMZN_TRACE_ID` sul valore dell'intestazione. L' X-Ray SDK lo legge per ottenere gli ID e determinare se tracciare la richiesta.

## Risposta all'invocazione

Percorso – `/runtime/invocation/AwsRequestId/response`

Metodo – POST

Una volta che la funzione è stata eseguita fino al completamento, il runtime invia una risposta di chiamata a Lambda. Per le chiamate sincrone, Lambda invia la risposta al client.

### Intestazioni della richiesta

**Lambda-Runtime-Invocation-Id**— Restituisce il valore ricevuto da `/next` Lambda rifiuta la richiesta con `400 InvalidInvocationId` se il valore non corrisponde alla chiamata attiva.

## Example Richiesta con esito positivo

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
INVOCATION_ID=<value from Lambda-Runtime-Invocation-Id response header>
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/${REQUEST_ID}/response" -d "SUCCESS" --header "Lambda-Runtime-Invocation-Id: $INVOCATION_ID"
```

## Errore di inizializzazione

Se la funzione restituisce un errore o il runtime rileva un errore durante l'inizializzazione, il runtime utilizzerà questo metodo per segnalare l'errore a Lambda.

Percorso – `/runtime/init/error`

Metodo – POST

Headers

`Lambda-Runtime-Function-Error-Type`: il tipo di errore rilevato dal runtime. Questa intestazione è opzionale. Lambda accetta qualsiasi valore di stringa; consigliamo di utilizzare il formato `<Category.Reason>`, dove `Category` è `Runtime` o `Function` e `Reason` inizia con una lettera maiuscola. Ad esempio:

- `Runtime.NoSuchHandler`
- `Runtime.APIKeyNotFound`
- `Runtime.ConfigInvalid`
- `Runtime.BeforeSnapshotError(per) SnapStart`
- `Runtime.UnknownReason`

I valori che non corrispondono a questo modello vengono normalizzati in `Runtime.Unknown` o `Function.Unknown`.

Parametri corpo

`ErrorRequest` – Informazioni sull'errore. Campo obbligatorio: no.

Questo campo è un oggetto JSON con la seguente struttura:

```
{
  errorMessage: string (text description of the error),
  errorType: string,
```

```
    stackTrace: array of strings  
}
```

NB: Lambda accetta qualsiasi valore per `errorType`.

Nell'esempio seguente viene mostrato un messaggio di errore della funzione Lambda in cui la funzione non è stata in grado di analizzare i dati evento forniti nell'invocazione.

#### Example Errore di funzione

```
{  
  "errorMessage" : "Error parsing event data.",  
  "errorType" : "InvalidEventDataException",  
  "stackTrace": [ ]  
}
```

#### Parametri del corpo della risposta

- **StatusResponse** – Stringa. Informazioni sullo stato, inviate con codici di risposta 202.
- **ErrorResponse**— Informazioni aggiuntive sull'errore, inviate con i codici di risposta all'errore. **ErrorResponse** contiene un tipo di errore e un messaggio di errore.

#### Codice di risposta

- 202 – Accettato
- 403 – Non consentito
- 500 — Errore nel contenitore. Non-recoverable stato. Il runtime dovrebbe uscire tempestivamente.

#### Example Richiesta con errore di inizializzazione

```
ERROR="{\"errorMessage\" : \"Failed to load function.\", \"errorType\" :  
  \"InvalidFunctionException\"}"  
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/init/error" -d "$ERROR" --  
header "Lambda-Runtime-Function-Error-Type: Unhandled"
```

#### Errore della chiamata

Se la funzione restituisce un errore o il runtime rileva un errore, il runtime utilizzerà questo metodo per segnalare l'errore a Lambda.

Percorso – `/runtime/invoke/AwsRequestId/error`

Metodo – POST

Headers

`Lambda-Runtime-Function-Error-Type` – Tipo di errore rilevato dal runtime. Campo obbligatorio: no.

L'intestazione è costituita da un valore stringa. Lambda accetta qualsiasi stringa, ma si consiglia di utilizzare il formato `<categoria.motivo>`. Ad esempio:

- `Runtime.NoSuchHandler`
- `Runtime.APIKeyNotFound`
- `Runtime.ConfigInvalid`
- `Runtime.UnknownReason`

`Lambda-Runtime-Invocation-Id`— Restituisce il valore ricevuto da `/next`. Lambda rifiuta la richiesta con `400 InvalidInvocationId` se il valore non corrisponde alla chiamata attiva.

Parametri corpo

`ErrorRequest` – Informazioni sull'errore. Campo obbligatorio: no.

Questo campo è un oggetto JSON con la seguente struttura:

```
{
  errorMessage: string (text description of the error),
  errorType: string,
  stackTrace: array of strings
}
```

NB: Lambda accetta qualsiasi valore per `errorType`.

Nell'esempio seguente viene mostrato un messaggio di errore della funzione Lambda in cui la funzione non è stata in grado di analizzare i dati evento forniti nell'invocazione.

Example Errore di funzione

```
{
```

```
    "errorMessage" : "Error parsing event data.",
    "errorType" : "InvalidEventDataException",
    "stackTrace": [ ]
}
```

## Parametri del corpo della risposta

- **StatusResponse** – Stringa. Informazioni sullo stato, inviate con codici di risposta 202.
- **ErrorResponse**— Informazioni aggiuntive sull'errore, inviate con i codici di risposta all'errore. ErrorResponse contiene un tipo di errore e un messaggio di errore.

## Codice di risposta

- 202 – Accettato
- 400 – Richiesta non valida
- 403 – Non consentito
- 500 — Errore nel contenitore. Non-recoverable stato. Il tempo di esecuzione dovrebbe uscire tempestivamente.

## Example Richiesta con esito errato

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
ERROR="{\"errorMessage\" : \"Error parsing event data.\", \"errorType\" :
  \"InvalidEventDataException\"}"
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invoke/$REQUEST_ID/error"
-d "$ERROR" --header "Lambda-Runtime-Function-Error-Type: Unhandled"
```

## After-Restore (applicabile solo per SnapStart)

Percorso – `/runtime/restore/next`

Metodo – GET

Una volta completati gli hook pre-snapshot, il runtime chiama. GET `/runtime/restore/next`. Si tratta di una chiamata di blocco in stile iteratore, simile a `/runtime/invoke/next`, che segnala a Lambda che il runtime è pronto per l'istantanea dell'ambiente di esecuzione. La richiesta si blocca finché Lambda non ripristina l'ambiente di esecuzione da un'istantanea, quindi restituisce una risposta HTTP 200 con un corpo vuoto.

## Headers

Non sono richieste intestazioni.

## Codice di risposta

- 200 — Lambda ha ripristinato l'ambiente di esecuzione. Esegui gli hook dopo il ripristino. Il corpo della risposta è vuoto.
- 403 — Vietato. Lo stato del runtime non è consentito `/restore/next` (ad esempio, il runtime ha già chiamato `/invocation/next` o `/restore/next`).
- 404: non SnapStart è abilitato per questa funzione.
- 500 – Errore del container. L'ambiente di esecuzione è in uno stato non recuperabile. Uscire dal processo di runtime.

## Sintassi della richiesta

```
GET /2018-06-01/runtime/restore/next HTTP/1.1
Host: ${AWS_LAMBDA_RUNTIME_API}
```

## Sintassi della risposta

```
HTTP/1.1 200 OK
Content-Length: 0
```

### Note

Non impostate un socket sul lato client o un timeout di lettura su questa (o qualsiasi altra) richiesta Runtime API. Si tratta di una chiamata di blocco in stile iteratore; Lambda blocca l'ambiente di esecuzione mentre la richiesta è aperta. La richiesta può rimanere aperta per l'intera durata dell'istanza (potenzialmente giorni, settimane o più) senza che la connessione venga considerata inattiva dal servizio Lambda.

## Errore di ripristino (applicabile solo per) SnapStart

Se un hook dopo il ripristino fallisce o il runtime rileva un errore durante il ripristino, utilizza questo metodo per segnalare l'errore a Lambda. Lambda fallisce l'invocazione in corso e distrugge l'ambiente di esecuzione.



## Percorso – /runtime/restore/error

### Metodo – POST

### Headers

`Lambda-Runtime-Function-Error-Type`: il tipo di errore rilevato dal runtime. Questa intestazione è opzionale. Lambda accetta qualsiasi valore di stringa; consigliamo di utilizzare il formato `<Category.Reason>`, dove `Category` è `Runtime` o `Function` e `Reason` inizia con una lettera maiuscola (ad esempio, `Runtime.AfterRestoreError`). I valori che non corrispondono a questo modello vengono normalizzati in `Runtime.Unknown` o `Function.Unknown`.

### Codice di risposta

- 202 — Accettato. Il corpo di risposta è `{"status": "OK"}`. Il runtime dovrebbe uscire dal processo.
- 403 — Proibito. Lo stato del runtime non è consentito `/restore/error` (ad esempio, non `/restore/next` è stato chiamato).
- 404: non `SnapStart` è abilitato per questa funzione.
- 500 – Errore del container. L'ambiente di esecuzione è in uno stato non recuperabile. Uscire dal processo di runtime.

### Example Richiesta di esempio

```
POST /2018-06-01/runtime/restore/error HTTP/1.1
Host: ${AWS_LAMBDA_RUNTIME_API}
Lambda-Runtime-Function-Error-Type: Runtime.AfterRestoreError
```

### Example Esempio di risposta

```
HTTP/1.1 202 Accepted
Content-Type: application/json

{"status": "OK"}
```

## Quando usare i runtime di Lambda OS-only

Lambda fornisce runtime [gestiti](#) per Java, Python Node.js, .NET e Ruby. Per creare funzioni Lambda in un linguaggio di programmazione che non è disponibile come runtime gestito, usa un runtime (la famiglia OS-only di runtime). provided Esistono tre casi d'uso principali per i OS-only runtime:

- **Compilazione nativa anticipata (AOT):** linguaggi come Go, Rust, Swift e C++ vengono compilati in modo nativo su un file binario eseguibile, che non richiede un runtime di linguaggio dedicato. Questi linguaggi richiedono solo un ambiente di sistema operativo in cui sia possibile eseguire il file binario compilato. Puoi anche utilizzare i OS-only runtime Lambda per distribuire file binari compilati con .NET Native AOT e Java GraalVM Native Image.

È necessario includere un client dell'interfaccia di runtime nel file binario. Il client dell'interfaccia di runtime chiama [Utilizzo dell'API di runtime Lambda per runtime personalizzati](#) per recuperare le invocazioni della funzione e quindi esegue la chiamata al gestore della funzione. Lambda fornisce client di interfaccia di runtime per [Rust](#), [Go](#), [.NET Native AOT](#), [Swift \(sperimentale\)](#) e [C++ \(sperimentale\)](#). <https://github.com/aws-labs/swift-aws-lambda-runtime> <https://github.com/aws-labs/aws-lambda-cpp>

Devi compilare il file binario per un ambiente Linux e per la stessa architettura di set di istruzioni che intendi utilizzare per la funzione (x86\_64 o arm64).

- **Third-party runtime:** puoi eseguire funzioni Lambda utilizzando runtime standard come Bref per PHP. <https://bref.sh/docs/news/01-bref-1.0.html#amazon-linux-2>
- **Runtime personalizzati:** puoi creare il tuo runtime per una lingua o una versione per la quale Lambda non fornisce un runtime gestito, ad esempio 19. Node.js Per ulteriori informazioni, consulta [Creazione di un runtime personalizzato per AWS Lambda](#). Questo è il caso d'uso meno comune per i OS-only runtime.

Lambda supporta i seguenti runtime: OS-only

| Nome            | Identificatore      | Sistema operativo | Data di ritiro | Blocco creazione funzioni | Blocco aggiornamento funzioni |
|-----------------|---------------------|-------------------|----------------|---------------------------|-------------------------------|
| OS-only Runtime | provided.<br>al2023 | Amazon Linux 2023 | 30 giugno 2029 | 31 luglio 2029            | 31 agosto 2029                |

Il runtime Amazon Linux 2023 (`provided.al2023`) offre diversi vantaggi rispetto ad Amazon Linux 2, tra cui un'impronta di implementazione ridotta e versioni aggiornate di librerie come `glibc`.

Il runtime `provided.al2023` utilizza `dnf` come gestore di pacchetti invece di `yum`, che è il gestore di pacchetti predefinito in Amazon Linux 2. Per ulteriori informazioni sulle differenze tra `provided.al2023` e `provided.al2`, consulta [Introducing the Amazon Linux 2023 runtime for AWS Lambda](#) nel AWS Compute Blog.

## Creazione di un runtime personalizzato per AWS Lambda

È possibile implementare un AWS Lambda runtime in qualsiasi linguaggio di programmazione. Il runtime è un programma che esegue un metodo del gestore della funzione Lambda quando la funzione viene richiamata. Puoi includere il runtime nel pacchetto di implementazione della funzione o distribuirlo in un [livello](#). Quando crei la funzione Lambda, scegli un [OS-only runtime](#) (la famiglia `provided` di runtime).

### Note

La creazione di un runtime personalizzato è un caso d'uso avanzato. Se stai cercando informazioni sulla compilazione in un file binario nativo o sull'uso di un runtime di terze parti pronto all'uso, consulta [Quando usare i runtime di Lambda OS-only](#).

Per una procedura guidata sul processo di implementazione del runtime personalizzato, consulta [Tutorial: Creazione di un runtime personalizzato](#).

### Argomenti

- [Requisiti](#)
- [Implementazione dello streaming delle risposte in un runtime personalizzato](#)
- [Creazione di runtime personalizzati per le istanze gestite Lambda](#)

### Requisiti

I runtime personalizzati devono completare determinate attività di inizializzazione ed elaborazione. Il runtime è responsabile dell'esecuzione del codice di configurazione della funzione, della lettura del nome del gestore da una variabile di ambiente e della lettura degli eventi di richiamo dall'API del runtime Lambda. Il runtime passa i dati dell'evento al gestore della funzioni e invia la risposta dal gestore a Lambda.

## Attività di inizializzazione

Le attività di inizializzazione vengono eseguite una volta [per istanza della funzione](#) per preparare l'ambiente a gestire le chiamate.

- Recupero delle impostazioni – Leggi le variabili d'ambiente per ottenere dettagli sulla funzione e sull'ambiente.
  - `_HANDLER` – Il percorso del gestore della configurazione della funzione. Il formato standard è `file.method`, dove `file` è il nome del file senza estensione e `method` è il nome di un metodo o una funzione definita nel file.
  - `LAMBDA_TASK_ROOT` – La directory che contiene il codice della funzione.
  - `AWS_LAMBDA_RUNTIME_API` – L'host e la porta dell'API di runtime.

Per l'elenco completo delle variabili disponibili, consulta la pagina [Variabili di ambiente di runtime definite](#).

- Inizializzazione della funzione – Carica il file del gestore ed esegui il codice globale o statico che contiene. Le funzioni devono creare le risorse statiche come i client SDK e le connessioni al database e quindi riutilizzarle per più chiamate.
- Gestione degli errori – Se si verifica un errore, viene chiamata l'API [errore di inizializzazione](#) e viene chiusa la funzione.

L'inizializzazione viene calcolata nella fatturazione del runtime e del timeout. Quando un'esecuzione attiva l'inizializzazione di una nuova istanza della funzione, puoi visualizzare il tempo di inizializzazione nei log e nella [traccia AWS X-Ray](#).

### Example log

```
REPORT RequestId: f8ac1208... Init Duration: 48.26 ms    Duration: 237.17 ms    Billed
Duration: 300 ms    Memory Size: 128 MB    Max Memory Used: 26 MB
```

## Attività di elaborazione

Durante l'esecuzione, un runtime usa l'[interfaccia del runtime Lambda](#) per gestire gli eventi in entrata e segnalare gli errori. Dopo aver completato le attività di inizializzazione, il runtime elabora gli eventi in entrata in un ciclo. Nel codice di runtime, eseguire le seguenti fasi in ordine.

- Recupero di un evento – Viene invocata l'API [chiamata successiva](#) per ottenere l'evento seguente. Il corpo della risposta contiene i dati dell'evento. Le intestazioni di risposta contengono l'ID della richiesta e altre informazioni.
- Propaga l'intestazione di tracciamento: ottieni l'intestazione di X-Ray tracciamento dall'intestazione nella `Lambda-Runtime-Trace-Id` risposta API. Imposta la variabile di ambiente `_X_AMZN_TRACE_ID` locale allo stesso valore. L' X-Ray SDK utilizza questo valore per connettere i dati di traccia tra i servizi.
- Creazione di un oggetto contesto – Crea un oggetto con informazioni di contesto dalle variabili di ambiente e dalle intestazioni nella risposta dell'API.
- Richiamo del gestore della funzione – Passa l'evento e l'oggetto contesto al gestore.
- Gestione della risposta – Viene [invocata l'API risposta](#) della chiamata per pubblicare la risposta del gestore.
- Gestione degli errori – Se si verifica un errore, viene chiamata l'API [errore di chiamata](#).
- Pulizia – Rilascia le risorse inutilizzate, invia i dati ad altri servizi o esegui attività aggiuntive prima di ottenere il prossimo evento.

## Entrypoint

Il punto di ingresso di un runtime personalizzato è un file eseguibile denominato `bootstrap`. Il file di `bootstrap` può essere il runtime oppure può richiamare un altro file che crea il runtime. Se la root del pacchetto di implementazione non contiene un file denominato `bootstrap`, Lambda cerca il file nei livelli della funzione. Se il file `bootstrap` non esiste o non è eseguibile, la funzione restituisce un errore `Runtime.InvalidEntrypoint` in caso di invocazione.

Ecco un `bootstrap` file di esempio che utilizza una versione in bundle di Node.js per eseguire un JavaScript runtime in un file separato denominato `runtime.js`

## Example bootstrap

```
#!/bin/sh
cd $LAMBDA_TASK_ROOT
./node-v11.1.0-linux-x64/bin/node runtime.js
```

## Implementazione dello streaming delle risposte in un runtime personalizzato

Per le [funzioni di streaming delle risposte](#), gli endpoint `response` ed `error` manifestano un comportamento leggermente diverso che consente al runtime di trasmettere risposte parziali al client

e restituire i payload in blocchi. Per ulteriori informazioni sul comportamento specifico, consulta le seguenti risorse:

- `/runtime/invoke/RequestId/response`: propaga l'intestazione `Content-Type` dal runtime per inviarla al client. Lambda restituisce il payload della risposta in blocchi utilizzando la codifica di trasferimento in blocchi. HTTP/1.1 Per trasmettere la risposta in streaming a Lambda, il runtime deve:
  - Impostare l'intestazione `HTTP Lambda-Runtime-Function-Response-Mode` su `streaming`.
  - Imposta l'intestazione `Transfer-Encoding` su `chunked`.
  - Scrivi la risposta in conformità alla specifica di codifica del trasferimento in blocchi. HTTP/1.1
  - Chiudere la connessione sottostante dopo la corretta scrittura della risposta.
- `/runtime/invoke/RequestId/error`: il runtime può utilizzare questo endpoint per segnalare errori di funzione o di runtime a Lambda, che accetta anche l'intestazione `Transfer-Encoding`. Questo endpoint può essere chiamato solo prima che il runtime inizi a inviare una risposta alla chiamata.
- Segnala gli errori intermedi utilizzando i trailer degli errori in `/runtime/invoke/RequestId/response`: per segnalare gli errori che si verificano dopo aver iniziato a scrivere la risposta alla chiamata, il runtime può facoltativamente collegare intestazioni HTTP finali denominate `Lambda-Runtime-Function-Error-Type` e `Lambda-Runtime-Function-Error-Body`. Lambda considera questa come una risposta riuscita e inoltra i metadati degli errori che il runtime fornisce al client.

 Note

Per allegare le intestazioni finali, il runtime deve impostare il valore dell'intestazione `Trailer` all'inizio della richiesta HTTP. Questo è un requisito della specifica di codifica del trasferimento in HTTP/1.1 blocchi.

- `Lambda-Runtime-Function-Error-Type`: il tipo di errore rilevato dal runtime. L'intestazione è costituita da un valore stringa. Lambda accetta qualsiasi stringa, ma consigliamo un formato di `<category.reason>` Ad esempio, `Runtime.APIKeyNotFound`.
- `Lambda-Runtime-Function-Error-Body`— Base64-encoded informazioni sull'errore.

## Creazione di runtime personalizzati per le istanze gestite Lambda

Le istanze gestite Lambda utilizzano la stessa API di runtime delle funzioni Lambda (impostazione predefinita). Tuttavia, ci sono differenze fondamentali nel modo in cui i runtime personalizzati devono essere implementati per supportare il modello di esecuzione simultanea delle istanze gestite.

### Gestione simultanea delle richieste

La differenza principale nella creazione di runtime personalizzati per le istanze gestite è il supporto per le chiamate simultanee. A differenza delle funzioni Lambda (predefinite) in cui il runtime elabora una chiamata alla volta, le istanze gestite possono elaborare più invocazioni contemporaneamente all'interno di un unico ambiente di esecuzione.

Il tuo runtime personalizzato deve:

- Supporta **/next** richieste simultanee: il runtime può effettuare più chiamate simultanee all'[API di chiamata](#) successiva, fino al limite specificato dalla variabile di ambiente. `AWS_LAMBDA_MAX_CONCURRENCY`
- Gestione **/response** delle richieste simultanee: più chiamate possono chiamare contemporaneamente l'API di risposta alle chiamate. [???](#)
- Implementa una gestione delle richieste thread-safe: assicurati che le chiamate simultanee non interferiscano tra loro gestendo correttamente le risorse e lo stato condivisi.
- Utilizza correttamente gli ID di richiesta e gli ID di chiamata: un ID di richiesta identifica l'evento; utilizzalo nel percorso dell'URL. Un ID di chiamata rappresenta un singolo tentativo di chiamata: un singolo ID di richiesta potrebbe comportare più tentativi, ciascuno con un ID di chiamata univoco. Riattiva l'ID di chiamata e chiama. `/response /error` L'intestazione è facoltativa: la sua omissione è accettata per motivi di compatibilità con le versioni precedenti. Lambda rifiuta con solo `400 InvalidInvocationId` quando l'intestazione è presente ma il suo valore non corrisponde.

### Schema di implementazione

Un tipico modello di implementazione per i runtime di Managed Instances prevede la creazione di thread o processi di lavoro per gestire le chiamate simultanee:

1. Leggi il limite di concorrenza: al momento dell'inizializzazione, leggi la variabile di `AWS_LAMBDA_MAX_CONCURRENCY` ambiente per determinare quante chiamate simultanee supportare.

2. Crea pool di lavoratori: inizializza un pool di lavoratori (thread, processi o attività asincrone) pari al limite di concorrenza.
3. Ciclo di elaborazione dei lavoratori: ogni lavoratore in modo indipendente:
  - Chiama `/runtime/invocation/next` per ottenere un evento di chiamata
  - Richiama il gestore della funzione con i dati dell'evento
  - Pubblica la risposta a `/runtime/invocation/AwsRequestId/response`
  - Ripete il ciclo

### Ulteriori considerazioni

- Formato di registrazione: le istanze gestite supportano solo il formato di registro JSON. Assicurati che il runtime rispetti la variabile di `AWS_LAMBDA_LOG_FORMAT` ambiente e utilizzi solo il formato JSON. Per ulteriori informazioni, consulta [Configurazione dei formati di log JSON e testo normale](#).
- Risorse condivise: fai attenzione quando usi risorse condivise come la `/tmp` directory con invocazioni simultanee. Implementa meccanismi di blocco adeguati per prevenire condizioni di gara.

Per ulteriori informazioni sugli ambienti di esecuzione di Lambda Managed Instances, consulta [Comprendere l'ambiente di esecuzione delle istanze gestite Lambda](#)

## Tutorial: Creazione di un runtime personalizzato

In questo tutorial crei una funzione Lambda con un runtime personalizzato. Si inizia includendo il runtime nel pacchetto di distribuzione della funzione. Quindi lo trasferisci in un livello che gestisci in modo indipendente dalla funzione. Infine, condividi il livello del runtime con tutti aggiornando la policy delle autorizzazioni basate sulle risorse.

### Prerequisiti

Questo tutorial presuppone una certa conoscenza delle operazioni di base di Lambda e della console relativa. Se non lo si è già fatto, seguire le istruzioni riportate in [Creare una funzione Lambda con la console](#) per creare la prima funzione Lambda.

Per completare i passaggi seguenti, è necessaria la [AWS CLI versione 2](#). I comandi e l'output previsto sono elencati in blocchi separati:

```
aws --version
```



Verrà visualizzato l'output seguente:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Per i comandi lunghi viene utilizzato un carattere di escape (\) per dividere un comando su più righe.

In Linux e macOS utilizzare la propria shell e il proprio programma di gestione dei pacchetti preferiti.

#### Note

Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere una Windows-integrated versione di Ubuntu e Bash, [installa il sottosistema Windows per Linux](#). I comandi della CLI di esempio in questa guida utilizzano la formattazione Linux. Se si utilizza la CLI di Windows, i comandi che includono documenti JSON in linea dovranno essere riformattati.

È necessario un ruolo IAM per creare una funzione Lambda. Il ruolo richiede l'autorizzazione per inviare i CloudWatch log a Logs e accedere a quelli Servizi AWS utilizzati dalla funzione. Se non hai un ruolo per lo sviluppo di funzioni, creane uno ora.

Per creare un ruolo di esecuzione

1. Apri la pagina [Ruoli](#) nella console IAM.
2. Scegliere Crea ruolo.
3. Creare un ruolo con le seguenti proprietà.
  - Trusted entity (Entità attendibile – Lambda
  - Autorizzazioni — AWSLambdaBasicExecutionRole.
  - Nome ruolo – **lambda-role**.

La `AWSLambdaBasicExecutionRole` policy dispone delle autorizzazioni necessarie alla funzione per scrivere i log in Logs. CloudWatch

## Creazione di una funzione

Crea una funzione Lambda con un runtime personalizzato. Questo esempio include due file: un file bootstrap del runtime e un gestore della funzione. Entrambi sono implementati in Bash.

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir runtime-tutorial
cd runtime-tutorial
```

2. Crea un nuovo file denominato bootstrap. Questo è il runtime personalizzato.

### Example bootstrap

```
#!/bin/sh

set -euo pipefail

# Initialization - load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"

# Processing
while true
do
    HEADERS="$(mktemp)"
    # Get an event. The HTTP request will block until one is received
    EVENT_DATA=$(curl -sS -LD "$HEADERS" "http://
${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next")

    # Extract request ID by scraping response headers received above
    REQUEST_ID=$(grep -Fi Lambda-Runtime-Aws-Request-Id "$HEADERS" | tr -d
'[:space:]' | cut -d: -f2)

    # Extract invocation ID for cross-wiring protection
    INVOCATION_ID=$(grep -Fi Lambda-Runtime-Invocation-Id "$HEADERS" | tr -d
'[:space:]' | cut -d: -f2)

    # Run the handler function from the script
    RESPONSE=$((echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")

    # Send the response, echoing the invocation ID
    curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response" -d "$RESPONSE" \
```

```

    ${INVOCATION_ID:+--header "Lambda-Runtime-Invocation-Id: $INVOCATION_ID"}
done

```

Il runtime carica uno script di funzione dal pacchetto di distribuzione. Utilizza due variabili per individuare lo script. `LAMBDA_TASK_ROOT` indica il percorso da cui il pacchetto è stato estratto e `_HANDLER` include il nome dello script.

Dopo che il runtime carica lo script della funzione, utilizza l'API del runtime per recuperare un evento di chiamata da Lambda, passa l'evento al gestore e trasmette la risposta a Lambda. Per ottenere l'ID della richiesta, il runtime salva le intestazioni dalla risposta dell'API in un file temporaneo e legge l'intestazione `Lambda-Runtime-Aws-Request-Id` dal file. Inoltre legge l'`Lambda-Runtime-Invocation-Id` intestazione e la riproduce nella risposta.

#### Note

I runtime hanno responsabilità aggiuntive inclusa la gestione degli errori e forniscono al gestore le informazioni sul contesto. Per informazioni dettagliate, vedi [Requisiti](#).

3. Crea uno script per la funzione. Lo script di esempio seguente definisce una funzione del gestore che richiede i dati dell'evento, li registra in `stderr` e li restituisce.

#### Example function.sh

```

function handler () {
    EVENT_DATA=$1
    echo "$EVENT_DATA" 1>&2;
    RESPONSE="Echoing request: '$EVENT_DATA'"

    echo $RESPONSE
}

```

L'aspetto della directory `runtime-tutorial` dovrebbe essere simile al seguente:

```

runtime-tutorial
# bootstrap
# function.sh

```

4. Rendere i file eseguibili e aggiungerli ad un archive ZIP. Questo è il pacchetto di implementazione.

```
chmod 755 function.sh bootstrap
zip function.zip function.sh bootstrap
```

5. Crea una funzione denominata `bash-runtime`. Per `--role`, inserisci l'ARN del tuo [ruolo di esecuzione](#) Lambda.

```
aws lambda create-function --function-name bash-runtime \
--zip-file fileb://function.zip --handler function.handler --runtime
provided.al2023 \
--role arn:aws:iam::123456789012:role/lambda-role
```

6. Richiama la funzione.

```
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}'
response.txt --cli-binary-format raw-in-base64-out
```

L'`cli-binary-format` opzione è obbligatoria se si utilizza AWS CLI la versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Dovresti ottenere una risposta simile a questa:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
```

7. Verifica la risposta.

```
cat response.txt
```

Dovresti ottenere una risposta simile a questa:

```
Echoing request: '{"text":"Hello"}'
```

## Crea un livello

Per separare il codice del runtime dal codice della funzione, crea un livello che contenga solo il runtime. I livelli consentono di sviluppare le dipendenze della funzione in modo indipendente e possono ridurre l'utilizzo dello storage quando usi lo stesso livello con più funzioni. Per ulteriori informazioni, consulta [Gestione delle dipendenze Lambda con i livelli](#).

1. Crea un file .zip contenente il file bootstrap.

```
zip runtime.zip bootstrap
```

2. Crea un livello con il comando [publish-layer-version](#).

```
aws lambda publish-layer-version --layer-name bash-runtime --zip-file fileb://  
runtime.zip
```

In tal modo viene creata la prima versione del livello.

## Aggiorna la funzione

Per utilizzare il livello del runtime nella funzione, configura la funzione affinché utilizzi il livello e rimuovi il codice del runtime dalla funzione.

1. Aggiorna la configurazione della funzione da inserire nel livello.

```
aws lambda update-function-configuration --function-name bash-runtime \  
--layers arn:aws:lambda:us-east-1:123456789012:layer:bash-runtime:1
```

Questo aggiunge il runtime alla funzione nella directory /opt. Per garantire che Lambda utilizzi il runtime nel livello, è necessario rimuovere il bootstrap dal pacchetto di implementazione della funzione, come illustrato nei due passaggi successivi.

2. Crea un file .zip contenente il codice della funzione.

```
zip function-only.zip function.sh
```

3. Aggiorna il codice della funzione in modo da includere soltanto lo script del gestore.

```
aws lambda update-function-code --function-name bash-runtime --zip-file fileb://  
function-only.zip
```

4. Chiama la funzione per verificare che funzioni con il livello del runtime.

```
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}'  
response.txt --cli-binary-format raw-in-base64-out
```

L'`cli-binary-format` opzione è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Dovresti ottenere una risposta simile a questa:

```
{  
  "StatusCode": 200,  
  "ExecutedVersion": "$LATEST"  
}
```

5. Verifica la risposta.

```
cat response.txt
```

Dovresti ottenere una risposta simile a questa:

```
Echoing request: '{"text":"Hello"}'
```

## Aggiorna il runtime

1. Per registrare le informazioni sull'ambiente di esecuzione, aggiorna lo script del runtime sulle variabili di ambiente di output.

### Example bootstrap

```
#!/bin/sh  
  
set -euo pipefail  
  
# Configure runtime to output environment variables  
echo "## Environment variables:"  
env
```

```
# Load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"

# Processing
while true
do
    HEADERS="$(mktemp)"
    # Get an event. The HTTP request will block until one is received
    EVENT_DATA=$(curl -sS -LD "$HEADERS" "http://
${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next")

    # Extract request ID by scraping response headers received above
    REQUEST_ID=$(grep -Fi Lambda-Request-Id "$HEADERS" | tr -d
'[:space:]' | cut -d: -f2)

    # Extract invocation ID for cross-wiring protection
    INVOCATION_ID=$(grep -Fi Lambda-Invocation-Id "$HEADERS" | tr -d
'[:space:]' | cut -d: -f2)

    # Run the handler function from the script
    RESPONSE=$(echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")

    # Send the response, echoing the invocation ID
    curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response" -d "$RESPONSE" \
    ${INVOCATION_ID:+--header "Lambda-Invocation-Id: $INVOCATION_ID"}
done
```

2. Crea un file .zip contenente la nuova versione del file bootstrap.

```
zip runtime.zip bootstrap
```

3. Crea una nuova versione del livello bash-runtime.

```
aws lambda publish-layer-version --layer-name bash-runtime --zip-file fileb://
runtime.zip
```

4. Configura la funzione per utilizzare la nuova versione del livello.

```
aws lambda update-function-configuration --function-name bash-runtime \
--layers arn:aws:lambda:us-east-1:123456789012:layer:bash-runtime:2
```

## Condividi il livello

Per condividere un layer con un altro Account AWS, aggiungi una dichiarazione delle autorizzazioni tra account alla policy basata sulle [risorse del layer](#). Esegui il comando [add-layer-version-permission](#) e specifica l'ID dell'account come `principal`. In ogni istruzione, puoi concedere l'autorizzazione a un singolo account, a tutti gli account o a un'organizzazione in [AWS Organizations](#).

L'esempio seguente concede all'account 111122223333 l'accesso alla versione 2 del livello `bash-runtime`.

```
aws lambda add-layer-version-permission \
  --layer-name bash-runtime \
  --version-number 2 \
  --statement-id xaccount \
  --action lambda:GetLayerVersion \
  --principal 111122223333 \
  --output text
```

Verrà visualizzato un output simile al seguente:

```
{"Sid": "xaccount", "Effect": "Allow", "Principal":
{"AWS": "arn:aws:iam::111122223333:root"}, "Action": "lambda:GetLayerVersion", "Resource": "arn:aws:
east-1:123456789012:layer:bash-runtime:2"}
```

Le autorizzazioni si applicano solo a un'unica versione di un livello. Ripeti la procedura ogni volta che crei la nuova versione di un livello.

## Eliminazione

Eliminare ciascuna versione del livello.

```
aws lambda delete-layer-version --layer-name bash-runtime --version-number 1
aws lambda delete-layer-version --layer-name bash-runtime --version-number 2
```

Poiché contiene un riferimento alla versione 2 del livello, la funzione è ancora presente in Lambda. Continua a operare, ma le funzioni non possono più essere configurate per utilizzare la versione eliminata. Se modifichi l'elenco dei livelli sulla funzione, devi specificare una nuova versione oppure omettere il livello eliminato.

Elimina la funzione con il comando [delete-function](#).



```
aws lambda delete-function --function-name bash-runtime
```

## Repository open source

AWS Lambda fornisce una varietà di strumenti, librerie e componenti open source per aiutarti a creare, personalizzare e ottimizzare le tue applicazioni serverless. Queste risorse includono client di interfaccia di runtime, librerie di eventi, immagini di base per contenitori, strumenti di sviluppo e progetti di esempio gestiti AWS e disponibili su GitHub. Utilizzando questi repository open source, puoi estendere le funzionalità di Lambda, creare runtime personalizzati, elaborare eventi da vari AWS servizi e ottenere informazioni più approfondite sulle prestazioni della tua funzione. Questa pagina fornisce una panoramica dei principali progetti open source che supportano lo sviluppo di Lambda.

### Client Runtime Interface

I Lambda Runtime Interface Clients (RIC) sono librerie open source che implementano l'API [Runtime](#) e gestiscono l'interazione tra il codice funzione e il servizio Lambda. Questi client gestiscono la ricezione degli eventi di chiamata, la trasmissione di informazioni contestuali e la segnalazione degli errori.

I client dell'interfaccia di runtime utilizzati dai runtime gestiti da Lambda e dalle immagini di base dei container sono pubblicati come open source. Quando crei runtime personalizzati o estendi quelli esistenti, puoi utilizzare queste librerie open source per semplificare l'implementazione. I seguenti GitHub repository open source contengono il codice sorgente per i RIC di Lambda:

- [Node.js Client Runtime Interface](#)
- [Client di interfaccia di runtime per Python](#)
- [Client di interfaccia di runtime per Java](#)
- [Client dell'interfaccia runtime per Ruby](#)
- [Client dell'interfaccia runtime .NET](#)
- [Client di interfaccia Rust Runtime](#)
- [Client Go Runtime Interface](#)
- [Client Swift Runtime Interface](#) (sperimentale)
- [Client C++ Runtime Interface](#) (sperimentale)
- [Immagini Lambda Base](#)

Per ulteriori informazioni sull'utilizzo di questi client per creare runtime personalizzati, consulta [the section called “Compilazione di un runtime personalizzato”](#)

## Librerie di eventi

Le librerie di eventi Lambda forniscono definizioni di tipo e utilità di supporto per l'elaborazione di eventi provenienti da vari servizi. AWS Queste librerie consentono di analizzare e gestire i dati degli eventi in modo tipo-safe, semplificando l'utilizzo di eventi provenienti da servizi come Amazon S3, Amazon DynamoDB e Amazon API Gateway.

Per i linguaggi compilati, fornisce le seguenti librerie di eventi: AWS

- [Libreria di eventi Java](#)
- [Librerie di eventi.NET](#)
- [Libreria eventi Go](#)
- [Libreria di eventi Rust](#)

Per linguaggi interpretati come Node.js Python e Ruby, gli eventi possono essere analizzati direttamente come oggetti JSON senza richiedere una libreria separata. Tuttavia, gli sviluppatori che utilizzano Node.js Python possono utilizzare powertools for AWS Lambda, che fornisce schemi integrati per AWS eventi che offrono suggerimenti sui tipi, convalida dei dati e funzionalità simili a quelle fornite dalle librerie di linguaggi compilate.

- [Powertools per TypeScript](#)
- [Powertools per Python](#)

## Immagini di base dei contenitori

AWS fornisce immagini di base di container open source che puoi utilizzare come punto di partenza per creare immagini di container per le tue funzioni Lambda. Queste immagini di base includono il client dell'interfaccia di runtime e altri componenti necessari per eseguire le funzioni nell'ambiente di esecuzione Lambda.

Per ulteriori informazioni sulle immagini di base disponibili e su come utilizzarle, consulta il [repository AWS Lambda Base Images](#) e [the section called “Immagini di container”](#)

## Strumenti di sviluppo

AWS fornisce strumenti di sviluppo open source aggiuntivi per aiutarti a creare e ottimizzare le tue funzioni Lambda:

### Powertools per AWS Lambda

Powertools for AWS Lambda semplifica lo sviluppo senza server con utilità essenziali per prevenire l'elaborazione duplicata e l'elaborazione in batch per la gestione di più record e la libreria di consumo Kafka. Queste funzionalità consentono di ridurre al minimo la complessità del codice e il sovraccarico operativo.

Puoi anche utilizzare la convalida dello schema degli eventi, la registrazione e la traccia strutturate integrate e l'integrazione dell'archivio dei parametri, progettate per accelerare la creazione di funzioni Lambda pronte per la produzione seguendo al contempo best practice ben progettate. AWS

GitHub archivi:

- [Python](#)
- [TypeScript](#)
- [Java](#)
- [.NET](#)

### strumenti di sviluppo Java

- [Java Profiler \(sperimentale\)](#): uno strumento per la profilazione delle funzioni Java Lambda.
- [Librerie Java](#): un repository che contiene una raccolta completa di librerie e strumenti Java per lo sviluppo Lambda, inclusi progetti chiave come le utilità di test JUnit e gli strumenti di profilazione.
- [Serverless Java Container](#): una libreria che puoi utilizzare per eseguire applicazioni Java esistenti su Lambda con modifiche minime.

### Strumenti di sviluppo.NET

Il [repository](#) AWS Lambda .NET fornisce librerie e strumenti .NET per lo sviluppo Lambda, inclusi progetti chiave come AWS Lambda strumenti per la CLI .NET e il server .NET Core per l'hosting di applicazioni .NET Core.

## Progetti di esempio

Esplora una raccolta completa di esempi di progetti e applicazioni Lambda nei repository [Serverless Land](#). Questi esempi illustrano vari casi d'uso, modelli di integrazione e best practice di Lambda per aiutarti a iniziare con le tue applicazioni serverless.

## Configurazione AWS Lambda funzioni

Scopri come configurare le funzionalità e le opzioni principali per la funzione Lambda tramite l'API o la console Lambda.

### [archivi di file.zip](#)

Crea un pacchetto di distribuzione delle funzioni Lambda quando desideri includere dipendenze, livelli di runtime personalizzati o qualsiasi file oltre al codice della funzione. Il pacchetto di distribuzione è un archivio di file .zip contenente il codice della funzione e le dipendenze.

### [Immagini di container](#)

Usa le immagini dei contenitori per impacchettare il codice della funzione e le dipendenze quando hai bisogno di un maggiore controllo sul processo di compilazione o se la tua funzione richiede configurazioni di runtime personalizzate. Puoi creare, testare e distribuire funzioni Lambda come immagini di contenitori utilizzando strumenti come Docker CLI.

### [Self-managed Archiviazione del codice S3](#)

Configura le tue funzioni e i tuoi livelli per utilizzare il tuo bucket S3 per l'archiviazione del codice. Lambda fa riferimento al codice direttamente dal bucket S3, eliminando i limiti di Lambda-managed archiviazione e riducendo i tempi di primo richiamo.

### [Memoria](#)

Scopri come e quando aumentare la memoria della funzione.

### [Archiviazione temporanea](#)

Scopri come e quando aumentare la capacità di archiviazione temporanea della tua funzione.

### [Timeout](#)

Scopri come e quando aumentare il valore di timeout della funzione.

## [Variabili d'ambiente](#)

È possibile rendere il codice della funzione portabile e nascondere i segreti del codice archiviandoli nella configurazione della funzione utilizzando le variabili di ambiente.

## [Rete in uscita](#)

Puoi usare la tua funzione Lambda con le AWS risorse in un Amazon VPC. La connessione della tua funzione a un VPC ti consente di accedere alle risorse in una sottorete privata, come database relazionali e cache.

## [Rete in entrata](#)

È possibile utilizzare un endpoint VPC dell'interfaccia per richiamare le funzioni Lambda senza attraversare l'Internet pubblico.

## [File system](#)

È possibile utilizzare la funzione Lambda per montare un Amazon EFS in una directory locale. Un file system consente al codice della funzione di accedere e modificare le risorse condivise in modo sicuro e con una simultaneità elevata.

## [Alias](#)

Quindi è possibile configurare i client per invocare una versione della funzione Lambda specifica utilizzando un alias anziché aggiornando il client.

## [Versioni](#)

Pubblicando una versione della funzione, sarà possibile archiviare il codice e la configurazione come risorse separate che non possono essere modificate.

## [Tag](#)

Usa i tag per abilitare il controllo degli accessi basato sugli attributi (ABAC), organizzare le tue funzioni Lambda e filtrare e generare report sulle tue funzioni utilizzando i servizi di fatturazione e gestione dei costi. AWS Cost Explorer AWS

## [Streaming delle risposte](#)

È possibile configurare gli URL delle funzioni Lambda per trasmettere i payload di risposta ai client. Lo streaming delle risposte può favorire le applicazioni sensibili alla latenza migliorando le prestazioni del time to first byte (TTFB). Questo perché consente di inviare risposte parziali al client non appena diventano disponibili. Inoltre, lo streaming delle risposte permette di creare funzioni che restituiscono payload più grandi.

## Endpoint dei metadati

Usa l'endpoint di metadati Lambda per scoprire in quale zona di disponibilità è in esecuzione la tua funzione, consentendoti di ottimizzare la latenza indirizzando le risorse della stessa AZ e di implementare modelli di resilienza. AZ-aware

## Distribuzione di funzioni Lambda come archivi di file .zip

Quando si crea una funzione Lambda, viene utilizzato un pacchetto di implementazione per impacchettare il codice della funzione. Lambda supporta due tipi di pacchetti di implementazione: immagini di container e archivi di file .zip. Il flusso di lavoro per creare una funzione dipende dal tipo di pacchetto di implementazione. Per creare una funzione definita come immagine di container, consulta [the section called “Immagini di container”](#).

Puoi utilizzare la console Lambda e l'API Lambda per creare una funzione definita con un archivio di file .zip. È inoltre possibile caricare un file .zip aggiornato per modificare il codice funzione.

Per impostazione predefinita, quando carichi un file .zip, Lambda archivia una copia del codice sorgente in Lambda-managed memoria. Con [l'archiviazione del codice S3 autogestita](#), Lambda legge il codice sorgente dal bucket S3 durante le operazioni di creazione e aggiornamento delle funzioni e non memorizza una copia separata del codice sorgente. Lambda crea comunque una rappresentazione interna ottimizzata del codice per l'invocazione. Il codice sorgente in S3 non è accessibile ad ogni avvio a freddo.

### Note

Non è possibile modificare il [tipo di pacchetto di implementazione](#) (.zip o immagine di container) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio di file .zip. È necessario creare una nuova funzione.

### Argomenti

- [Creazione della funzione](#)
- [Utilizzo dell'editor di codice della console](#)
- [Aggiornamento del codice della funzione](#)
- [Modifica del runtime](#)
- [Modifica dell'architettura](#)
- [Utilizzo dell'API Lambda](#)
- [Download del codice della funzione](#)
- [CloudFormation](#)
- [Crittografia dei pacchetti di implementazione .zip per Lambda](#)

## Creazione della funzione

Quando si crea una funzione definita con un archivio di file .zip, si scelgono un modello di codice, la versione della lingua e il ruolo di esecuzione per la funzione. Si aggiunge il codice della funzione dopo che Lambda ha creato la funzione.

### Creazione della funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli Crea funzione.
3. Scegliere Author from scratch (Crea da zero) o Use a blueprint (Usa un piano) per creare la funzione
4. In Basic information (Informazioni di base) eseguire queste operazioni:
  - a. In Function name (Nome funzione), immettere il nome della funzione. I nomi delle funzioni hanno un limite di lunghezza di 64 caratteri.
  - b. Per Runtime, scegliere la versione della lingua da utilizzare per la funzione.
  - c. (Opzionale) Per Architecture (Architettura), scegli l'architettura del set di istruzioni da utilizzare per la funzione. L'architettura predefinita è x86\_64. Quando crei il pacchetto di implementazione per la tua funzione, assicurati che sia compatibile con questa [architettura del set di istruzioni](#).
5. (Opzionale) In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito. Puoi creare un nuovo ruolo di esecuzione o utilizzare un ruolo esistente.
6. (Facoltativo) Espandi Advanced settings (Impostazioni avanzate). È possibile scegliere una Code signing configuration (Configurazione della firma del codice) per la funzione. Puoi anche configurare un (Amazon VPC) per accedere alla funzione.
7. Scegli Crea funzione.

Lambda crea la nuova funzione. È ora possibile utilizzare la console per aggiungere il codice della funzione e configurare altri parametri di funzione e funzionalità. Per le istruzioni di distribuzione del codice, consulta la pagina del gestore del runtime utilizzato dalla funzione.

### Node.js

[Distribuisci funzioni Node.js Lambda con archivi di file .zip](#)



## Python

[Utilizzo di archivi di file .zip per le funzioni Lambda in Python](#)

## Ruby

[Distribuire le funzioni Ruby Lambda con gli archivi di file .zip](#)

## Java

[Distribuisce funzioni Lambda per Java con archivi di file .zip o JAR](#)

## Go

[Distribuisce funzioni Lambda per Go con gli archivi di file .zip](#)

## C#

[Crea e implementa le funzioni Lambda C# con gli archivi di file .zip](#)

## PowerShell

[Distribuzione delle funzioni Lambda di PowerShell con gli archivi di file .zip](#)

## Utilizzo dell'editor di codice della console

La console crea una funzione Lambda con un unico file di origine. Per i linguaggi di scripting, è possibile modificare questo file e aggiungere altri file con l'editor di codice incorporato. Per salvare le modifiche, scegliere Save (Salva). Quindi, per eseguire il codice, scegliere Test (Testa).

Quando si salva il codice funzione, la console Lambda crea un pacchetto di implementazione dell'archivio di file .zip. Quando sviluppi il codice funzione al di fuori della console (utilizzando un IDE) devi [creare un pacchetto di implementazione](#) per caricare il codice nella funzione Lambda.

## Aggiornamento del codice della funzione

Per i linguaggi di scripting (Node.jsPython e Ruby), puoi modificare il codice della funzione nell'editor di codice incorporato. Se il codice supera i 3 MB, o se è necessario aggiungere librerie, o per linguaggi che l'editor non supporta (come Java, Go e C#), è necessario caricare il codice funzione come archivio .zip. Se l'archivio di file .zip è inferiore a 50 MB, è possibile caricare l'archivio di file .zip direttamente dal computer locale. Se le dimensioni dell'archivio .zip sono maggiori di 50 MB, carica il file sulla funzione da un bucket Amazon S3.

Per caricare il codice funzione come archivio .zip

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere la funzione da aggiornare e scegliere la scheda Codice
3. In Code source (Origine codice), scegli Upload from (Carica da).
4. Scegli .zip file, quindi scegli Upload (Carica).
  - Nella finestra di selezione dei file, selezionare la nuova versione dell'immagine e scegliere Open (Apri), quindi scegliere Save (Salva).
5. (Alternativa al passaggio 4) Scegli Percorso Amazon S3.
  - a. Nella casella di testo, inserisci l'URL del link S3 dell'archivio di file .zip.
  - b. In modalità di archiviazione del codice, scegli una delle seguenti opzioni:
    - Modalità copia (impostazione predefinita)
    - Modalità di riferimento
  - c. Scegli Save (Salva).

Quando si aggiorna il codice funzionale da un bucket Amazon S3 utilizzando il AWS CLI, è possibile specificare la modalità di archiviazione del codice. Impostato S3ObjectStorageMode per REFERENCE utilizzare lo storage di codice S3 [autogestito o COPY \(impostazione predefinita\) per utilizzare lo storage](#). Lambda-managed È possibile modificare la modalità di archiviazione per ogni aggiornamento.

```
aws lambda update-function-code \  
  --function-name my-function \  
  --s3-bucket my-bucket \  
  --s3-key my-function.zip \  
  --s3-object-version abc123 \  
  --s3-object-storage-mode REFERENCE
```

## Modifica del runtime

Se si aggiorna la configurazione della funzione per utilizzare un nuovo runtime, potrebbe essere necessario aggiornare il codice della funzione per renderlo compatibile con il nuovo runtime. Se si aggiorna la configurazione della funzione per utilizzare un runtime diverso, è necessario fornire un nuovo codice funzione compatibile con il runtime e l'architettura. Per istruzioni su come creare

un pacchetto di implementazione per il codice della funzione, consulta la pagina del gestore per il runtime utilizzato dalla funzione.

Le immagini di base Node.js 2.0, Python 3.12, Java 21, .NET 8, Ruby 3.3 e versioni successive si basano sull'immagine minima del contenitore di Amazon Linux 2023. Le immagini di base precedenti utilizzano Amazon Linux 2. AL2023 offre diversi vantaggi rispetto ad Amazon Linux 2, tra cui un'impronta di implementazione ridotta e versioni aggiornate di librerie come glibc. Per ulteriori informazioni, consulta il post [Introducing the Amazon Linux 2023 runtime for AWS Lambda](#) del blog AWS Compute.

## Modifica del runtime

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere la funzione da aggiornare e scegliere la scheda Codice
3. Scorrere fino alla sezione Runtime settings (Impostazioni runtime) sotto l'editor di codice.
4. Scegli Modifica.
  - a. Per Runtime, seleziona l'identificatore di runtime.
  - b. In Gestore, specifica il nome del file e il gestore per la funzione.
  - c. Per Architecture (Architettura), scegli l'architettura del set di istruzioni da utilizzare per la funzione.
5. Scegli Save (Salva).

## Modifica dell'architettura

Prima di poter modificare l'architettura del set di istruzioni, è necessario assicurarsi che il codice della funzione sia compatibile con l'architettura di destinazione.

Se usi Node.js Python o Ruby e modifichi il codice della funzione nell'editor incorporato, il codice esistente potrebbe essere eseguito senza modifiche.

Tuttavia, se si fornisce il codice della funzione utilizzando un pacchetto di implementazione con un archivio di file .zip, è necessario preparare un nuovo archivio di file .zip compilato e sviluppato correttamente per il runtime e l'architettura del set di istruzioni di destinazione. Per le istruzioni, consulta la pagina del gestore del runtime della funzione.

Per modificare l'architettura del set di istruzioni

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere la funzione da aggiornare e scegliere la scheda Codice
3. In Impostazioni runtime, scegliere Modifica.
4. Per Architecture (Architettura), scegli l'architettura del set di istruzioni da utilizzare per la funzione.
5. Scegli Save (Salva).

## Utilizzo dell'API Lambda

Per creare e configurare una funzione che utilizza un archivio di file .zip, utilizzare le seguenti operazioni API:

- [CreateFunction](#)
- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)

## Download del codice della funzione

Puoi scaricare l'attuale versione non pubblicata (\$LATEST) del tuo codice funzione .zip dalla console Lambda. Per fare ciò, assicurati innanzitutto di disporre delle seguenti autorizzazioni IAM:

- iam:GetPolicy
- iam:GetPolicyVersion
- iam:GetRole
- iam:GetRolePolicy
- iam:ListAttachedRolePolicies
- iam:ListRolePolicies
- iam:ListRoles

Per scaricare il file .zip del codice della funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione per la quale desideri scaricare il file .zip codice della funzione.

3. Nella panoramica delle funzioni, scegli il pulsante Scarica, quindi Scarica il file .zip del codice della funzione.
  - In alternativa, scegli Scarica AWS SAM file per generare e scaricare un modello SAM in base alla configurazione della tua funzione. Puoi anche scegliere Scarica entrambi per scaricare sia il file .zip che il modello SAM.

## CloudFormation

Puoi utilizzarlo CloudFormation per creare una funzione Lambda che utilizza un archivio di file .zip. Nel CloudFormation modello, la `AWS::Lambda::Function` risorsa specifica la funzione Lambda. Per le descrizioni delle proprietà della `AWS::Lambda::Function` risorsa, vedi [AWS::Lambda::Function](#) nella Guida per l'utente. AWS CloudFormation

Nella risorsa `AWS::Lambda::Function`, impostare le seguenti proprietà per creare una funzione definita come archivio di file .zip:

- `AWS::Lambda::Funzione`
  - `PackageType` — Impostato su. Zip
  - `Codice`: inserisci il nome del bucket Amazon S3 e il nome del file .zip nei campi `S3Bucket` e `S3Key`. Per Node.js Python, puoi fornire il codice sorgente in linea della tua funzione Lambda.
  - `Runtime`: imposta il valore di runtime.
  - `Architettura`: imposta il valore dell'architettura su per utilizzare il processore arm64 Graviton2. AWS Per impostazione predefinita, il valore dell'architettura è x86\_64.
  - `S3ObjectStorageMode` — (Facoltativo) Impostato per utilizzare l'archiviazione del REFERENCE codice S3 [autogestita](#). Impostato su COPY (impostazione predefinita) per utilizzare l' Lambda-managed archiviazione.

## Crittografia dei pacchetti di implementazione .zip per Lambda

AWS Lambda fornisce sempre la crittografia lato server a riposo per i pacchetti di distribuzione .zip e i dettagli di configurazione delle funzioni con un. AWS KMS key Per impostazione predefinita, Lambda utilizza un [Chiave di proprietà di AWS](#). Se questo comportamento predefinito è adatto al tuo flusso di lavoro, non devi configurare nient'altro. AWS l'utilizzo di questa chiave non comporta alcun costo.

Se preferisci, puoi invece fornire una chiave gestita AWS KMS dal cliente. È possibile eseguire questa operazione per avere il controllo sulla rotazione della chiave KMS o per soddisfare i requisiti

dell'organizzazione per la gestione delle chiavi KMS. Quando si utilizza una chiave gestita dal cliente, solo gli utenti del tuo account con accesso alla chiave KMS possono visualizzare o gestire il codice o la configurazione della funzione.

Le chiavi gestite dal cliente sono soggette a AWS KMS costi standard. Per ulteriori informazioni, consultare [Prezzi di AWS Key Management Service](#).

## Creazione di una chiave gestita dal cliente

È possibile creare una chiave simmetrica gestita dal cliente utilizzando le Console di gestione AWS, o le API. AWS KMS

Per creare una chiave simmetrica gestita dal cliente

Segui le fasi descritte in [Creazione delle chiavi KMS per la crittografia simmetrica](#) nella Guida per gli sviluppatori di AWS Key Management Service .

## Permissions

### Policy della chiave

Le [policy della chiave](#) controllano l'accesso alla chiave gestita dal cliente. Ogni chiave gestita dal cliente deve avere esattamente una policy della chiave, che contiene istruzioni che determinano chi può usare la chiave e come la possono usare. Per ulteriori informazioni, consulta [Come modificare una policy delle chiavi](#) nella Guida per gli sviluppatori di AWS Key Management Service .

Quando si utilizza una chiave gestita dal cliente per crittografare un pacchetto di implementazione .zip, Lambda non aggiunge alcuna [concessione](#) alla chiave. La tua policy AWS KMS chiave deve invece consentire a Lambda di chiamare le seguenti operazioni AWS KMS API per tuo conto:

- [km: GenerateDataKey](#)
- [kms:Decrypt](#)

Il seguente esempio di policy chiave consente a tutte le funzioni Lambda nell'account 111122223333 di richiamare AWS KMS le operazioni richieste per la chiave gestita dal cliente specificata:

## Example AWS KMS politica chiave

### JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-1:111122223333:key/key-id",
      "Condition": {
        "StringLike": {
          "kms:EncryptionContext:aws:lambda:FunctionArn":
            "arn:aws:lambda:us-east-1:111122223333:function:*"
        }
      }
    }
  ]
}
```

Per informazioni sulla [Risoluzione dei problemi delle chiavi di accesso](#) consulta la Guida per gli sviluppatori di AWS Key Management Service .

### Autorizzazioni del principale

Quando si utilizza una chiave gestita dal cliente per crittografare un pacchetto di implementazione .zip, solo i [principali](#) che hanno accesso a quella chiave possono accedere al pacchetto di implementazione .zip. Ad esempio, i committenti che non hanno accesso alla chiave gestita dal cliente non possono scaricare il pacchetto .zip utilizzando l'URL S3 prefirato incluso nella risposta. [GetFunction](#) AccessDeniedException viene restituito nella sessione Code della risposta.

## Example AWS KMS AccessDeniedException

```
{
  "Code": {
    "RepositoryType": "S3",
    "Error": {
      "ErrorCode": "AccessDeniedException",
      "Message": "KMS access is denied. Check your KMS permissions. KMS
Exception: AccessDeniedException KMS Message: User: arn:aws:sts::111122223333:assumed-
role/LambdaTestRole/session is not authorized to perform: kms:Decrypt on resource:
arn:aws:kms:us-east-1:111122223333:key/key-id with an explicit deny in a resource-
based policy"
    },
    "SourceKMSKeyArn": "arn:aws:kms:us-east-1:111122223333:key/key-id"
  },
  ...
}
```

Per ulteriori informazioni sulle autorizzazioni per le AWS KMS chiavi, consulta [Autenticazione e controllo degli accessi per. AWS KMS](#)

Utilizzo di una chiave gestita dal cliente per il pacchetto di implementazione .zip

Utilizza i seguenti parametri API per configurare le chiavi gestite dal cliente per i pacchetti di implementazione .zip:

- [SourceKMSKeyArn](#): crittografa il pacchetto di distribuzione .zip di origine (il file che carichi).
- [KMSKeyArn](#): crittografa le variabili [di](#) ambiente e le istantanee Lambda. [SnapStart](#)
- [DurableConfig.KMSKeyArn](#): [crittografa i dati di esecuzione durevoli](#). Questa chiave è indipendente dal livello di funzioneKMSKeyArn.

Quando SourceKMSKeyArn e KMSKeyArn sono specificati entrambi, Lambda utilizza la chiave KMSKeyArn per crittografare la versione decompressa del pacchetto utilizzata da Lambda per richiamare la funzione. Quando SourceKMSKeyArn è specificato ma KMSKeyArn non lo è, Lambda utilizza una [Chiave gestita da AWS](#) per crittografare la versione decompressa del pacchetto.

### Note

Per le funzioni e i livelli che utilizzano l'archiviazione di codice S3 [autogestita](#), Lambda non memorizza una copia del codice sorgente. Il codice rimane nel bucket S3 e la crittografia inattiva del codice sorgente è gestita dalla configurazione del bucket S3. Lambda utilizzerà



KMSKeyArn per crittografare la versione decompressa del pacchetto utilizzata da Lambda per richiamare la funzione. Per ulteriori informazioni, consulta [Protecting data with server-side encryption](#) nella Amazon Simple Storage Service User Guide.

## Lambda console

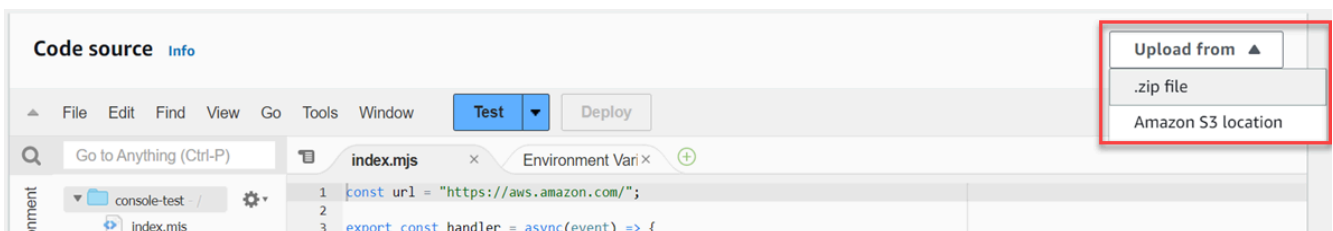
Per aggiungere la crittografia della chiave gestita dal cliente durante la creazione di una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli Crea funzione.
3. Scegliere Author from scratch (Crea da zero) o Container image (Immagine di container).
4. In Basic information (Informazioni di base) eseguire queste operazioni:
  - a. In Function name (Nome funzione), immettere il nome della funzione.
  - b. Per Runtime, scegliere la versione della lingua da utilizzare per la funzione.
5. Espandi Impostazioni avanzate, quindi seleziona Abilita la crittografia con una chiave gestita AWS KMS dal cliente.
6. Scegli una chiave gestita dal cliente.
7. Scegli Crea funzione.

Per rimuovere la crittografia con la chiave gestita dal cliente o per utilizzare una chiave diversa, è necessario caricare nuovamente il pacchetto di implementazione .zip.

Per aggiungere la crittografia con la chiave gestita dal cliente a una funzione esistente

1. Aprire la pagina [Functions](#) (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli il file .zip o la posizione di Amazon S3.



5. Carica il file o inserisci la posizione Amazon S3.
6. Scegli Abilita la crittografia con una chiave gestita AWS KMS dal cliente.
7. Scegli una chiave gestita dal cliente.
8. Scegli Save (Salva).

## AWS CLI

Per aggiungere la crittografia della chiave gestita dal cliente durante la creazione di una funzione

Nel seguente esempio [create-function](#):

- `--code`: specifica il percorso locale del pacchetto di distribuzione .zip (ZipFile) e la chiave gestita dal cliente per crittografarlo (). `SourceKMSKeyArn`
- `--kms-key-arn`: specifica la chiave gestita dal cliente per crittografare le variabili d'ambiente e la versione compressa del pacchetto di implementazione.

```
aws lambda create-function \  
  --function-name myFunction \  
  --runtime nodejs24.x \  
  --handler index.handler \  
  --role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
  --code ZipFile=fileb://myFunction.zip,SourceKMSKeyArn=arn:aws:kms:us-  
east-1:111122223333:key/key-id \  
  --kms-key-arn arn:aws:kms:us-east-1:111122223333:key/key2-id
```

Nel seguente esempio [create-function](#):

- `--code`: specifica la posizione del file .zip in un bucket Amazon S3 (`S3Bucket`, `S3ObjectVersion`) e la chiave gestita dal cliente per `S3Key` crittografarlo (). `SourceKMSKeyArn`
- `--kms-key-arn`: specifica la chiave gestita dal cliente per crittografare le variabili d'ambiente e la versione compressa del pacchetto di implementazione.

```
aws lambda create-function \  
  --function-name myFunction \  
  --runtime nodejs24.x --handler index.handler \  
  --role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
  --code S3Bucket=myBucket,S3ObjectVersion=1,S3Key=myFunction.zip,SourceKMSKeyArn=arn:aws:kms:us-east-1:111122223333:key/key-id
```

```
--code S3Bucket=amzn-s3-demo-bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion,SourceKMSKeyArn=arn:aws:kms:us-east-1:111122223333:key/key-id \  
--kms-key-arn arn:aws:kms:us-east-1:111122223333:key/key2-id
```

Per aggiungere la crittografia con la chiave gestita dal cliente a una funzione esistente

Nel seguente esempio di [update-function-code](#):

- `--zip-file`: specifica il percorso locale del pacchetto di implementazione .zip.
- `--source-kms-key-arn`: specifica la chiave gestita dal cliente per crittografare la versione compressa del pacchetto di implementazione. Lambda utilizza una chiave AWS proprietaria per crittografare il pacchetto decompresso per le invocazioni di funzioni. Se desideri utilizzare una chiave gestita dal cliente per crittografare la versione decompressa del pacchetto, esegui il comando [update-function-configuration](#) con l'opzione `--kms-key-arn`.

```
aws lambda update-function-code \  
  --function-name myFunction \  
  --zip-file fileb://myFunction.zip \  
  --source-kms-key-arn arn:aws:kms:us-east-1:111122223333:key/key-id
```

Nel seguente esempio di [update-function-code](#):

- `--s3-bucket`: specifica la posizione del file .zip in un bucket Amazon S3.
- `--s3-key`: specifica la chiave Amazon S3 del pacchetto di implementazione.
- `--s3-object-version`: per gli oggetti con versione, la versione dell'oggetto del pacchetto di implementazione da utilizzare.
- `--source-kms-key-arn`: specifica la chiave gestita dal cliente per crittografare la versione compressa del pacchetto di implementazione. Lambda utilizza una chiave AWS proprietaria per crittografare il pacchetto decompresso per le invocazioni di funzioni. Se desideri utilizzare una chiave gestita dal cliente per crittografare la versione decompressa del pacchetto, esegui il comando [update-function-configuration](#) con l'opzione `--kms-key-arn`.

```
aws lambda update-function-code \  
  --function-name myFunction \  
  --s3-bucket amzn-s3-demo-bucket \  
  --s3-key myFileName.zip \  
  --source-kms-key-arn arn:aws:kms:us-east-1:111122223333:key/key-id
```

```
--s3-object-version myObject Version  
--source-kms-key-arn arn:aws:kms:us-east-1:111122223333:key/key-id
```

Per rimuovere la crittografia con la chiave gestita dal cliente da una funzione esistente

Nel seguente esempio di [update-function-code](#), `--zip-file` specifica il percorso locale del pacchetto di implementazione `.zip`. Quando si esegue questo comando senza l'`--source-kms-key-arn` opzione, Lambda utilizza una chiave AWS proprietaria per crittografare la versione zippata del pacchetto di distribuzione.

```
aws lambda update-function-code \  
  --function-name myFunction \  
  --zip-file fileb://myFunction.zip
```

# Creare una funzione Lambda utilizzando un'immagine di container

Il codice della tua AWS Lambda funzione è costituito da script o programmi compilati e dalle relative dipendenze. Utilizza un pacchetto di implementazione per distribuire il codice della funzione a Lambda. Lambda supporta due tipi di pacchetti di implementazione: immagini di container e archivi di file .zip.

Esistono tre modi per creare un'immagine di container per una funzione Lambda:

- [Usando un AWS immagine base per Lambda](#)

[Le immagini di base AWS](#) sono precaricate con un runtime in linguaggio, un client di interfaccia di runtime per gestire l'interazione tra Lambda e il codice della funzione e un emulatore di interfaccia di runtime per i test locali.

- [Usando un AWS OS-only immagine di base](#)

[AWS OS-only le immagini di base](#) contengono una distribuzione Amazon Linux e l'emulatore dell'interfaccia [di runtime](#). Queste immagini vengono comunemente utilizzate per creare immagini contenitore per linguaggi compilati, come [Go](#) e [Rust](#), e per una lingua o una versione per la quale Lambda non fornisce un'immagine di base, come 19. Node.js Puoi anche utilizzare immagini di OS-only base per implementare un runtime [personalizzato](#). Per rendere l'immagine compatibile con Lambda, devi includere nell'immagine un [client di interfaccia di runtime](#) per il tuo linguaggio.

- [Usando un non-AWS immagine di base](#)

È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere nell'immagine un [client di interfaccia di runtime](#) per il tuo linguaggio.

## Tip

Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta [Utilizzo di compilazioni a più fasi](#) nella documentazione Docker. Per creare immagini di container efficienti, segui le [best practice per scrivere file Docker](#).

Per creare una funzione Lambda da un'immagine di un container, crea l'immagine localmente e carica in un repository Amazon Elastic Container Registry (Amazon ECR). Se si utilizza

un'immagine container fornita da un venditore [AWS Marketplace](#), devi prima clonare l'immagine nel tuo repository Amazon ECR privato. Quindi, specifica l'URI del repository al momento della creazione della funzione. Il repository Amazon ECR deve essere nello stesso della funzione Regione AWS Lambda. Puoi creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta [Autorizzazioni multiaccount Amazon ECR](#).

#### Note

Lambda non supporta gli endpoint FIPS di Amazon ECR per le immagini dei container. Se l'URI del repository include `ecr-fips`, stai utilizzando un endpoint FIPS. Esempio: `111122223333.dkr.ecr-fips.us-east-1.amazonaws.com`.

Questa pagina spiega i tipi di immagini di base e i requisiti per la creazione di immagini Lambda-compatible contenitore.

#### Note

Non è possibile modificare il [tipo di pacchetto di implementazione](#) (.zip o immagine di container) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio di file .zip. È necessario creare una nuova funzione.

## Argomenti

- [Requisiti](#)
- [Usando un AWS immagine base per Lambda](#)
- [Usando un AWS OS-only immagine di base](#)
- [Usando un non-AWS immagine di base](#)
- [Client di interfaccia runtime](#)
- [Autorizzazioni Amazon ECR](#)
- [Ciclo di vita delle funzioni](#)

## Requisiti

Installa la [AWS CLI versione 2](#) e la [CLI Docker](#). Sono inoltre necessari i seguenti requisiti:

- L'immagine di container deve implementare l'[Utilizzo dell'API di runtime Lambda per runtime personalizzati](#). I [client dell'interfaccia runtime AWS](#) open-source implementano l'API. È possibile aggiungere un client di interfaccia di runtime all'immagine di base preferita per renderla compatibile con Lambda.
- L'immagine di container deve essere in grado di funzionare su un filesystem di sola lettura. Il codice della funzione può accedere a una directory `/tmp` scrivibile con spazio di storage compreso tra 512 MB e 10.240 MB con incrementi di 1 MB.
- L'utente Lambda predefinito deve essere in grado di leggere tutti i file necessari per eseguire il codice della funzione. Lambda segue le best practice di sicurezza tramite la definizione di un utente Linux predefinito con autorizzazioni meno privilegiate. Ciò significa che non è necessario specificare un [USER](#) nel Dockerfile. Verifica che il codice dell'applicazione non si basi su file che altri utenti Linux non possono eseguire.
- Lambda supporta solo le immagini dei Linux-based contenitori.
- Lambda fornisce immagini di base multi-architettura. Tuttavia, l'immagine creata per la tua funzione deve essere destinata a una sola delle architetture. Lambda non supporta funzioni che utilizzano immagini container multi-architettura.

## Usando un AWS immagine base per Lambda

È possibile utilizzare una delle [immagini di base AWS](#) per Lambda per creare l'immagine di container per il codice della funzione. Le immagini di base sono precaricate con un runtime di lingua e altri componenti necessari per eseguire un'immagine container su Lambda. Aggiungere il codice funzione e le dipendenze all'immagine di base e quindi impacchettarlo come immagine container.

AWS fornisce periodicamente aggiornamenti alle immagini di AWS base per Lambda. Se il Dockerfile include il nome dell'immagine nella proprietà `FROM`, il client Docker estrae l'ultima versione dell'immagine dal [repository Amazon ECR](#). Per utilizzare l'immagine di base aggiornata, è necessario ricostruire l'immagine di container e [aggiornare il codice della funzione](#).

Le immagini di base Node.js 20, Python 3.12, Java 21, .NET 8, Ruby 3.3 e versioni successive sono basate sull'immagine minima del contenitore di [Amazon Linux 2023](#). Le immagini di base precedenti utilizzano Amazon Linux 2. AL2023 offre diversi vantaggi rispetto ad Amazon Linux 2, tra cui un'impronta di implementazione ridotta e versioni aggiornate di librerie come `glibc`.

AL2023-based le immagini utilizzano `microdnf` (con collegamenti simbolici `dnf`) come gestore di pacchetti invece di `yum`, che è il gestore di pacchetti predefinito in Amazon Linux 2. `microdnf` è un'implementazione autonoma di `dnf`. Per un elenco dei pacchetti inclusi nelle AL2023-based immagini, consulta le colonne Minimal Container in [Confronto dei pacchetti installati su Amazon Linux 2023 Container Images](#). Per ulteriori informazioni sulle differenze tra AL2023 e Amazon Linux 2, consulta [Introduzione del runtime di Amazon Linux 2023 per AWS Lambda](#) nel blog AWS Compute.

#### Note

Per eseguire AL2023-based le immagini localmente, incluso con AWS Serverless Application Model (AWS SAM), devi utilizzare la versione Docker 20.10.10 o successiva.

Per creare un'immagine del contenitore utilizzando un'immagine di AWS base, scegli le istruzioni per la tua lingua preferita:

- [Node.js](#)
- [TypeScript](#) (utilizza un'immagine Node.js di base)
- [Python](#)
- [Java](#)
- [Go](#)
- [.NET](#)
- [Ruby](#)

## Usando un AWS OS-only immagine di base

[AWS OS-only le immagini di base](#) contengono una distribuzione Amazon Linux e l'emulatore [dell'interfaccia](#) di runtime. Queste immagini vengono comunemente utilizzate per creare immagini contenitore per linguaggi compilati, come [Go](#) e [Rust](#), e per una lingua o una versione per la quale Lambda non fornisce un'immagine di base, come 19. Node.js. Puoi anche utilizzare immagini di OS-only base per implementare un runtime [personalizzato](#). Per rendere l'immagine compatibile con Lambda, devi includere nell'immagine un [client di interfaccia di runtime](#) per il tuo linguaggio.



| Tag    | Runtime         | Sistema operativo | Dockerfile                                                   | Raggiunta obsolescenza |
|--------|-----------------|-------------------|--------------------------------------------------------------|------------------------|
| al2023 | OS-only Runtime | Amazon Linux 2023 | <a href="#">Dockerfile per OS-only Runtime attivo GitHub</a> | 30 giugno 2029         |

Galleria pubblica del registro Amazon Elastic Container: gallery.ecr. [aws/lambda/fornito](#)

## Usando un non-AWS immagine di base

Lambda supporta qualsiasi immagine conforme a uno dei seguenti formati manifest per le immagini:

- Docker Image Manifest V2 Schema 2 (utilizzato con Docker versione 1.10 e successive)
- Open Container Initiative (OCI) Specifications (v1.0.0 e successive)

Lambda supporta una dimensione massima dell'immagine non compressa pari a 10 GB, inclusi tutti i livelli.

### Note

- Per rendere l'immagine compatibile con Lambda, devi includere nell'immagine un [client di interfaccia di runtime](#) per il tuo linguaggio.
- Per prestazioni ottimali, mantenere la dimensione del manifesto dell'immagine al di sotto di 25.400 byte. Per ridurre le dimensioni del manifesto dell'immagine, ridurre al minimo il numero di livelli nell'immagine e riduci le annotazioni.

## Client di interfaccia runtime

Se si utilizza un'immagine di [OS-only base](#) o un'immagine di base alternativa, è necessario includere un client di interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime deve estendere il [Utilizzo dell'API di runtime Lambda per runtime personalizzati](#), che gestisce l'interazione tra Lambda e il codice funzione. AWS fornisce client di interfaccia di runtime open source per i seguenti linguaggi:

- [Node.js](#)

- [Python](#)
- [Java](#)
- [.NET](#)
- [Go](#)
- [Ruby](#)
- [Rust](#) —

Se utilizzi un linguaggio che non dispone di un AWS client di interfaccia di runtime fornito, devi crearne uno tuo.

## Autorizzazioni Amazon ECR

Prima di creare la funzione Lambda da un'immagine di container, è necessario creare un'immagine del container in locale e caricarla in un repository Amazon ECR. Quando crei la funzione, specifica l'URI del repository Amazon ECR.

Assicurati che le autorizzazioni per l'utente o il ruolo che crea la funzione includano `GetRepositoryPolicy`, `SetRepositoryPolicy`, `BatchGetImage`, e `GetDownloadUrlForLayer`.

Ad esempio, utilizza la console IAM per creare un ruolo con la seguente policy:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "ecr:SetRepositoryPolicy",
        "ecr:GetRepositoryPolicy",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world"
    }
  ]
}
```

```
]
}
```

Le autorizzazioni necessarie a Lambda per recuperare l'immagine del contenitore dipendono dal fatto che il repository Amazon ECR si trovi nello stesso AWS account della funzione o in un account diverso. Le sezioni seguenti spiegano i requisiti per ogni scenario.

#### Note

In IAM, l'accesso con lo stesso account richiede solo una parte per concedere l'autorizzazione: la policy basata sull'identità (sul ruolo) o la policy basata sulle risorse (sul repository Amazon ECR). Cross-account l'accesso richiede che entrambe le parti concedano l'autorizzazione: la politica basata sull'identità sul ruolo nell'account di consumo E la politica basata sulle risorse sul repository Amazon ECR nell'account proprietario devono entrambe consentire l'azione.

### Same-account politiche del repository Amazon ECR

Per una funzione nello stesso account dell'immagine del contenitore in Amazon ECR, puoi concedere l'accesso a Lambda tramite la policy basata sull'identità sul ruolo di esecuzione o la policy basata sulle risorse sul repository Amazon ECR. Solo una parte deve consentire l'accesso.

Se scegli di utilizzare una policy di repository Amazon ECR, aggiungi `ecr:BatchGetImage` e `ecr:GetDownloadUrlForLayer` autorizzazioni. L'esempio seguente mostra il valore minimo della policy.

```
{
  "Sid": "LambdaECRImageRetrievalPolicy",
  "Effect": "Allow",
  "Principal": {
    "Service": "lambda.amazonaws.com"
  },
  "Action": [
    "ecr:BatchGetImage",
    "ecr:GetDownloadUrlForLayer"
  ]
}
```

Per ulteriori informazioni sulle autorizzazioni per il repository Amazon ECR, consulta la pagina [Policy dei repository privati](#) nella Guida per l'utente di Amazon Elastic Container Registry.

Se il repository Amazon ECR non include queste autorizzazioni, Lambda tenta di aggiungerle automaticamente. Lambda può aggiungere autorizzazioni solo se l'entità principale che chiama Lambda dispone delle autorizzazioni `ecr:getRepositoryPolicy` e `ecr:setRepositoryPolicy`.

Per visualizzare o modificare le autorizzazioni del repository Amazon ECR, segui le istruzioni riportate nella pagina [Impostazione di un'istruzione di policy per i repository privati](#) della Guida per l'utente di Amazon Elastic Container Registry.

### Autorizzazioni multiaccount Amazon ECR

Quando una funzione in un account utilizza un'immagine del contenitore da un repository Amazon ECR in un altro account, entrambe le parti devono concedere l'accesso. La politica basata sull'identità sul ruolo nell'account di consumo deve consentire `ecr:BatchGetImage` e `ecr:GetDownloadUrlForLayer`, inoltre, la politica basata sulle risorse sul repository Amazon ECR nell'account proprietario deve consentire queste azioni.

Nell'esempio seguente, la [policy delle autorizzazioni del repository Amazon ECR](#) richiede le seguenti istruzioni per concedere l'accesso all'account numero 123456789012.

- **CrossAccountPermission**— Consente all'account 123456789012 di creare e aggiornare funzioni Lambda che utilizzano immagini da questo repository ECR.
- **LambdaECRImageCrossAccountRetrievalPolicy**— Lambda alla fine imposta lo stato di una funzione su inattivo se non viene richiamata per un periodo prolungato. Questa istruzione è necessaria affinché Lambda possa recuperare l'immagine del container per l'ottimizzazione e la memorizzazione nella cache per conto della funzione di proprietà di 123456789012.

Example- Aggiunta di autorizzazioni multi-account al repository

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

    "Sid": "CrossAccountPermission",
    "Effect": "Allow",
    "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
    ],
    "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
    },
    "Resource": "arn:aws:ecr:us-east-1:123456789012:repository/example-lambda-
repository"
  },
  {
    "Sid": "LambdaECRImageCrossAccountRetrievalPolicy",
    "Effect": "Allow",
    "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
    ],
    "Principal": {
        "Service": "lambda.amazonaws.com"
    },
    "Condition": {
        "ArnLike": {
            "aws:sourceARN": "arn:aws:lambda:us-east-1:123456789012:function:*"
        }
    },
    "Resource": "arn:aws:ecr:us-east-1:123456789012:repository/example-lambda-
repository"
  }
]
}

```

Per consentire l'accesso a più account, aggiungi gli ID account all'elenco Principal nella CrossAccountPermission policy e alla lista di valutazione delle condizioni nel LambdaECRImageCrossAccountRetrievalPolicy.

Se lavori con più account in un' AWS organizzazione, ti consigliamo di enumerare ogni ID account nella politica delle autorizzazioni ECR. Questo approccio è in linea con la best practice di AWS sicurezza che prevede l'impostazione di autorizzazioni ristrette nelle politiche IAM.

Oltre alla policy del repository Amazon ECR, l'utente o il ruolo che crea la funzione deve disporre anche delle `GetDownloadUrlForLayer` autorizzazioni previste nella propria policy `BatchGetImage` basata sull'identità.

## Ciclo di vita delle funzioni

Dopo aver caricato un'immagine container nuova o aggiornata, Lambda ottimizza l'immagine prima che la funzione possa elaborare le chiamate. Il processo di ottimizzazione può richiedere alcuni secondi. La funzione rimane nello stato `Pending` fino al completamento del processo, quando lo stato passa a `Active`. Non è possibile richiamare la funzione fino a quando non raggiunge lo stato `Active`.

Se una funzione non viene richiamata per più settimane, Lambda recupera la sua versione ottimizzata e la funzione passa allo stato `Inactive`. Per riattivare la funzione, è necessario invocarla. Lambda rifiuta la prima invocazione e la funzione entra nello stato `Pending` finché Lambda non riottimizza l'immagine. La funzione ritorna quindi allo stato `Active`.

Lambda recupera periodicamente l'immagine di container associata dal repository Amazon ECR. Se l'immagine di container corrispondente non esiste più in Amazon ECR o le autorizzazioni sono state revocate, la funzione entra nello stato `Failed` e Lambda restituisce un errore per qualsiasi chiamata della funzione.

È possibile utilizzare l'API Lambda per ottenere informazioni sullo stato di una funzione. Per ulteriori informazioni, consulta [Stati funzione Lambda](#).

## Self-managed Archiviazione del codice S3

Per impostazione predefinita, quando crei o aggiorni una funzione o un layer Lambda da un archivio di file .zip, Lambda archivia una copia del codice in memoria. Lambda-managed Ogni AWS account ha una quota di 300 GB di spazio di Lambda-managed archiviazione per regione.

Con l'archiviazione del codice S3 autogestita, puoi configurare Lambda in modo che faccia riferimento al tuo codice direttamente da un bucket S3 nel tuo account. Lambda non memorizza una copia del codice, quindi il codice non viene conteggiato nella tua quota di archiviazione. Lambda-managed Lambda accede al codice direttamente dal bucket S3.

Self-managed L'archiviazione del codice S3 è disponibile per tutte le funzioni e i livelli creati e aggiornati utilizzando archivi di file .zip. È compatibile con tutte le classi di archiviazione S3 ad eccezione delle classi di archiviazione Glacier.

### Note

Self-managed L'archiviazione del codice S3 non modifica la dimensione massima del pacchetto di distribuzione .zip, che rimane a 250 MB (decompresso).

### Argomenti

- [Configurazione dell'archiviazione del codice S3 autogestita](#)
- [Configurazione delle autorizzazioni](#)
- [Creazione di una funzione con storage S3 autogestito](#)
- [Utilizzo dello storage S3 autogestito con livelli](#)
- [Passaggio da una modalità di archiviazione all'altra](#)
- [Ciclo di vita delle funzioni con storage autogestito](#)
- [Cross-account e utilizzo in più regioni](#)
- [Considerazioni](#)

## Configurazione dell'archiviazione del codice S3 autogestita

Per utilizzare lo storage di codice S3 autogestito, completa i seguenti passaggi:

1. Crea un bucket S3 o utilizza un bucket esistente nel tuo account.

2. Abilita il controllo delle versioni <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Versioning.html> S3 per il bucket. Lambda richiede il controllo delle versioni di S3 per tenere traccia della versione dell'oggetto sorgente da utilizzare. Ogni volta che carichi un nuovo pacchetto .zip, S3 crea una nuova versione in modo che Lambda possa distinguere gli aggiornamenti.
3. Carica il tuo pacchetto di distribuzione .zip nel bucket S3.
4. Concedi l'entità del servizio Lambda `s3:GetObject` e le `s3:GetObjectVersion` autorizzazioni per l'oggetto sorgente. Vedi [Configurazione delle autorizzazioni](#).
5. Crea o aggiorna la funzione o il livello, specificando l'oggetto S3, la chiave S3, la versione dell'oggetto S3 e impostando su. `S3ObjectStorageMode REFERENCE`

## Configurazione delle autorizzazioni

Per utilizzare l'archiviazione di codice S3 autogestita, concedi al servizio Lambda service principal (`lambda.amazonaws.com`) l'autorizzazione ad accedere agli oggetti sorgente. Aggiungi la seguente policy per il bucket S3:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "LambdaSelfManagedCodeAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3:::my-bucket/my-function.zip"
      ],
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:lambda:us-east-1:111122223333:function:my-function"
        }
      }
    }
  ]
}
```



```
}
```

## Creazione di una funzione con storage S3 autogestito

È possibile creare una funzione con archiviazione di codice S3 autogestita utilizzando la console, il, o. AWS CLI AWS CloudFormation

### Note

L'editor di codice della console Lambda non è disponibile per le funzioni che utilizzano la modalità Reference. Per modificare il codice della funzione, aggiorna il file .zip nel bucket S3 e aggiorna la funzione.

### Utilizzo della console

Quando aggiorni una funzione da una posizione S3 nella console Lambda, puoi scegliere la modalità di archiviazione del codice.

Per aggiornare una funzione per utilizzare la modalità Reference (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere la funzione da aggiornare e scegliere la scheda Codice
3. In Code source, scegli Aggiorna, quindi scegli Aggiorna da un file in Amazon S3.
4. Per l'URL del link di Amazon S3, inserisci l'URL del link S3 del tuo file .zip.
5. In modalità di archiviazione del codice, scegli la modalità di riferimento.
6. Scegli Save (Salva).

Dopo aver configurato la modalità di riferimento, la sezione delle proprietà del codice nella scheda Codice della funzione mostra la modalità di archiviazione del codice: Reference e la posizione S3 che punta al bucket S3.

### Utilizzo di AWS CLI

Usa il `create-function` comando con il parametro: `S3ObjectStorageMode=REFERENCE`

```
aws lambda create-function \  
  --function-name my-function \  
  --runtime python3.8 \  
  --code S3ObjectStorageMode=REFERENCE,s3Bucket=my-bucket,s3Key=my-key.zip
```

```
--runtime python3.12 \  
--role arn:aws:iam::111122223333:role/lambda-execution-role \  
--handler lambda_function.lambda_handler \  
--code S3Bucket=my-bucket,\  
S3Key=my-function.zip,\  
S3ObjectVersion=abc123def456,\  
S3ObjectStorageMode=REFERENCE
```

Per aggiornare una funzione esistente per utilizzare lo storage S3 autogestito:

```
aws lambda update-function-code \  
--function-name my-function \  
--s3-bucket my-bucket \  
--s3-key my-function.zip \  
--s3-object-version abc123def456 \  
--s3-object-storage-mode REFERENCE
```

#### Important

È necessario specificare `S3ObjectStorageMode=REFERENCE` in ogni chiamata a `update-function-code`. Se si omette `S3ObjectStorageMode`, il valore predefinito è `COPY` e Lambda memorizza il codice in memoria. Lambda-managed

## Utilizzo AWS CloudFormation

Nel CloudFormation modello, imposta su `S3ObjectStorageMode REFERENCE` nella `Code` proprietà della risorsa: `AWS::Lambda::Function`

```
Resources:  
  MyFunction:  
    Type: AWS::Lambda::Function  
    Properties:  
      FunctionName: my-function  
      Runtime: python3.12  
      Handler: lambda_function.lambda_handler  
      Role: !GetAtt LambdaExecutionRole.Arn  
      Code:  
        S3Bucket: my-bucket  
        S3Key: my-function.zip  
        S3ObjectVersion: abc123def456
```

## S3objectStorageMode: REFERENCE

### Utilizzo dello storage S3 autogestito con livelli

È inoltre possibile utilizzare lo storage di codice S3 autogestito per le versioni a più livelli. Usa il `publish-layer-version` comando con: `S3objectStorageMode=REFERENCE`

```
aws lambda publish-layer-version \  
  --layer-name my-layer \  
  --content S3Bucket=my-bucket,S3Key=my-layer.zip,S3objectStorageMode=REFERENCE \  
  --compatible-runtimes python3.12
```

È possibile utilizzare qualsiasi combinazione Lambda-managed e archiviazione autogestita per il codice funzione e il codice di livello.

### Passaggio da una modalità di archiviazione all'altra

È possibile passare dall'archiviazione del codice S3 all'altra Lambda-managed e gestirla automaticamente quando si aggiorna il codice della funzione o si pubblica una nuova versione del layer. Impostato su `S3objectStorageMode` uno o: `COPY` `REFERENCE`

- `COPY`(impostazione predefinita): Lambda copia il codice sorgente nell' `Lambda-managed` archivio.
- `REFERENCE`— Lambda fa riferimento al codice sorgente direttamente dal bucket S3 senza archiviarne una copia.

Quando si passa da `COPY` a `REFERENCE`, Lambda elimina la `Lambda-managed` copia del codice sorgente. Quando si passa da `REFERENCE` a `COPY`, Lambda crea una copia del codice sorgente in memoria. `Lambda-managed` Il passaggio a `COPY` non riesce se si supera il limite di archiviazione del `Lambda-managed` codice.

### Ciclo di vita delle funzioni con storage autogestito

Lambda accede periodicamente all'oggetto sorgente dal bucket S3 per riottimizzare il codice della funzione. È necessario mantenere l'accesso all'oggetto sorgente affinché la funzione rimanga attiva.

- Se Lambda perde l'accesso all'oggetto sorgente di una funzione, la funzione passa allo `Inactive` stato. Per ripristinare la funzione, ripristina l'accesso all'oggetto sorgente e aggiorna la funzione.
- Se Lambda perde l'accesso all'oggetto sorgente per un livello, la funzione rimane `Active`. È possibile aggiornare la configurazione della funzione (senza aggiornare i livelli), aggiornare il

codice della funzione o aggiornare la configurazione della funzione con nuovi livelli. Tuttavia, l'aggiornamento della configurazione della funzione con il livello inaccessibile non riesce.

## Cross-account e utilizzo in più regioni

Puoi creare funzioni e livelli dal codice sorgente nei bucket S3 in diversi account e in diverse regioni. Cross-Region il recupero degli oggetti comporta costi aggiuntivi per il trasferimento dei dati S3. Per i dettagli, consulta i prezzi [di S3](#).

## Considerazioni

- Self-managed L'archiviazione del codice S3 è disponibile solo per le funzioni e i livelli di archiviazione di file .zip. Le funzioni di immagine dei contenitori continuano a utilizzare Amazon ECR.
- La dimensione massima del pacchetto di distribuzione .zip (250 MB decompresso) rimane invariata.
- Paghi le tariffe di storage standard di Amazon S3 per il codice archiviato nei tuoi bucket autogestiti. Lambda non addebita alcun costo per il recupero di oggetti S3 all'interno della stessa regione.
- Self-managed L'archiviazione del codice S3 è compatibile con tutte le classi di archiviazione S3 ad eccezione delle classi di archiviazione Glacier.
- Puoi utilizzare funzionalità S3 come S3 Cross-Region Replication per condividere gli artefatti tra le regioni e le policy del ciclo di vita S3 per gestire gli oggetti.

# Configurare la memoria di una funzione Lambda

Lambda alloca la potenza della CPU in proporzione alla quantità di memoria configurata. Memory (Memoria) indica la quantità di memoria disponibile per la funzione Lambda in fase di runtime. È possibile aumentare o diminuire la memoria e la potenza della CPU assegnate alla funzione utilizzando l'impostazione Memoria. È possibile configurare memoria compresa tra 128 MB e 10.240 MB in incrementi di 1 MB. A 1.769 MB, una funzione ha l'equivalente di una vCPU (un v CPU-second di crediti al secondo).

In questa pagina viene descritto come e quando aggiornare l'impostazione della memoria per una funzione Lambda.

## Sections

- [Determinazione dell'impostazione di memoria appropriata per una funzione Lambda](#)
- [Configurazione della memoria delle funzioni \(console\)](#)
- [Configurazione della memoria delle funzioni \(AWS CLI\)](#)
- [Configurazione della memoria funzionale \(AWS SAM\)](#)
- [Accettazione dei suggerimenti relativi alla memoria delle funzioni \(console\)](#)

## Determinazione dell'impostazione di memoria appropriata per una funzione Lambda

La memoria è la leva principale per controllare le prestazioni di una funzione. L'impostazione predefinita, 128 MB, è l'impostazione più bassa possibile. Ti consigliamo di utilizzare 128 MB solo per funzioni Lambda semplici, come quelle che trasformano e instradano gli eventi verso altri servizi AWS .

Una allocazione maggiore della memoria può migliorare le prestazioni per le funzioni che utilizzano librerie importate, livelli [Lambda](#), Amazon Simple Storage Service (Amazon S3) o Amazon Elastic File System (Amazon EFS). L'aggiunta di più memoria aumenta proporzionalmente la quantità di CPU, aumentando la potenza di calcolo complessiva disponibile. Se una funzione è collegata alla CPU, alla rete o alla memoria, l'aumento dell'impostazione della memoria può migliorarne notevolmente le prestazioni.

Per trovare la giusta configurazione di memoria, monitora le tue funzioni con Amazon CloudWatch e imposta degli allarmi se il consumo di memoria si avvicina ai valori massimi configurati. Questo può aiutare a identificare le funzioni legate alla memoria. Per quanto riguarda CPU-bound e IO-bound

funzioni, anche il monitoraggio della durata può fornire informazioni. In questi casi, l'aumento della memoria può aiutare a risolvere i problemi di elaborazione o di rete.

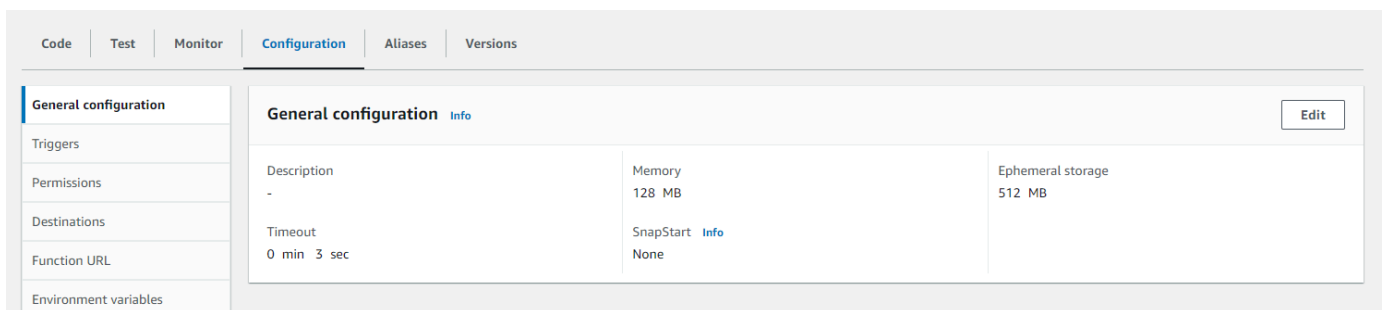
Puoi anche prendere in considerazione l'utilizzo dello strumento open source [Power Tuning AWS Lambda](#). Questo strumento consente AWS Step Functions di eseguire più versioni simultanee di una funzione Lambda con diverse allocazioni di memoria e misurare le prestazioni. La funzione di input viene eseguita nel tuo AWS account, eseguendo chiamate HTTP in tempo reale e interazioni SDK, per misurare le probabili prestazioni in uno scenario di produzione live. Puoi anche implementare un CI/CD processo per utilizzare questo strumento per misurare automaticamente le prestazioni delle nuove funzioni che distribuisce.

## Configurazione della memoria delle funzioni (console)

È possibile configurare la memoria della funzione nella console Lambda.

Per aggiornare la memoria di una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Seleziona la scheda Configurazione, quindi scegli Configurazione generale.



4. In Configurazione generale, scegli Modifica.
5. Per Memoria, impostare un valore compreso tra 128 MB e 10.240 MB.
6. Scegli Save (Salva).

## Configurazione della memoria delle funzioni (AWS CLI)

Per configurare la memoria della tua funzione, puoi usare il comando [update-function-configuration](#).

Example

```
aws lambda update-function-configuration \
```

```
--function-name my-function \  
--memory-size 1024
```

## Configurazione della memoria funzionale (AWS SAM)

Per configurare la memoria per la tua funzione, puoi usare [AWS Serverless Application Model](#). Aggiorna la [MemorySize](#) proprietà nel tuo `template.yaml` file e poi esegui [sam deploy](#).

Example `template.yaml`

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Description: An AWS Serverless Application Model template describing your function.  
Resources:  
  my-function:  
    Type: AWS::Serverless::Function  
    Properties:  
      CodeUri: .  
      Description: ''  
      MemorySize: 1024  
      # Other function properties...
```

## Accettazione dei suggerimenti relativi alla memoria delle funzioni (console)

Se disponi delle autorizzazioni di amministratore in AWS Identity and Access Management (IAM), puoi scegliere di ricevere consigli sulle impostazioni della memoria funzionale Lambda da AWS Compute Optimizer. Per istruzioni su come attivare i suggerimenti sulla memoria per il proprio account o la propria organizzazione, consultare [Accettazione delle opzioni nell'account](#) nella Guida per l'utente di AWS Compute Optimizer.

### Note

Compute Optimizer supporta solo le funzioni che utilizzano l'architettura x86\_64.

Dopo avere accettato le opzioni e se la [funzione Lambda soddisfa i requisiti del Sistema di ottimizzazione del calcolo](#), puoi visualizzare e accettare i suggerimenti sulla memoria della funzione del Sistema di ottimizzazione del calcolo nella console Lambda in Configurazione generale.

# Configurare lo spazio di archiviazione temporaneo per le funzioni Lambda

Lambda fornisce uno spazio di archiviazione temporaneo per le funzioni nella directory `/tmp`. Questo spazio di archiviazione è temporaneo e unico per ogni ambiente di esecuzione. È possibile controllare la quantità di spazio di archiviazione temporaneo allocato alla funzione utilizzando l'impostazione Archiviazione temporanea. È possibile configurare uno spazio di archiviazione temporaneo compreso tra 512 MB e 10.240 MB in incrementi di 1 MB. Tutti i dati archiviati in `/tmp` vengono crittografati a riposo con una chiave gestita da AWS.

In questa pagina sono descritti i casi d'uso comuni e come aggiornare lo spazio di archiviazione temporanea per una funzione Lambda.

## Sections

- [Casi d'uso comuni per una maggiore archiviazione temporanea](#)
- [Configurazione dell'archiviazione temporanea \(console\)](#)
- [Configurazione dell'archiviazione temporanea \(AWS CLI\)](#)
- [Configurazione dello storage temporaneo \(AWS SAM\)](#)

## Casi d'uso comuni per una maggiore archiviazione temporanea

Ecco alcuni casi d'uso comuni che traggono vantaggio dall'aumento dello spazio di archiviazione temporanea:

- Extract-transform-load Lavori (ETL): aumenta lo storage temporaneo quando il codice esegue calcoli intermedi o scarica altre risorse per completare l'elaborazione. Più spazio temporaneo consente l'esecuzione di lavori ETL più complessi nelle funzioni Lambda.
- Inferenza con il machine learning (ML): molte attività di inferenza si basano su file di dati di riferimento di grandi dimensioni, tra cui librerie e modelli. Con più spazio di archiviazione temporanea, puoi scaricare modelli più grandi da Amazon Simple Storage Service (Amazon S3) in `/tmp` e utilizzarli nell'elaborazione.
- Elaborazione dei dati: per i carichi di lavoro che scaricano oggetti da Amazon S3 in risposta a eventi S3, più spazio `/tmp` consente di gestire oggetti più grandi senza utilizzare l'elaborazione in memoria. Anche i carichi di lavoro che creano PDF o elaborano contenuti multimediali traggono vantaggio da più spazio di archiviazione temporanea.
- Elaborazione grafica: l'elaborazione delle immagini è un caso d'uso comune per le applicazioni. Lambda-based Per i carichi di lavoro che elaborano file TIFF di grandi dimensioni o immagini



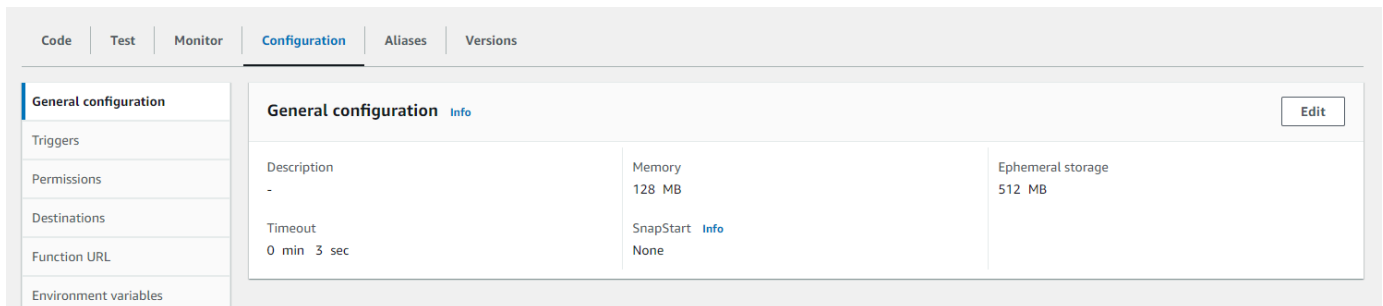
satellitari, uno spazio di archiviazione temporanea più grande semplifica l'uso delle librerie e l'esecuzione del calcolo in Lambda.

## Configurazione dell'archiviazione temporanea (console)

Puoi configurare lo spazio di archiviazione temporanea nella console Lambda.

Per modificare lo spazio di archiviazione temporanea effimera per una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Seleziona la scheda Configurazione, quindi scegli Configurazione generale.



4. In Configurazione generale, scegli Modifica.
5. Per Archiviazione temporanea, imposta un valore compreso tra 512 MB e 10.240 MB in incrementi di 1 MB.
6. Scegli Save (Salva).

## Configurazione dell'archiviazione temporanea (AWS CLI)

Per configurare lo spazio di archiviazione temporanea, puoi usare il comando [update-function-configuration](#).

### Example

```
aws lambda update-function-configuration \  
  --function-name my-function \  
  --ephemeral-storage '{"Size": 1024}'
```

## Configurazione dello storage temporaneo (AWS SAM)

Per configurare lo spazio di archiviazione temporanea per la tua funzione, puoi usare [AWS Serverless Application Model](#). Aggiorna la [EphemeralStorage](#) proprietà nel tuo `template.yaml` file e poi esegui [sam deploy](#).

Example `template.yaml`

```
AWS::TemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  my-function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 120
      Handler: index.handler
      Runtime: nodejs22.x
      Architectures:
        - x86_64
      EphemeralStorage:
        Size: 10240
      # Other function properties...
```

## Selezione e configurazione di un'architettura di set di istruzioni per la funzione Lambda

L'architettura del set di istruzioni di una funzione Lambda determina il tipo di processore del computer utilizzato da Lambda per eseguire la funzione. Lambda offre una scelta di architetture del set di istruzioni:

- arm64 — architettura ARM a 64 bit, per il processore AWS Graviton2.
- x86\_64: architettura x86 a 64 bit, per processori basati su x86.

### Note

L'architettura arm64 è disponibile nella maggior parte dei casi. Regioni AWS Per ulteriori informazioni, consulta [AWS Lambda Prezzi](#). Nella tabella dei prezzi delle memorie, scegli la scheda Arm Price, quindi apri l'elenco a discesa delle regioni per vedere quali Regioni AWS supportano arm64 con Lambda.

Per un esempio di come creare una funzione con architettura arm64, vedi [AWS Lambda Funzioni basate sul processore Graviton2. AWS](#)

### Argomenti

- [Vantaggi dell'utilizzo dell'architettura arm64](#)
- [Requisiti per la migrazione all'architettura arm64](#)
- [Compatibilità del codice della funzione con l'architettura arm64](#)
- [Come eseguire la migrazione all'architettura arm64](#)
- [Configurazione dell'architettura del set di istruzioni](#)

### Vantaggi dell'utilizzo dell'architettura arm64

Le funzioni Lambda che utilizzano l'architettura arm64 (processore AWS Graviton2) possono ottenere prezzi e prestazioni significativamente migliori rispetto alla funzione equivalente in esecuzione sull'architettura x86\_64. Prendi in considerazione l'utilizzo di arm64 per applicazioni ad uso intensivo di calcolo come computing ad alte prestazioni, codifica video e carichi di lavoro di simulazione.

La CPU Graviton2 utilizza il core Neoverse N1 e supporta (incluse le estensioni CRC e crittografiche) oltre a diverse altre estensioni architetturali. Armv8.2

Graviton2 riduce il tempo di lettura della memoria fornendo una cache L2 maggiore per vCPU, che migliora le prestazioni di latenza dei back-end Web e per dispositivi mobili, dei microservizi e dei sistemi di elaborazione dati. Graviton2 offre inoltre prestazioni di crittografia migliorate e supporta set di istruzioni che migliorano la latenza dell'inferenza dell'apprendimento automatico. CPU-based

Per ulteriori informazioni su AWS Graviton2, vedere Graviton Processor. [AWS](#)

## Requisiti per la migrazione all'architettura arm64

Quando selezioni una funzione Lambda per migrare all'architettura arm64, per garantire una migrazione fluida assicurati che la tua funzione soddisfi i seguenti requisiti:

- Il pacchetto di implementazione contiene solo componenti open source e codice sorgente sotto il tuo controllo, in modo da poter apportare tutti gli aggiornamenti necessari per la migrazione.
- Se il codice della funzione include dipendenze di terze parti, ogni libreria o pacchetto fornisce una versione arm64.

## Compatibilità del codice della funzione con l'architettura arm64

Il codice della funzione Lambda deve essere compatibile con l'architettura del set di istruzioni della funzione. Prima di migrare una funzione all'architettura arm64, tieni presente le seguenti osservazioni sul codice della funzione corrente:

- Se hai aggiunto il codice della funzione utilizzando l'editor di codice incorporato, probabilmente è possibile eseguire il codice su entrambe le architetture senza modifiche.
- Se il codice della funzione è stato caricato, è necessario caricare un nuovo codice compatibile con l'architettura specifica.
- Se la funzione utilizza livelli, è necessario [controllare ogni livello](#) per garantire che sia compatibile con la nuova architettura. Se un livello non è compatibile, modifica la funzione per sostituire la versione del livello corrente con una versione del livello compatibile.
- Se la tua funzione utilizza estensioni Lambda, è necessario controllare ciascuna estensione per assicurarsi che sia compatibile con la nuova architettura.
- Se la funzione usa un tipo di pacchetto di distribuzione dell'immagine del container, è necessario creare una nuova immagine del container compatibile con l'architettura della funzione.

## Come eseguire la migrazione all'architettura arm64

Per migrare una funzione Lambda all'architettura arm64, ti consigliamo di seguire questi passaggi:

1. Crea l'elenco di dipendenze per l'applicazione o il carico di lavoro. Le dipendenze comuni includono:
  - Tutte le librerie e i pacchetti utilizzati dalla funzione.
  - Gli strumenti utilizzati per creare, distribuire e testare la funzione, come compilatori, suite di test, pipeline di integrazione continua e distribuzione continua (CI/CD), strumenti di provisioning e script.
  - Le estensioni Lambda e gli strumenti di terza parte utilizzati per monitorare la funzione in produzione.
2. Per ciascuna delle dipendenze, controlla la versione e verifica se sono disponibili versioni arm64.
3. Crea un ambiente per eseguire la migrazione dell'applicazione.
4. Esegui il bootstrap dell'applicazione.
5. Test e debugging dell'applicazione.
6. Testa le prestazioni della funzione arm64. Confronta le prestazioni con la versione x86\_64.
7. Aggiorna la pipeline dell'infrastruttura per supportare le funzioni arm64 Lambda.
8. Simula l'implementazione per la produzione.

Ad esempio, è possibile usare la [configurazione del routing dell'alias](#) per dividere il traffico tra le versioni x86 e arm64 della funzione e confrontare le prestazioni e la latenza.

Per ulteriori informazioni su come creare un ambiente di codice per l'architettura arm64, incluse informazioni specifiche del linguaggio per Java, Go, .NET e Python, consulta il repository [Getting started with Graviton](#). [AWS](#) GitHub

## Configurazione dell'architettura del set di istruzioni

Puoi configurare l'architettura del set di istruzioni per funzioni Lambda nuove ed esistenti utilizzando la console Lambda, gli SDK, () o. AWS AWS Command Line Interface AWS CLI CloudFormation Completa questa procedura per modificare l'architettura del set di istruzioni per una funzione Lambda esistente dalla console.

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione per la quale desideri configurare l'architettura del set di istruzioni.
3. Nella scheda Codice principale, per la sezione Impostazioni di runtime, scegli Modifica.

4. In Architettura, scegli l'architettura del set di istruzioni da utilizzare per la funzione.
5. Selezionare Salva.

# Configurare il timeout della funzione Lambda

Lambda esegue il codice per un determinato periodo di tempo prima del timeout. Il timeout è il tempo massimo in secondi in cui una funzione Lambda può essere eseguita. Il valore predefinito per questa impostazione è 3 secondi, ma è possibile regolarlo con incrementi di 1 secondo fino a un valore massimo di 900 secondi (15 minuti).

In questa pagina viene descritto come e quando aggiornare l'impostazione di timeout per una funzione Lambda.

## Sections

- [Determinazione del valore di timeout appropriato per una funzione Lambda](#)
- [Configurazione del timeout \(console\)](#)
- [Configurazione del timeout \(AWS CLI\)](#)
- [Configurazione del timeout \(AWS SAM\)](#)

## Determinazione del valore di timeout appropriato per una funzione Lambda

Se il valore di timeout è vicino alla durata media di una funzione, c'è un rischio maggiore che la funzione scada inaspettatamente. La durata di una funzione può variare in base alla quantità di trasferimento ed elaborazione dei dati e alla latenza dei servizi con cui interagisce la funzione. Alcune cause comuni di timeout includono:

- I download da Amazon Simple Storage Service (Amazon S3) sono di dimensioni maggiori o richiedono più tempo della media.
- Una funzione invia una richiesta a un altro servizio, che impiega più tempo a rispondere.
- I parametri forniti a una funzione richiedono una maggiore complessità computazionale della funzione, il che fa sì che l'invocazione richieda più tempo.

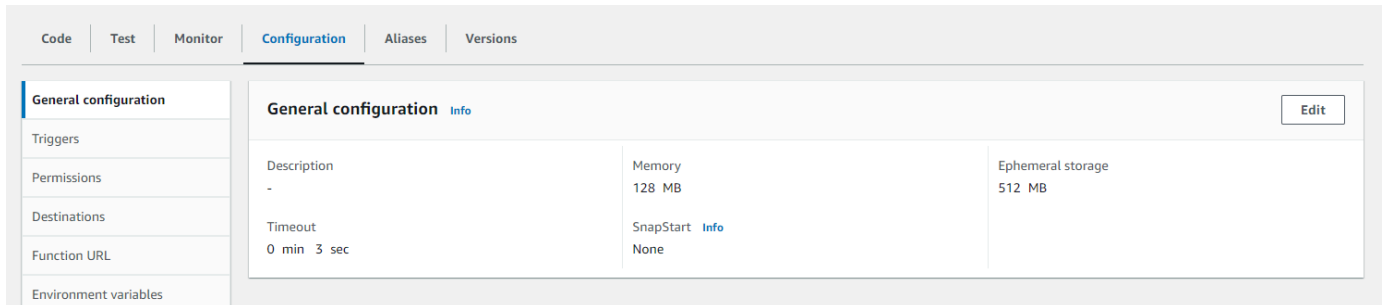
Nel testare l'applicazione, assicurati che i test riflettano accuratamente la dimensione e la quantità di dati e valori realistici dei parametri. I test utilizzano spesso campioni di piccole dimensioni per comodità, ma è consigliabile utilizzare set di dati al limite massimo di quanto ragionevolmente previsto per il carico di lavoro.

## Configurazione del timeout (console)

Puoi configurare il timeout della funzione nella console Lambda.

Per modificare il timeout di una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Seleziona la scheda Configurazione, quindi scegli Configurazione generale.



4. In Configurazione generale, scegli Modifica.
5. Per Timeout, imposta un valore compreso tra 1 e 900 secondi (15 minuti).
6. Scegli Save (Salva).

## Configurazione del timeout (AWS CLI)

Per configurare il valore di timeout, puoi usare il comando [update-function-configuration](#). Il comando di esempio seguente aumenta il timeout della funzione a 120 secondi (2 minuti).

### Example

```
aws lambda update-function-configuration \  
  --function-name my-function \  
  --timeout 120
```

## Configurazione del timeout (AWS SAM)

Per configurare il valore di timeout per la tua funzione, puoi usare [AWS Serverless Application Model](#). Aggiorna la proprietà [Timeout](#) nel file `template.yaml` e poi esegui [sam deploy](#).

### Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Description: An AWS Serverless Application Model template describing your function.  
Resources:
```



*my-function:*

Type: AWS::Serverless::Function

Properties:

CodeUri: .

Description: ''

MemorySize: 128

Timeout: *120*

*# Other function properties...*

## Utilizzo delle variabili di ambiente Lambda

È possibile utilizzare le variabili di ambiente per regolare il comportamento della funzione senza aggiornare il codice. Una variabile di ambiente è una coppia di stringhe archiviata nella configurazione specifica della versione di una funzione. Il runtime Lambda rende le variabili di ambiente disponibili per il codice e imposta variabili di ambiente aggiuntive che contengono informazioni sulla richiesta di funzione e invocazione.

### Note

Per aumentare la sicurezza, ti consigliamo di utilizzare AWS Secrets Manager al posto delle variabili di ambiente per archiviare le credenziali del database e altre informazioni sensibili come chiavi API o token di autorizzazione. Per ulteriori informazioni, consulta [Uso dei segreti in Secrets Manager nelle funzioni Lambda](#).

Le variabili di ambiente non vengono valutate prima dell'invocazione della funzione. Qualsiasi valore definito è considerato una stringa letterale e non espanso. Eseguire la valutazione della variabile nel codice funzione.

## Creazione delle variabili di ambiente Lambda

Puoi configurare le variabili di ambiente in Lambda utilizzando la console Lambda, (), AWS Command Line Interface AWS Serverless Application Model (AWS CLI) o utilizzando un AWS SAM SDK. AWS

### Console

Puoi definire le variabili di ambiente sulla versione non pubblicata della funzione. Quando pubblichi una versione, le variabili di ambiente sono bloccate per quella versione insieme ad altre [impostazioni di configurazioni specifiche della versione](#).

È possibile creare una variabile di ambiente per la funzione definendo una chiave e un valore. La funzione utilizza il nome della chiave per recuperare il valore della variabile di ambiente.

Per impostare le variabili di ambiente nella console Lambda

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegli Configurazione, quindi scegli Variabili di ambiente.

4. In Environment variables (Variabili di ambiente), scegliere Edit (Modifica).
5. Scegli Add environment variable (Aggiungi variabile d'ambiente).
6. Inserisci una coppia chiave valore.

#### Requisiti

- Le chiavi iniziano con una lettera e sono di almeno due caratteri.
  - Le chiavi contengono solo lettere, numeri e il carattere di sottolineatura (\_).
  - Le chiavi non sono [riservate da Lambda](#).
  - La dimensione totale di tutte le variabili di ambiente non supera i 4 KB.
7. Scegli Save (Salva).

#### Generazione di un elenco di variabili di ambiente nell'editor di codice della console

Puoi generare un elenco di variabili di ambiente nell'editor di codice Lambda. È un modo rapido per fare riferimento alle variabili di ambiente durante la scrittura del codice.

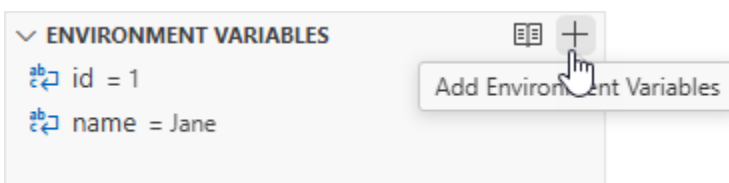
1. Scegli la scheda Codice.
2. Scorri verso il basso fino alla sezione VARIABILI DI AMBIENTE dell'editor di codice. Le variabili di ambiente esistenti sono elencate qui:



3. Per creare nuove variabili di ambiente, scegli il segno più

(+)

):



Le variabili di ambiente rimangono crittografate quando sono elencate nell'editor di codice della console. Se hai abilitato gli helper di crittografia per la crittografia in transito, tali impostazioni rimangono invariate. Per ulteriori informazioni, consulta [Protezione delle variabili di ambiente Lambda](#).

L'elenco delle variabili di ambiente è di sola lettura ed è disponibile solo nella console Lambda. Questo file non è incluso quando scarichi l'archivio di file .zip della funzione e non è possibile aggiungere variabili di ambiente caricando questo file.

## AWS CLI

L'esempio seguente imposta due variabili di ambiente in una funzione denominata `my-function`.

```
aws lambda update-function-configuration \  
  --function-name my-function \  
  --environment "Variables={BUCKET=amzn-s3-demo-bucket,KEY=file.txt}"
```

Quando applichi le variabili di ambiente con il comando `update-function-configuration`, viene sostituito l'intero contenuto della struttura `Variables`. Per mantenere le variabili di ambiente esistenti quando ne aggiungi una nuova, includi tutti i valori esistenti nella richiesta.

Per ottenere la configurazione corrente, utilizza il comando `get-function-configuration`.

```
aws lambda get-function-configuration \  
  --function-name my-function
```

Verrà visualizzato l'output seguente:

```
{  
  "FunctionName": "my-function",  
  "FunctionArn": "arn:aws:lambda:us-east-2:111122223333:function:my-function",  
  "Runtime": "nodejs24.x",  
  "Role": "arn:aws:iam::111122223333:role/lambda-role",  
  "Environment": {  
    "Variables": {  
      "BUCKET": "amzn-s3-demo-bucket",  
      "KEY": "file.txt"  
    }  
  },  
  "RevisionId": "0894d3c1-2a3d-4d48-bf7f-abade99f3c15",  
  ...  
}
```

È possibile passare l'ID di revisione dall'output di `get-function-configuration` come parametro a `update-function-configuration`. Ciò garantisce che i valori non cambino da quando leggi la configurazione a quando la aggiorni.

Per configurare la chiave di crittografia di una funzione, imposta l'opzione `KMSKeyARN`.

```
aws lambda update-function-configuration \  
  --function-name my-function \  
  --kms-key-arn arn:aws:kms:us-east-2:111122223333:key/055efbb4-xmpl-4336-  
ba9c-538c7d31f599
```

## AWS SAM

Per configurare le variabili di ambiente per la tua funzione, puoi utilizzare [AWS Serverless Application Model](#). Aggiorna le proprietà [Environment](#) e [Variables](#) nel tuo file `template.yaml`, quindi esegui [sam deploy](#).

Example `template.yaml`

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Description: An AWS Serverless Application Model template describing your function.  
Resources:  
  my-function:  
    Type: AWS::Serverless::Function  
    Properties:  
      CodeUri: .  
      Description: ''  
      MemorySize: 128  
      Timeout: 120  
      Handler: index.handler  
      Runtime: nodejs24.x  
      Architectures:  
        - x86_64  
      EphemeralStorage:  
        Size: 10240  
      Environment:  
        Variables:  
          BUCKET: amzn-s3-demo-bucket  
          KEY: file.txt  
      # Other function properties...
```

## AWS SDKs

Per gestire le variabili di ambiente utilizzando un AWS SDK, utilizza le seguenti operazioni API.

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

Per ulteriori informazioni, consulta la [documentazione dell'SDK AWS](#) relativa al linguaggio di programmazione preferito.

## Scenario di esempio per le variabili di ambiente

È possibile utilizzare le variabili di ambiente per personalizzare il comportamento delle funzioni nell'ambiente di test e nell'ambiente di produzione. Ad esempio, puoi creare due funzioni con stesso codice ma configurazione diversa. Una funzione si connette a un database di test e l'altra a un database di produzione. In questo caso, puoi utilizzare le variabili di ambiente per passare alla funzione il nome host e altri dettagli di connessione per il database.

Nell'esempio seguente viene illustrato come definire l'host del database e il nome del database come variabili di ambiente.

|              |                            |        |
|--------------|----------------------------|--------|
| ENVIRONMENT  | DEVELOPMENT                | Remove |
| databaseHost | lambdadb                   | Remove |
| databaseName | rd1owwlydynnm5.cuovuyfg087 | Remove |
| Key          | Value                      | Remove |

Se si desidera che l'ambiente di test generi più informazioni di debug rispetto all'ambiente di produzione, è possibile impostare una variabile di ambiente per configurare l'ambiente di test in modo da utilizzare una registrazione più dettagliata o una traccia più dettagliata.

Ad esempio, nell'ambiente di test, è possibile impostare una variabile di ambiente con la chiave LOG\_LEVEL e un valore che indica un livello di log di debug o trace. Nel codice della funzione Lambda, puoi quindi utilizzare questa variabile di ambiente per impostare il livello di log.

I seguenti esempi di codice in Python Node.js illustrano come raggiungere questo obiettivo. Questi esempi presuppongono che la variabile di ambiente abbia un valore DEBUG in Python o in. debug Node.js

## Python

### Example Codice Python per impostare il livello di log

```
import os
import logging

# Initialize the logger
logger = logging.getLogger()

# Get the log level from the environment variable and default to INFO if not set
log_level = os.environ.get('LOG_LEVEL', 'INFO')

# Set the log level
logger.setLevel(log_level)

def lambda_handler(event, context):
    # Produce some example log outputs
    logger.debug('This is a log with detailed debug information - shown only in test environment')
    logger.info('This is a log with standard information - shown in production and test environments')
```

## Node.js (ES module format)

### Example Node.js codice per impostare il livello di log

Questo esempio utilizza la libreria di registrazione winston. Usa npm per aggiungere questa libreria al pacchetto di implementazione della funzione. Per ulteriori informazioni, consulta [the section called “Creazione di un pacchetto di implementazione .zip con dipendenze”](#).

```
import winston from 'winston';

// Initialize the logger using the log level from environment variables, defaulting
// to INFO if not set
const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.json(),
  transports: [new winston.transports.Console()]
});
```

```
export const handler = async (event) => {  
  // Produce some example log outputs  
  logger.debug('This is a log with detailed debug information - shown only in test  
environment');  
  logger.info('This is a log with standard information - shown in production and  
test environment');  
  
};
```

## Recupero delle variabili di ambiente Lambda

Per recuperare le variabili di ambiente nel codice della funzione, utilizza il metodo standard per il linguaggio di programmazione.

### Node.js

```
let region = process.env.AWS_REGION
```

### Python

```
import os  
region = os.environ['AWS_REGION']
```

#### Note

In alcuni casi, potrebbe essere necessario utilizzare il seguente formato:

```
region = os.environ.get('AWS_REGION')
```

### Ruby

```
region = ENV["AWS_REGION"]
```

### Java

```
String region = System.getenv("AWS_REGION");
```



## Go

```
var region = os.Getenv("AWS_REGION")
```

## C#

```
string region = Environment.GetEnvironmentVariable("AWS_REGION");
```

## PowerShell

```
$region = $env:AWS_REGION
```

Lambda memorizza le variabili di ambiente in modo sicuro crittografandole quando inattive. Puoi [configurare Lambda per utilizzare una chiave di crittografia diversa](#), crittografare i valori delle variabili di ambiente sul lato client o impostare le variabili di ambiente in un CloudFormation modello con AWS Secrets Manager

## Variabili di ambiente di runtime definite

I [tempi di esecuzione](#) Lambda impostano diverse variabili di ambiente durante l'inizializzazione. La maggior parte delle variabili di ambiente fornisce informazioni sulla funzione o sul runtime. Le chiavi per queste variabili di ambiente sono riservate e non possono essere impostate nella configurazione della funzione.

### Variabili d'ambiente riservate

- `_HANDLER`: la posizione del gestore configurata nella funzione.
- `_X_AMZN_TRACE_ID` – L'[intestazione di tracciamento X-Ray](#). Questa variabile di ambiente cambia a ogni invocazione.
  - Questa variabile di ambiente non è definita per i OS-only runtime (la famiglia di `provided` runtime). Puoi impostare `_X_AMZN_TRACE_ID` per runtime personalizzati utilizzando l'intestazione di risposta `Lambda-Response-Trace-Id` dal [Chiamata successiva](#).
  - Per le versioni di runtime Java 17 e successive, questa variabile di ambiente non viene utilizzata. Lambda archivia invece le informazioni di tracciamento nella proprietà di sistema `com.amazonaws.xray.traceHeader`.
- `AWS_DEFAULT_REGION`— L'impostazione predefinita Regione AWS in cui viene eseguita la funzione Lambda.

- **AWS\_REGION**— Il Regione AWS luogo in cui viene eseguita la funzione Lambda. Se immesso, questo valore sovrascrive **AWS\_DEFAULT\_REGION**.
  - Per ulteriori informazioni sull'utilizzo delle variabili di Regione AWS ambiente con gli AWS SDK, consulta [AWS Region](#) nella Guida di riferimento per AWS SDK and Tools.
- **AWS\_EXECUTION\_ENV**: l'[identificatore di runtime](#), preceduto da **AWS\_Lambda\_**, ad esempio **AWS\_Lambda\_java8**. Questa variabile di ambiente non è definita per i OS-only runtime (la famiglia di provided runtime).
- **AWS\_LAMBDA\_FUNCTION\_NAME**: il nome della funzione.
- **AWS\_LAMBDA\_FUNCTION\_MEMORY\_SIZE**: la quantità di memoria disponibile per la funzione in MB.
- **AWS\_LAMBDA\_FUNCTION\_VERSION**: la versione della funzione in esecuzione.
- **AWS\_LAMBDA\_INITIALIZATION\_TYPE**— Il tipo di inizializzazione della funzione, che è on-demand, provisioned-concurrency o snap-start, o. lambda-managed-instances Per informazioni, vedere [Configurazione della concorrenza](#) predisposta,, o. [Miglioramento delle prestazioni di avvio con Lambda SnapStart](#) [Istanze Lambda gestite](#)
- **AWS\_LAMBDA\_LOG\_GROUP\_NAME**, **AWS\_LAMBDA\_LOG\_STREAM\_NAME** — Il nome del gruppo e dello stream Amazon CloudWatch Logs per la funzione. Le variabili di **AWS\_LAMBDA\_LOG\_STREAM\_NAME** [ambiente AWS\\_LAMBDA\\_LOG\\_GROUP\\_NAME e non](#) sono disponibili nelle SnapStart funzioni Lambda.
- **AWS\_ACCESS\_KEY**, **AWS\_ACCESS\_KEY\_ID**, **AWS\_SECRET\_ACCESS\_KEY**, **AWS\_SESSION\_TOKEN**: le chiavi di accesso ottenute dal [ruolo di esecuzione](#) della funzione.
- **AWS\_LAMBDA\_RUNTIME\_API**: ([Runtime personalizzato](#)) L'host e la porta dell'[API di runtime](#).
- **LAMBDA\_TASK\_ROOT**: il percorso del codice della funzione Lambda.
- **LAMBDA\_RUNTIME\_DIR**: il percorso delle librerie di runtime.
- **AWS\_LAMBDA\_MAX\_CONCURRENCY**— (solo istanze gestite da Lambda) Il numero massimo di chiamate simultanee che Lambda invierà a un ambiente di esecuzione.
- **AWS\_LAMBDA\_METADATA\_API**— L'indirizzo del [server endpoint dei](#) metadati nel formato (ad esempio,). {ipv4\_address}:{port} 169.254.100.1:9001
- **AWS\_LAMBDA\_METADATA\_TOKEN**— Un token di autenticazione univoco per l'ambiente di esecuzione corrente utilizzato per autenticare le richieste all'endpoint dei metadati. [???](#) Lambda genera questo token automaticamente al momento dell'inizializzazione.

Le seguenti variabili di ambiente aggiuntive non sono riservate e possono essere estese nella configurazione della funzione.

## Variabili d'ambiente non riservate

- **LANG** – Le impostazioni locali del runtime (`en_US.UTF-8`).
- **PATH**: il percorso di esecuzione (`/usr/local/bin:/usr/bin:/bin:/opt/bin`).
- **LD\_LIBRARY\_PATH**: il percorso della libreria di sistema (`/var/lang/lib:/lib64:/usr/lib64:$LAMBDA_RUNTIME_DIR:$LAMBDA_RUNTIME_DIR/lib:$LAMBDA_TASK_ROOT:$LAMBDA_TASK_ROOT/lib:/opt/lib`).
- **NODE\_PATH**— ([Node.js](#)) Il percorso della Node.js libreria (`/opt/nodejs/node12/node_modules:/opt/nodejs/node_modules:$LAMBDA_RUNTIME_DIR/node_modules`).
- **NODE\_OPTIONS**— ([Node.js](#)) Per i Node.js runtime, è possibile utilizzare per **NODE\_OPTIONS** riattivare le funzionalità sperimentali che Lambda disabilita per impostazione predefinita.
- **PYTHONPATH**: ([Python](#)) il percorso della libreria Python (`$LAMBDA_RUNTIME_DIR`).
- **GEM\_PATH** – ([Ruby](#)) Il percorso della libreria Ruby (`$LAMBDA_TASK_ROOT/vendor/bundle/ruby/3.3.0:/opt/ruby/gems/3.3.0`).
- **AWS\_XRAY\_CONTEXT\_MISSING**— Per il X-Ray tracciamento, Lambda lo imposta per evitare di generare errori di `LOG_ERROR` runtime dall'SDK. X-Ray
- **AWS\_XRAY\_DAEMON\_ADDRESS**— Per il X-Ray tracciamento, l'indirizzo IP e la porta del demone. X-Ray
- **AWS\_LAMBDA\_DOTNET\_PREJIT**: ([.NET](#)) imposta questa variabile per abilitare o disabilitare le ottimizzazioni di runtime specifiche di .NET. I valori includono `always`, `never` e `provisioned-concurrency`. Per ulteriori informazioni, consulta [Configurazione della simultaneità fornita per una funzione](#).
- **TZ**: il fuso orario dell'ambiente (`:UTC`). L'ambiente di esecuzione utilizza NTP per sincronizzare l'orologio di sistema.

I valori di esempio mostrati riflettono i runtime più recenti. La presenza di variabili specifiche o dei loro valori può variare nei runtime precedenti.

## Protezione delle variabili di ambiente Lambda

Per proteggere le variabili di ambiente, è possibile utilizzare la crittografia lato server per proteggere i dati inattivi e la crittografia lato client per proteggere i dati in transito.

 Note

Per aumentare la sicurezza del database, si consiglia di utilizzare AWS Secrets Manager al posto delle variabili di ambiente per archiviare le credenziali del database. Per ulteriori informazioni, consulta [Uso dei segreti in Secrets Manager nelle funzioni Lambda](#).

## Sicurezza dei dati inattivi

Lambda fornisce sempre la crittografia lato server dei dati inattivi con un AWS KMS key. Per impostazione predefinita, Lambda utilizza una Chiave gestita da AWS. Se questo comportamento predefinito si adatta al flusso di lavoro, non è necessario impostare altro. Lambda li crea Chiave gestita da AWS nel tuo account e gestisce le autorizzazioni per te. AWS non ti addebita alcun costo per l'utilizzo di questa chiave.

Se preferisci, puoi invece fornire una chiave gestita AWS KMS dal cliente. È possibile eseguire questa operazione per avere il controllo sulla rotazione della chiave KMS o per soddisfare i requisiti dell'organizzazione per la gestione delle chiavi KMS. Quando si utilizza la chiave gestita dal cliente, solo gli utenti del tuo account con accesso alla chiave possono visualizzare o gestire le variabili di ambiente sulla funzione.

Le chiavi gestite dal cliente sono soggette a costi standard AWS KMS . Per ulteriori informazioni, consultare [Prezzi di AWS Key Management Service](#).

## Sicurezza in transito

Per una maggiore sicurezza, è possibile abilitare gli helper per la crittografia in transito, assicurando così che le variabili di ambiente siano crittografate lato client per la protezione in transito.

Per configurare la crittografia per le variabili di ambiente

1. Usa AWS Key Management Service (AWS KMS) per creare qualsiasi chiave gestita dal cliente che Lambda possa utilizzare per la crittografia lato server e lato client. Per ulteriori informazioni, consulta [Creazione di chiavi](#) nella AWS Key Management Service Guida per gli sviluppatori di .
2. Utilizzando la console Lambda, accedere alla pagina Edit environment variables (Modifica delle variabili di ambiente).
  - a. Aprire la pagina [Funzioni](#) della console Lambda.
  - b. Scegliere una funzione.

- c. Scegliere Configurazione, quindi scegliere Variabili di ambiente nella barra di navigazione sinistra.
  - d. Nella sezione Variabili di ambiente, scegliere Modifica.
  - e. Espandere Encryption configuration (Configurazione della crittografia).
3. (Facoltativo) Abilita gli helper di crittografia della console per utilizzare la crittografia lato client per proteggere i dati in transito.
  - a. In Crittografia in transito, scegliere Enable helpers for encryption in transit (Abilita helper per la crittografia in transito).
  - b. Per ogni variabile di ambiente per cui si desidera abilitare gli helper di crittografia della console, scegliere Encrypt (Crittografia) accanto alla variabile di ambiente.
  - c. In AWS KMS key Per crittografare in transito, scegli una chiave gestita dal cliente che hai creato all'inizio di questa procedura.
  - d. Scegliere Execution role policy (Policy del ruolo di esecuzione) e copiare la policy. Questa policy concede l'autorizzazione al ruolo di esecuzione della funzione per decrittare le variabili di ambiente.  
  
Salvare la policy da utilizzare nell'ultima fase di questa procedura.
  - e. Aggiungi il codice alla funzione che decrittica le variabili di ambiente. Per visualizzare un esempio, scegli Decrittica frammento di segreto.
4. (Facoltativo) Specifica la chiave gestita dal cliente per la crittografia dei dati inattivi.
  - a. Scegliere Use a customer master key (Utilizza una chiave master del cliente).
  - b. Scegliere una chiave gestita dal cliente creata all'inizio di questa procedura.
5. Scegli Save (Salva).
6. Impostare le autorizzazioni.

Se stai utilizzando una chiave gestita dal cliente con crittografia lato server, concedi le autorizzazioni a qualsiasi utente o ruolo che desideri possa visualizzare o gestire le variabili di ambiente sulla funzione. Per ulteriori informazioni, consulta [Gestione delle autorizzazioni per la chiave KMS di crittografia lato server](#).

Se si abilita la crittografia lato client per la sicurezza in transito, la funzione richiede l'autorizzazione per chiamare l'operazione API kms : Decrypt. Aggiungere la policy salvata in precedenza in questa procedura al [ruolo di esecuzione](#) della funzione.

## Gestione delle autorizzazioni per la chiave KMS di crittografia lato server

Non sono necessarie AWS KMS autorizzazioni affinché l'utente o il ruolo di esecuzione della funzione utilizzino la chiave di crittografia predefinita. Per utilizzare una chiave gestita dal cliente, occorre l'autorizzazione all'utilizzo della chiave. Lambda utilizza queste autorizzazioni per creare una concessione sulla chiave. Questo consente a Lambda di usarla per la crittografia.

- `kms:ListAliases` – Per visualizzare i tasti nella console Lambda.
- `kms:CreateGrant`, `kms:Encrypt` – Per configurare una chiave gestita dal cliente su una funzione.
- `kms:Decrypt` – Per visualizzare e gestire le variabili di ambiente crittografate con la chiave gestita dal cliente.

Puoi ottenere queste autorizzazioni dalla tua politica di autorizzazioni basata sulle risorse Account AWS o sulla politica delle autorizzazioni basata sulle risorse di una chiave. `ListAliases` è fornito dalle [policy gestite per Lambda](#). Le policy della chiave concedono le autorizzazioni rimanenti agli utenti del gruppo Key users (Utenti chiave).

Gli utenti senza autorizzazioni `Decrypt` possono comunque gestire le funzioni, ma non possono visualizzare le variabili di ambiente o gestirle nella console Lambda. Per impedire a un utente di visualizzare le variabili di ambiente, aggiungere un'istruzione alle autorizzazioni dell'utente che nega l'accesso alla chiave predefinita, a una chiave gestita dal cliente o a tutte le chiavi.

Example Policy IAM: negano l'accesso in base all'ARN della chiave

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Deny",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-2:111122223333:key/3be10e2d-
xmp1-4be4-bc9d-0405a71945cc"
    }
  ]
}
```

```
} ]
```

Per informazioni dettagliate sulla gestione delle autorizzazioni delle chiavi, consulta la pagina [Utilizzo delle policy delle chiavi in AWS KMS](#) nella Guida per gli sviluppatori di AWS Key Management Service .

# Consentire alle funzioni Lambda l'accesso alle risorse in un Amazon VPC

Con Amazon Virtual Private Cloud (Amazon VPC), puoi creare reti private nelle tue Account AWS risorse di hosting come istanze Amazon Elastic Compute Cloud (Amazon EC2), istanze Amazon Relational Database Service (Amazon RDS) e istanze Amazon ElastiCache. Puoi consentire alla tua funzione Lambda di accedere alle risorse ospitate in un Amazon VPC collegando la funzione al VPC tramite le sottoreti private che contengono le risorse. Segui le istruzioni nelle sezioni seguenti per collegare una funzione Lambda a un Amazon VPC utilizzando la console Lambda, la `()` o AWS Command Line Interface AWS CLI AWS SAM.

## Note

Ogni funzione Lambda viene eseguita all'interno di un VPC di proprietà e gestito dal servizio Lambda. Questi VPC vengono gestiti automaticamente da Lambda e non sono visibili ai clienti. La configurazione della funzione per accedere ad altre AWS risorse in un Amazon VPC non ha alcun effetto sul VPC in cui viene eseguita la Lambda-managed funzione.

## Sections

- [Autorizzazioni IAM richieste](#)
- [Collegamento delle funzioni Lambda a un Amazon VPC nel tuo Account AWS](#)
- [Accesso a Internet se collegato a un VPC](#)
- [Supporto IPv6](#)
- [Best practice per utilizzare Lambda con Amazon VPC](#)
- [Informazioni sulle Hyperplane Elastic Network Interface \(ENI\)](#)
- [Utilizzo dei tasti di condizione IAM per le impostazioni VPC](#)
- [Tutorial VPC](#)

## Autorizzazioni IAM richieste

Per collegare una funzione Lambda a un Amazon VPC nel tuo Account AWS, Lambda ha bisogno delle autorizzazioni per creare e gestire le interfacce di rete che utilizza per consentire alla tua funzione di accedere alle risorse nel VPC.



Le interfacce di rete create da Lambda sono note come Hyperplane Elastic Network Interfaces o Hyperplane ENI. Per ulteriori informazioni su queste interfacce di rete, consulta [the section called “Informazioni sulle Hyperplane Elastic Network Interface \(ENI\)”](#).

Puoi assegnare alla tua funzione le autorizzazioni necessarie allegando la policy AWS gestita al ruolo di esecuzione della funzione. [AWSLambdaVPCAccessExecutionRole](#) Quando crei una nuova funzione nella console Lambda e la colleghi a un VPC, Lambda aggiunge automaticamente questa policy di autorizzazioni per tuo conto.

Se preferisci creare la tua policy di autorizzazioni IAM, assicurati di aggiungere tutte le seguenti autorizzazioni e di consentirle su tutt le risorse ("Resource": "\*"):

- ec2: CreateNetworkInterface
- ec2: DescribeNetworkInterfaces
- ec2: DescribeSubnets
- ec2: DeleteNetworkInterface
- ec2: AssignPrivateIpAddresses
- ec2: UnassignPrivateIpAddresses

Tieni presente che il ruolo della tua funzione necessita di queste autorizzazioni solo per creare le interfacce di rete, non per richiamare la tua funzione. Puoi comunque richiamare correttamente la tua funzione quando è collegata a un Amazon VPC, anche se rimuovi queste autorizzazioni dal ruolo di esecuzione della tua funzione.

Per collegare la tua funzione a un VPC, Lambda deve anche verificare le risorse di rete utilizzando il tuo ruolo utente IAM. Assicurati che il tuo ruolo utente disponga delle seguenti autorizzazioni IAM:

- ec2: DescribeSecurityGroups
- ec2: DescribeSubnets
- ec2: DescribeVpcs
- ec2: GetSecurityGroupsForVpc

#### Note

Le autorizzazioni Amazon EC2 concesse al ruolo di esecuzione della funzione vengono utilizzate dal servizio Lambda per collegare la funzione a un VPC. Tuttavia, stai anche

concedendo implicitamente queste autorizzazioni al codice della tua funzione. Ciò significa che il codice della funzione è in grado di effettuare queste chiamate API Amazon EC2. Per ulteriori informazioni su come seguire la best practice di sicurezza, consulta [the section called “Best practice di sicurezza”](#).

## Collegamento delle funzioni Lambda a un Amazon VPC nel tuo Account AWS

Collega la tua funzione a un Amazon VPC nel tuo utilizzando la Account AWS console Lambda, o. AWS CLI AWS SAM Se stai usando AWS CLI or o AWS SAM colleghi una funzione esistente a un VPC usando la console Lambda, assicurati che il ruolo di esecuzione della tua funzione disponga delle autorizzazioni necessarie elencate nella sezione precedente.

Le funzioni Lambda non possono connettersi direttamente a un VPC con la [tenancy dell'istanza dedicata](#). Per connetterti alle risorse in un VPC dedicato, [esegui il peering a un secondo VPC con la tenancy predefinita](#).

### Lambda console

Per collegare una funzione a un Amazon VPC al momento della creazione

1. Apri la pagina [Funzioni](#) della console Lambda e scegli Crea funzione.
2. In Informazioni di base, immettere un nome per la funzione in Nome funzione.
3. Configura le impostazioni del VPC per la funzione effettuando le seguenti operazioni:
  - a. Espandere Advanced settings (Impostazioni avanzate).
  - b. Seleziona Abilita VPC, quindi scegli il VPC a cui desideri collegare la funzione.
  - c. (Facoltativo) Per consentire il [traffico IPv6 in uscita](#), selezionare Consenti il traffico IPv6 per sottoreti dual-stack.
  - d. Scegli le sottoreti e i gruppi di sicurezza per cui creare l'interfaccia di rete. Se hai selezionato Consenti il traffico IPv6 per sottoreti dual-stack, tutte le sottoreti selezionate devono avere un blocco CIDR IPv4 e un blocco CIDR IPv6.

#### Note

Connetti la tua funzione a sottoreti private per accedere alle risorse private. Se la tua funzione ha bisogno dell'accesso a Internet, consulta [the section called](#)

[“Accesso Internet per funzioni del VPC”](#). La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet o a un indirizzo IP pubblico.

#### 4. Scegli Crea funzione.

Per collegare una funzione esistente a un Amazon VPC

1. Apri la [pagina Funzioni](#) della console Lambda e scegli la tua funzione.
2. Scegli la scheda Configurazione, quindi scegli VPC.
3. Scegli Modifica.
4. In VPC, seleziona l'Amazon VPC a cui desideri collegare la funzione.
5. (Facoltativo) Per consentire il [traffico IPv6 in uscita](#), selezionare Consenti il traffico IPv6 per sottoreti dual-stack.
6. Scegli le sottoreti e i gruppi di sicurezza per cui creare l'interfaccia di rete. Se hai selezionato Consenti il traffico IPv6 per sottoreti dual-stack, tutte le sottoreti selezionate devono avere un blocco CIDR IPv4 e un blocco CIDR IPv6.

#### Note

Connetti la tua funzione a sottoreti private per accedere alle risorse private. Se la tua funzione ha bisogno dell'accesso a Internet, consulta [the section called “Accesso Internet per funzioni del VPC”](#). La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet o a un indirizzo IP pubblico.

#### 7. Scegli Save (Salva).

## AWS CLI

Per collegare una funzione a un Amazon VPC al momento della creazione

- Per creare una funzione Lambda e collegarla a un VPC, esegui il comando della CLI `create-function` seguente.

```
aws lambda create-function --function-name my-function \  
--runtime nodejs24.x --handler index.js --zip-file fileb://function.zip \  
--role arn:aws:iam::123456789012:role/lambda-role \  

```

**--vpc-config**

```
Ipv6AllowedForDualStack=true, SubnetIds=subnet-071f712345678e7c8, subnet-07fd123456788a036
```

Specifica le tue sottoreti e i tuoi gruppi di sicurezza e imposta `Ipv6AllowedForDualStack` su `true` o `false` in base al tuo caso d'uso.

Per collegare una funzione esistente a un Amazon VPC

- Per collegare una funzione esistente a un VPC, esegui il comando della CLI `update-function-configuration` seguente.

```
aws lambda update-function-configuration --function-name my-function \
--vpc-config Ipv6AllowedForDualStack=true,
SubnetIds=subnet-071f712345678e7c8, subnet-07fd123456788a036, SecurityGroupIds=sg-0859123
```

Per scollegare la tua funzione da un VPC

- Per scollegare la funzione da un VPC, esegui il comando della CLI `update-function-configuration` seguente con un elenco vuoto di sottoreti VPC e gruppi di sicurezza.

```
aws lambda update-function-configuration --function-name my-function \
--vpc-config SubnetIds=[], SecurityGroupIds=[]
```

## AWS SAM

Per collegare la tua funzione a un VPC

- Per collegare una funzione Lambda a un Amazon VPC, aggiungi la proprietà `VpcConfig` alla definizione della funzione come mostrato nel seguente modello di esempio. Per ulteriori informazioni su questa proprietà, consulta [AWS: :Lambda: :Function VpcConfig](#) nella Guida per l'CloudFormation utente (la AWS SAM `VpcConfig` proprietà viene passata direttamente alla proprietà di una risorsa). `VpcConfig CloudFormation AWS::Lambda::Function`

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Resources:
  MyFunction:
```

```
Type: AWS::Serverless::Function
Properties:
  CodeUri: ./lambda_function/
  Handler: lambda_function.handler
  Runtime: python3.12
  VpcConfig:
    SecurityGroupIds:
      - !Ref MySecurityGroup
    SubnetIds:
      - !Ref MySubnet1
      - !Ref MySubnet2
  Policies:
    - AWSLambdaVPCLambdaAccessExecutionRole

MySecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Security group for Lambda function
    VpcId: !Ref MyVPC

MySubnet1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyVPC
    CidrBlock: 10.0.1.0/24

MySubnet2:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyVPC
    CidrBlock: 10.0.2.0/24

MyVPC:
  Type: AWS::EC2::VPC
  Properties:
    CidrBlock: 10.0.0.0/16
```

Per ulteriori informazioni sulla configurazione del VPC in AWS SAM, consulta [AWS: :EC2: :VPC nella Guida per l'utente](#). CloudFormation

## Accesso a Internet se collegato a un VPC

Per impostazione predefinita, le funzioni Lambda hanno accesso a Internet pubblico. Quando si collega la funzione a un VPC, può accedere solo alle risorse disponibili all'interno di tale VPC. Per concedere alla funzione l'accesso a Internet, è inoltre necessario configurare il VPC per l'accesso a Internet. Per ulteriori informazioni, consulta [the section called “Accesso Internet per funzioni del VPC”](#).

## Supporto IPv6

La funzione può connettersi alle risorse nelle sottoreti VPC dual-stack su IPv6. Per impostazione predefinita, questa opzione è disabilitata. Per consentire il traffico IPv6 in uscita, usa la console o l'opzione `--vpc-config Ipv6AllowedForDualStack=true` con il comando [create-function](#) o [update-function-configuration](#).

### Note

Per consentire il traffico IPv6 in uscita in un VPC, tutte le sottoreti connesse alla funzione devono essere sottoreti dual-stack. Lambda non supporta le connessioni IPv6 in uscita per le IPv6-only sottoreti in un VPC o le connessioni IPv6 in uscita per le funzioni che non sono connesse a un VPC.

È possibile aggiornare il codice della funzione per connettersi esplicitamente alle risorse di sottorete tramite IPv6. Il seguente esempio di Python apre un socket e si connette a un server IPv6.

### Example— Connettiti al server IPv6

```
def connect_to_server(event, context):
    server_address = event['host']
    server_port = event['port']
    message = event['message']
    run_connect_to_server(server_address, server_port, message)

def run_connect_to_server(server_address, server_port, message):
    sock = socket.socket(socket.AF_INET6, socket.SOCK_STREAM, 0)
    try:
        # Send data
        sock.connect((server_address, int(server_port), 0, 0))
        sock.sendall(message.encode())
```

```
    BUFF_SIZE = 4096
    data = b''
    while True:
        segment = sock.recv(BUFF_SIZE)
        data += segment
        # Either 0 or end of data
        if len(segment) < BUFF_SIZE:
            break
    return data
finally:
    sock.close()
```

## Best practice per utilizzare Lambda con Amazon VPC

Per garantire che la configurazione del VPC Lambda soddisfi le linee guida delle best practice, segui i consigli nelle sezioni seguenti.

### Best practice di sicurezza

Per collegare la funzione Lambda a un VPC, devi assegnare al ruolo di esecuzione della funzione una serie di autorizzazioni Amazon EC2. Queste autorizzazioni sono necessarie per creare le interfacce di rete utilizzate dalla funzione per accedere alle risorse nel VPC. Tuttavia, queste autorizzazioni sono anche implicitamente concesse al codice della tua funzione. Ciò significa che il codice della funzione ha l'autorizzazione per effettuare queste chiamate API Amazon EC2.

Per seguire il principio dell'accesso con privilegi minimi, aggiungete una politica di negazione come nell'esempio seguente al ruolo di esecuzione della funzione. Questa policy impedisce al codice della tua funzione di effettuare chiamate alle API di Amazon EC2, pur consentendo al servizio Lambda di gestire le risorse VPC per tuo conto. La policy utilizza la chiave di condizione `lambda:SourceFunctionArn`, che si applica solo alle chiamate API effettuate dal codice della funzione durante l'esecuzione. Per ulteriori informazioni, consulta [the section called “ARN della funzione di origine”](#).

### JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
```

```

    "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSubnets",
        "ec2:DetachNetworkInterface",
        "ec2:AssignPrivateIpAddresses",
        "ec2:UnassignPrivateIpAddresses"
    ],
    "Resource": [ "*" ],
    "Condition": {
        "ArnEquals": {
            "lambda:SourceFunctionArn": [
                "arn:aws:lambda:us-
west-2:123456789012:function:my_function"
            ]
        }
    }
}

```

AWS fornisce gruppi [di sicurezza](#) e elenchi di controllo degli accessi alla [rete \(ACL\)](#) per aumentare la sicurezza nel tuo VPC. I gruppi di sicurezza controllano il traffico in entrata e in uscita per le tue risorse, mentre le liste di controllo degli accessi di rete controllano il traffico in entrata e in uscita per le tue sottoreti. I gruppi di sicurezza forniscono un controllo di accesso sufficiente per la maggior parte delle sottoreti. Se desideri un livello di sicurezza aggiuntivo per il tuo VPC, puoi utilizzare le liste di controllo degli accessi di rete. Per linee guida generali sulle best practice di sicurezza per l'utilizzo di Amazon VPC, consulta [Best practice di sicurezza per il tuo VPC](#) nella Guida per l'utente di Amazon Virtual Private Cloud.

### Best practice sulle prestazioni

Quando colleghi la tua funzione a un VPC, Lambda verifica se esiste una risorsa di rete disponibile (Hyperplane ENI) a cui può connettersi. Le Hyperplane ENI sono associate a una particolare combinazione di gruppi di sicurezza e sottoreti VPC. Se hai già collegato una funzione a un VPC, specificando le stesse sottoreti e gruppi di sicurezza quando colleghi un'altra funzione significa che Lambda può condividere le risorse di rete ed evitare la necessità di creare un nuovo Hyperplane ENI. Per ulteriori informazioni sulle Hyperplane ENI e sul loro ciclo di vita, consulta [the section called "Informazioni sulle Hyperplane Elastic Network Interface \(ENI\)"](#).



## Informazioni sulle Hyperplane Elastic Network Interface (ENI)

Una Hyperplane ENI è una risorsa gestita che funge da interfaccia di rete tra la tua funzione Lambda e le risorse a cui desideri che la tua funzione si connetta. Il servizio Lambda crea e gestisce queste ENI automaticamente quando colleghi la tua funzione a un VPC.

Gli ENI Hyperplane non sono direttamente visibili all'utente e non è necessario configurarli o gestirli. Tuttavia, sapere come funzionano può aiutarti a comprendere il comportamento della tua funzione quando la colleghi a un VPC.

La prima volta che si collega una funzione a un VPC utilizzando una particolare combinazione di sottorete e gruppo di sicurezza, Lambda crea una Hyperplane ENI. Anche altre funzioni del tuo account che utilizzano la stessa combinazione di sottorete e gruppo di sicurezza possono utilizzare questa ENI. Laddove possibile, Lambda riutilizza le ENI esistenti per ottimizzare l'utilizzo delle risorse e ridurre al minimo la creazione di nuove ENI. Ogni Hyperplane ENI ne supporta fino a 65.000. connections/ports. Se il numero di connessioni supera questo limite, Lambda ridimensiona automaticamente il numero di ENI in base al traffico di rete e ai requisiti di simultaneità.

Per le nuove funzioni, mentre Lambda crea un Hyperplane ENI, la funzione rimane nello stato Pending e non è possibile richiamarla. La funzione passa allo stato Attivo solo quando l'Hyperplane ENI è pronta, il che può richiedere diversi minuti. Per le funzioni esistenti, non è possibile eseguire operazioni aggiuntive mirate alla funzione, come la creazione di versioni o l'aggiornamento del codice della funzione, ma è possibile continuare a richiamare le versioni precedenti della funzione.

Come parte della gestione del ciclo di vita ENI, Lambda potrebbe eliminare e ricreare gli ENI per bilanciare il carico del traffico di rete tra gli ENI o per risolvere i problemi riscontrati nei controlli sanitari di ENI. In aggiunta, se una funzione Lambda rimane inattiva per 14 giorni, Lambda recupera le Hyperplane ENI non utilizzate e imposta lo stato della funzione su Inactive. L'invocazione successiva avrà esito negativo e la funzione entrerà di nuovo nello stato in sospeso fino a quando Lambda non completa la creazione o l'allocazione di una Hyperplane ENI. Ti consigliamo di non basare il tuo design sulla persistenza delle ENI.

Quando si aggiorna una funzione per rimuoverne la configurazione VPC, Lambda richiede fino a 20 minuti per eliminare l'Hyperplane ENI collegato. Lambda elimina l'ENI solo se nessun'altra funzione (o versione della funzione pubblicata) utilizza l'Hyperplane ENI.

Per eliminare l'Hyperplane ENI, Lambda si basa sulle autorizzazioni del [ruolo di esecuzione](#) della funzione. Se elimini il ruolo di esecuzione prima che Lambda abbia eliminato l'Hyperplane ENI, Lambda non sarà più in grado di eliminare l'Hyperplane ENI. Puoi eseguire manualmente l'eliminazione.

## Utilizzo dei tasti di condizione IAM per le impostazioni VPC

Puoi utilizzare le chiavi di Lambda-specific condizione per le impostazioni VPC per fornire controlli di autorizzazione aggiuntivi per le tue funzioni Lambda. Ad esempio, è possibile richiedere che tutte le funzioni dell'organizzazione siano connesse a un VPC. È inoltre possibile specificare le sottoreti e i gruppi di sicurezza che gli utenti della funzione possono e non possono utilizzare.

Lambda supporta inoltre le seguenti chiavi di condizione nelle policy IAM:

- `lambda: VpcIds` — Consenti o nega uno o più VPC.
- `lambda: SubnetIds` — Consenti o nega una o più sottoreti.
- `lambda: SecurityGroupIds` — Consenti o nega uno o più gruppi di sicurezza.

L'API Lambda opera [CreateFunction](#) e [UpdateFunctionConfiguration](#) supporta queste chiavi di condizione. Per ulteriori informazioni sull'uso delle chiavi di condizione nelle policy IAM, consulta [elementi della policy IAM JSON: condizione](#) nella Guida per l'utente di IAM.

### Tip

Se la funzione include già una configurazione VPC da una richiesta API precedente, è possibile inviare una richiesta `UpdateFunctionConfiguration` senza la configurazione VPC.

## Policy di esempio con chiavi di condizione per le impostazioni VPC

Negli esempi seguenti viene illustrato come utilizzare le chiavi di condizione per le impostazioni VPC. Dopo aver creato un'istruzione delle policy con le restrizioni desiderate, aggiungere l'istruzione delle policy per l'utente o il ruolo di destinazione.

Assicurati che gli utenti implementino solo funzioni VPC-connected

Per garantire che tutti gli utenti implementino solo VPC-connected funzioni, puoi negare le operazioni di creazione e aggiornamento delle funzioni che non includono un ID VPC valido.

Si noti che l'ID VPC non è un parametro di input per la richiesta `CreateFunction` o `UpdateFunctionConfiguration`. Lambda recupera il valore dell'ID VPC in base ai parametri della sottorete e del gruppo di sicurezza.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceVPCFunction",
      "Action": [
        "lambda:CreateFunction",
        "lambda:UpdateFunctionConfiguration"
      ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "Null": {
          "lambda:VpcIds": "true"
        }
      }
    }
  ]
}
```

Negare agli utenti l'accesso a specifici VPC, sottoreti o gruppi di sicurezza

Per negare agli utenti l'accesso a specifici VPC, utilizzare `StringEquals` per verificare il valore della condizione `lambda:VpcIds`. Nell'esempio seguente viene negato agli utenti l'accesso a `vpc-1` e `vpc-2`.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceOutOfVPC",
      "Action": [
        "lambda:CreateFunction",
        "lambda:UpdateFunctionConfiguration"
      ],

```

```

        "Effect": "Deny",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "lambda:VpcIds": [
                    "vpc-1",
                    "vpc-2"
                ]
            }
        }
    ]
}

```

Per negare agli utenti l'accesso a subnet specifiche, utilizzare `StringEquals` per verificare il valore della condizione `lambda:SubnetIds`. Nell'esempio seguente viene negato agli utenti l'accesso a `subnet-1` e `subnet-2`.

```

{
    "Sid": "EnforceOutOfSubnet",
    "Action": [
        "lambda:CreateFunction",
        "lambda:UpdateFunctionConfiguration"
    ],
    "Effect": "Deny",
    "Resource": "*",
    "Condition": {
        "ForAnyValue:StringEquals": {
            "lambda:SubnetIds": ["subnet-1", "subnet-2"]
        }
    }
}

```

Per negare agli utenti l'accesso a specifici gruppi di sicurezza, utilizzare `StringEquals` per verificare il valore della condizione `lambda:SecurityGroupIds`. Nell'esempio seguente viene negato agli utenti l'accesso a `sg-1` e `sg-2`.

```

{
    "Sid": "EnforceOutOfSecurityGroups",
    "Action": [

```

```

        "lambda:CreateFunction",
        "lambda:UpdateFunctionConfiguration"
    ],
    "Effect": "Deny",
    "Resource": "*",
    "Condition": {
        "ForAnyValue:StringEquals": {
            "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
        }
    }
}
]
}

```

Consenti agli utenti di creare e aggiornare funzioni con impostazioni VPC specifiche

Per consentire agli utenti di accedere a specifici VPC, utilizzare `StringEquals` per verificare il valore della condizione `lambda:VpcIds`. L'esempio seguente consente agli utenti di accedere a `vpc-1` e `vpc-2`.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceStayInSpecificVpc",
      "Action": [
        "lambda:CreateFunction",
        "lambda:UpdateFunctionConfiguration"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "lambda:VpcIds": [
            "vpc-1",
            "vpc-2"
          ]
        }
      }
    }
  ]
}

```

```
    }  
  ]  
}
```

Per consentire agli utenti di accedere a sottoreti specifiche, utilizzare `StringEquals` per verificare il valore della condizione `lambda:SubnetIds`. L'esempio seguente consente agli utenti di accedere a `subnet-1` e `subnet-2`.

```
{  
  "Sid": "EnforceStayInSpecificSubnets",  
  "Action": [  
    "lambda:CreateFunction",  
    "lambda:UpdateFunctionConfiguration"  
  ],  
  "Effect": "Allow",  
  "Resource": "*",  
  "Condition": {  
    "ForAllValues:StringEquals": {  
      "lambda:SubnetIds": ["subnet-1", "subnet-2"]  
    }  
  }  
}
```

Per consentire agli utenti di accedere a gruppi di sicurezza specifici, utilizzare `StringEquals` per verificare il valore della condizione `lambda:SecurityGroupIds`. L'esempio seguente consente agli utenti di accedere a `sg-1` e `sg-2`.

```
{  
  "Sid": "EnforceStayInSpecificSecurityGroup",  
  "Action": [  
    "lambda:CreateFunction",  
    "lambda:UpdateFunctionConfiguration"  
  ],  
  "Effect": "Allow",  
  "Resource": "*",  
  "Condition": {  
    "ForAllValues:StringEquals": {  
      "lambda:SecurityGroupIds": ["sg-1", "sg-2"]  
    }  
  }  
}
```

```
}  
]  
}
```

## Tutorial VPC

Nelle esercitazioni seguenti è possibile connettere una funzione Lambda alle risorse del VPC.

- [Tutorial: Utilizzo di una funzione Lambda per l'accesso ad Amazon RDS in un VPC Amazon](#)
- [Tutorial: Configurazione di una funzione Lambda per accedere ad Amazon in un Amazon VPC ElastiCache](#)

Per connettere la tua funzione a risorse in un VPC che appartiene a un altro, consulta. Account AWS [the section called “Collegare funzioni alle risorse di un altro account”](#)

# Consentire alle funzioni Lambda di accedere a una risorsa in un Amazon VPC in un altro account

Puoi consentire alla tua AWS Lambda funzione di accedere a una risorsa in un Amazon VPC in Amazon Virtual Private Cloud gestito da un altro account, senza esporre nessuno dei due VPC a Internet. Con questo modello di accesso, puoi condividere i dati con altre organizzazioni utilizzando AWS. Utilizzando questo modello di accesso, puoi condividere dati tra VPC con un livello di sicurezza e prestazioni maggiore rispetto a Internet. Configura la tua funzione Lambda per utilizzare una connessione peering Amazon VPC per accedere a queste risorse.

## Warning

Quando consenti l'accesso tra account o VPC, verifica che il tuo piano soddisfi i requisiti di sicurezza delle rispettive organizzazioni che gestiscono questi account. Seguire le istruzioni contenute in questo documento influisce sul livello di sicurezza delle risorse.

In questo tutorial, collegherai due account con una connessione peering tramite IPv4. Si configura una funzione Lambda che non è già connessa a un Amazon VPC. Viene configurata la risoluzione DNS per connettere la funzione a risorse che non forniscono IP statici. Per adattare queste istruzioni ad altri scenari di peering, consulta la [Guida al peering VPC](#).

## Prerequisiti

Per consentire a una funzione Lambda di accedere a una risorsa in un altro account, devi avere:

- Una funzione Lambda configurata per l'autenticazione e la successiva lettura dalla risorsa.
- Una risorsa in un altro account, ad esempio un cluster Amazon RDS, disponibile tramite Amazon VPC.
- Le credenziali per l'account della funzione Lambda e l'account della risorsa. Se non sei autorizzato a utilizzare l'account della tua risorsa, contatta un utente autorizzato per preparare quell'account.
- L'autorizzazione per creare e aggiornare un VPC (e le risorse del Amazon VPC di supporto) da associare alla funzione Lambda.
- L'autorizzazione per aggiornare il ruolo di esecuzione e la configurazione VPC per la funzione Lambda.
- L'autorizzazione per creare una connessione peering VPC nell'account della funzione Lambda.



- L'autorizzazione per accettare una connessione peering VPC nell'account della risorsa.
- L'autorizzazione per aggiornare la configurazione del VPC della risorsa (e le risorse Amazon VPC di supporto).
- Autorizzazione per richiamare la funzione Lambda.

## Creare un Amazon VPC nell'account della tua funzione

Crea un Amazon VPC, le sottoreti, le tabelle di routing e un gruppo di sicurezza nell'account della tua funzione Lambda.

Come creare un VPC, sottoreti e altre risorse VPC tramite la console

1. Apri la console Amazon VPC all'indirizzo <https://console.aws.amazon.com/vpc/>.
2. Nel pannello di controllo, scegli Crea VPC.
3. Per il blocco CIDR IPv4, fornisci un blocco CIDR privato. Il blocco CIDR non deve sovrapporsi ai blocchi utilizzati nel VPC della risorsa. Non scegliere un blocco utilizzato dal VPC delle risorse per assegnare gli IP alle risorse o un blocco già definito nelle tabelle di routing nel VPC delle risorse. Per ulteriori informazioni sulla definizione dei blocchi CIDR appropriati, consulta [Blocchi CIDR VPC](#).
4. Scegli Personalizza AZ.
5. Seleziona le stesse AZ della tua risorsa.
6. Per Numero di sottoreti pubbliche, scegli 0.
7. Per VPC endpoints (Endpoint VPC), scegli None (Nessuno).
8. Seleziona Crea VPC.

## Concedere le autorizzazioni VPC al ruolo di esecuzione della funzione

[AWSLambdaVPCAccessExecutionRole](#) Collegati al ruolo di esecuzione della tua funzione per consentirle di connettersi ai VPC.

Per concedere le autorizzazioni VPC al ruolo di esecuzione della funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione .
3. Scegliere Configuration (Configurazione).

4. Seleziona Autorizzazioni.
5. In Nome del ruolo, scegli il ruolo di esecuzione.
6. Nella sezione Policy di autorizzazioni, seleziona Aggiungi autorizzazioni.
7. Dall'elenco a discesa scegli Collega policy.
8. Nella casella di ricerca immetti `AWSLambdaVPCAccessExecutionRole`.
9. Seleziona la casella di controllo sulla sinistra del nome della policy.
10. Scegli Add Permissions (Aggiungi autorizzazioni).

Per collegare la tua funzione al tuo Amazon VPC

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione .
3. Scegli la scheda Configurazione, quindi scegli VPC.
4. Scegli Modifica.
5. In VPC, seleziona il tuo VPC.
6. In Sottoreti, scegli le tue sottoreti.
7. Per Gruppi di sicurezza, scegli il gruppo di sicurezza predefinito del VPC.
8. Selezionare Salva.

## Creare una richiesta di connessione peering VPC

Crea una richiesta di connessione peering VPC dal VPC della tua funzione (il VPC richiedente) al VPC della tua risorsa (il VPC accettante).

Per richiedere una connessione peering VPC al VPC della funzione

1. Apri la <https://console.aws.amazon.com/vpc/>.
2. Nel riquadro di navigazione, seleziona Connessioni peering.
3. Scegli Create peering connection (Crea connessione peering).
4. Per ID VPC (richiedente), seleziona il VPC della tua funzione.
5. In ID account immetti l'ID dell'account della risorsa.
6. Per ID VPC (accettante), inserisci il VPC della tua risorsa.

## Preparare l'account della tua risorsa

Per creare la connessione peering e preparare il VPC della risorsa all'utilizzo della connessione, accedi all'account della risorsa con un ruolo che detiene le autorizzazioni elencate nei prerequisiti. I passaggi per accedere potrebbero essere diversi in base alla protezione dell'account. Per ulteriori informazioni su come accedere a un AWS account, consulta la Guida per l'[AWS Sign-in utente](#). Nell'account della risorsa, completa le seguenti procedure.

Per accettare una richiesta di connessione peering VPC.

1. Apri la <https://console.aws.amazon.com/vpc/>.
2. Nel riquadro di navigazione, seleziona Connessioni peering.
3. Seleziona la connessione peering VPC in attesa (lo stato è pending-acceptance).
4. Scegli Azioni
5. Dal menù a discesa, scegli Accetta richiesta.
6. Quando viene chiesta la conferma, seleziona Accetta richiesta.
7. Scegli Modifica subito le tabelle di routing per aggiungere una route alla tabella di routing principale del VPC in modo da poter inviare e ricevere traffico attraverso la connessione peering.

Ispeziona le tabelle di routing per il VPC della risorsa. Il percorso generato da Amazon VPC potrebbe non stabilire la connettività a seconda di come è configurato il VPC della risorsa. Verifica la presenza di conflitti tra la nuova route e la configurazione esistente per il VPC. Per ulteriori informazioni sulla risoluzione dei problemi, consulta [Risoluzione dei problemi di una connessione peering VPC](#) nella Guida al peering VPC di Amazon Virtual Private Cloud Guide.

Per aggiornare il gruppo di sicurezza per la risorsa

1. Apri la <https://console.aws.amazon.com/vpc/>.
2. Nel pannello di navigazione, seleziona Gruppi di sicurezza.
3. Seleziona il gruppo di sicurezza per la tua risorsa.
4. Scegli Azioni.
5. Dall'elenco a discesa, scegli Modifica regole in entrata.
6. Scegli Aggiungi regola.
7. In Source, inserisci l'ID dell'account della funzione e l'ID del gruppo di sicurezza, separati da una barra (ad esempio, 111122223333/sg-1a2b3c4d).

8. Scegli Edit outbound rules (Modifica regole in uscita).
9. Verifica se il traffico in uscita è limitato. Le impostazioni VPC predefinite autorizzano tutto il traffico in uscita. Se il traffico in uscita è limitato, continua con il passaggio successivo.
10. Scegli Aggiungi regola.
11. In Destinazione, inserisci l'ID dell'account e l'ID del gruppo di sicurezza della tua funzione, separati da una barra (ad esempio, -1a2b3c4d). 111122223333/sg
12. Scegliere Salva regole.

Per abilitare la risoluzione DNS per la connessione peering

1. Apri la <https://console.aws.amazon.com/vpc/>.
2. Nel riquadro di navigazione, seleziona Connessioni peering.
3. Seleziona la connessione peering.
4. Scegli Azioni.
5. Seleziona Modifica impostazioni DNS.
6. In Risoluzione DNS accettante, seleziona Consenti al VPC richiedente di risolvere il DNS degli host VPC accettanti su IP privato.
7. Scegli Save changes (Salva modifiche).

## Aggiornare la configurazione VPC nell'account della tua funzione

Accedi all'account della funzione, quindi aggiorna la configurazione VPC.

Per aggiungere una route per la connessione peering VPC

1. Apri la <https://console.aws.amazon.com/vpc/>.
2. Nel riquadro di navigazione, seleziona Tabelle di routing.
3. Seleziona la casella di controllo accanto al nome della tabella di routing per la sottorete associata alla tua funzione.
4. Scegli Azioni.
5. Selezionare Modifica route.
6. Seleziona Aggiungi route.
7. Per Destinazione, inserisci il blocco CIDR per il VPC della risorsa.

8. Per Destinazione seleziona la connessione peering VPC.
9. Scegli Save changes (Salva modifiche).

Per ulteriori informazioni sulle considerazioni che potresti riscontrare durante l'aggiornamento delle tabelle di percorso, consulta [Update your route tables for a VPC peering connection](#).

Per aggiornare il gruppo di sicurezza per la tua funzione Lambda

1. Apri la <https://console.aws.amazon.com/vpc/>.
2. Nel pannello di navigazione, seleziona Gruppi di sicurezza.
3. Scegli Azioni.
4. Scegliere Edit inbound rules (Modifica regole in entrata).
5. Scegli Aggiungi regola.
6. Per Fonte, inserisci l'ID dell'account della risorsa e l'ID del gruppo di sicurezza, separati da una barra (ad esempio, -1a2b3c4d). 111122223333/sg
7. Scegliere Salva regole.

Per abilitare la risoluzione DNS per la connessione peering

1. Apri la <https://console.aws.amazon.com/vpc/>.
2. Nel riquadro di navigazione, seleziona Connessioni peering.
3. Seleziona la connessione peering.
4. Scegli Azioni.
5. Seleziona Modifica impostazioni DNS.
6. In Risoluzione DNS richiedente, seleziona Consenti al VPC accettante di risolvere il DNS degli host VPC richiedenti su IP privato.
7. Scegli Save changes (Salva modifiche).

## Test della funzione

Per creare un evento di test e controllare l'output della funzione

1. Nel riquadro Origine del codice, scegli Test.
2. Seleziona Crea nuovo evento.

3. Nel pannello JSON evento, sostituisci i valori predefiniti con un input appropriato per la tua funzione Lambda.
4. Scegli Richiama .
5. Nella scheda Risultati di esecuzione, conferma che la sezione Risposta contenga l'output previsto.

Inoltre, puoi controllare i log della tua funzione per verificare che siano quelli previsti.

Per visualizzare i record di chiamata della funzione in Logs CloudWatch

1. Selezionare la scheda Monitor (Monitora).
2. Scegli Visualizza CloudWatch registri.
3. Nella scheda Flussi di log, scegli il flusso di log per l'invocazione della funzione.
4. Conferma che i log siano quelli che ti aspetti.

## Abilita l'accesso a Internet per le funzioni VPC-connected Lambda

Per impostazione predefinita, le funzioni Lambda vengono eseguite in un Lambda-managed VPC con accesso a Internet. Per accedere alle risorse in un VPC nel tuo account, puoi aggiungere una configurazione VPC a una funzione. Ciò limita la funzione alle risorse all'interno di quel VPC, a meno che il VPC non abbia accesso a Internet. Questa pagina spiega come fornire l'accesso a Internet alle funzioni VPC-connected Lambda.

Non ho ancora un VPC

### Creazione del VPC

Il flusso di lavoro Crea VPC crea tutte le risorse VPC necessarie per una funzione Lambda per accedere alla rete Internet pubblica da una sottorete privata, incluse sottoreti, gateway NAT, gateway Internet e voci della tabella di routing.

Per creare il VPC

1. Apri la console Amazon VPC all'indirizzo <https://console.aws.amazon.com/vpc/>.
2. Nel pannello di controllo, scegli Crea VPC.
3. Per Risorse da creare, scegli VPC e altro.
4. Configurazione del VPC
  - a. Per Name tag auto-generation (Generazione automatica di tag nome), immetti un nome per il VPC.
  - b. Per Blocco CIDR IPv4, mantieni il suggerimento predefinito o, in alternativa, inserisci il blocco CIDR richiesto dall'applicazione o dalla rete.
  - c. Se l'applicazione comunica utilizzando indirizzi IPv6, scegli il blocco CIDR IPv6, il blocco CIDR IPv6. Amazon-provided
5. Configurazione delle sottoreti
  - a. In Numero di zone di disponibilità, scegli 2. Per una disponibilità elevata consigliamo almeno due zone di disponibilità.
  - b. Per Number of public subnets (Numero di sottoreti pubbliche), scegli 2.
  - c. Per Number of private subnets (Numero di sottoreti private), scegli 2.
  - d. Puoi mantenere il blocco CIDR predefinito per la sottorete pubblica o, in alternativa, espandere Personalizza blocchi CIDR della sottorete e inserire un blocco CIDR. Per ulteriori informazioni, consulta [Blocchi CIDR della sottorete](#).

6. Per Gateway NAT, scegli 1 per AZ per migliorare la resilienza.
7. Per Gateway Internet solo in uscita, scegli Sì se hai scelto di includere un blocco CIDR IPv6.
8. Per Endpoint VPC, mantieni il valore predefinito, Gateway S3. Questa opzione non prevede alcun costo. Per ulteriori informazioni, consulta [Tipi di endpoint VPC per Amazon S3](#).
9. Per Opzioni DNS, mantieni le impostazioni predefinite.
10. Seleziona Crea VPC.

## Configura la funzione Lambda

Per configurare un VPC quando si crea una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli Crea funzione.
3. In Informazioni di base, immettere un nome per la funzione in Nome funzione.
4. Espandere Advanced settings (Impostazioni avanzate).
5. Seleziona Abilita VPC, quindi scegli un VPC.
6. (Facoltativo) Per consentire il [traffico IPv6 in uscita](#), selezionare Consenti il traffico IPv6 per sottoreti dual-stack.
7. Per Sottoreti, seleziona tutte le sottoreti private. Le sottoreti private possono accedere a Internet attraverso un gateway NAT. La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet.

### Note

Se hai selezionato Consenti il traffico IPv6 per sottoreti dual-stack, tutte le sottoreti selezionate devono avere un blocco CIDR IPv4 e un blocco CIDR IPv6.

8. Per Gruppi di sicurezza, seleziona un gruppo di sicurezza che consenta il traffico in uscita.
9. Scegli Crea funzione.

Lambda crea automaticamente un ruolo di esecuzione con la policy gestita.

[AWSLambdaVPCAccessExecutionRole](#) AWS Le autorizzazioni in questa policy sono necessarie solo per creare interfacce di rete elastiche per la configurazione VPC, non per richiamare la funzione. Per applicare le autorizzazioni con privilegi minimi, puoi rimuovere la



`AWSLambdaVPCAccessExecutionRole` policy dal tuo ruolo di esecuzione dopo aver creato la funzione e la configurazione del VPC. Per ulteriori informazioni, consulta [Autorizzazioni IAM richieste](#).

Per configurare un VPC per una funzione esistente

Per aggiungere una configurazione VPC a una funzione esistente, il ruolo di esecuzione della funzione deve disporre dell'[autorizzazione per creare e gestire interfacce di rete elastiche](#). La policy [AWSLambdaVPCAccessExecutionRole](#) AWS gestita include le autorizzazioni richieste. Per applicare le autorizzazioni con privilegi minimi, puoi rimuovere la `AWSLambdaVPCAccessExecutionRole` policy dal tuo ruolo di esecuzione dopo aver creato la configurazione VPC.

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegli la scheda Configurazione, quindi scegli VPC.
4. In VPC, scegli Modifica.
5. Seleziona il VPC
6. (Facoltativo) Per consentire il [traffico IPv6 in uscita](#), selezionare Consenti il traffico IPv6 per sottoreti dual-stack.
7. Per Sottoreti, seleziona tutte le sottoreti private. Le sottoreti private possono accedere a Internet attraverso un gateway NAT. La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet.

 Note

Se hai selezionato Consenti il traffico IPv6 per sottoreti dual-stack, tutte le sottoreti selezionate devono avere un blocco CIDR IPv4 e un blocco CIDR IPv6.

8. Per Gruppi di sicurezza, seleziona un gruppo di sicurezza che consenta il traffico in uscita.
9. Scegli Save (Salva).

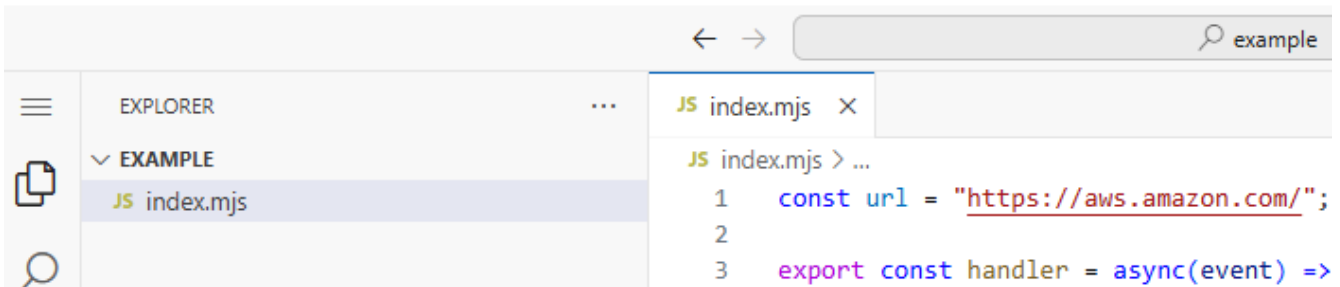
## Test della funzione

Utilizza il seguente codice di esempio per confermare che la tua VPC-connected funzione può raggiungere la rete Internet pubblica. In caso di successo, il codice restituisce un codice di stato 200. Se non ha esito positivo, la funzione scade.

## Node.js

1. Nel riquadro Codice sorgente della console Lambda, incolla il codice seguente nel file `index.mjs`. La funzione effettua una richiesta HTTP GET a un endpoint pubblico e restituisce il codice di risposta HTTP per verificare se la funzione ha accesso alla rete Internet pubblica.

### Code source [Info](#)

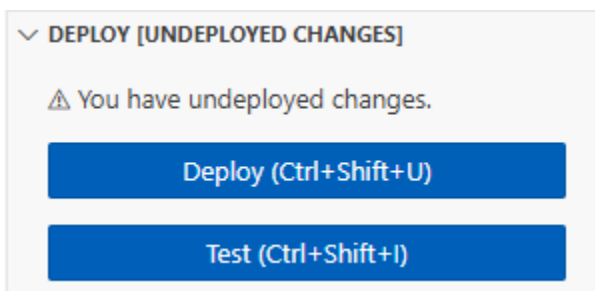


### Example— Richiesta HTTP con async/await

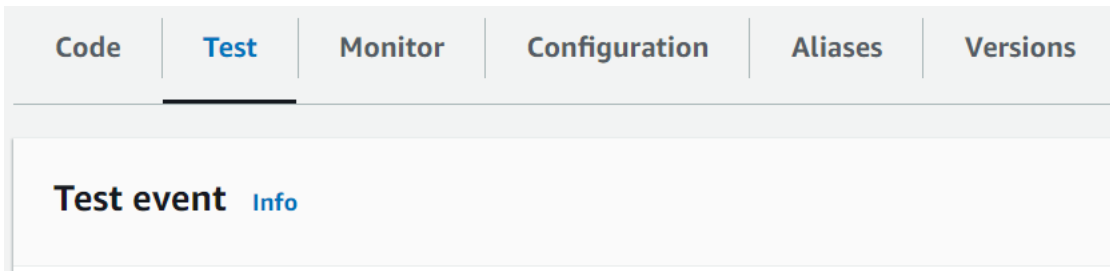
```
const url = "https://aws.amazon.com/";

export const handler = async(event) => {
  try {
    const res = await fetch(url);
    console.info("status", res.status);
    return res.status;
  }
  catch (e) {
    console.error(e);
    return 500;
  }
};
```

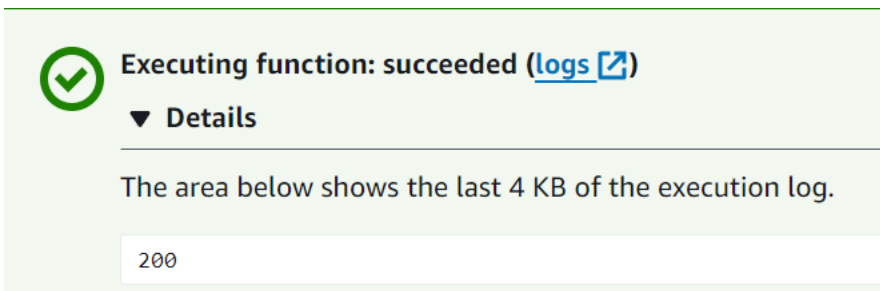
2. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



3. Seleziona la scheda Test.



4. Scegli Test (Esegui test).
5. La funzione restituisce un codice di stato 200. Ciò significa che la funzione dispone di un accesso a Internet in uscita.



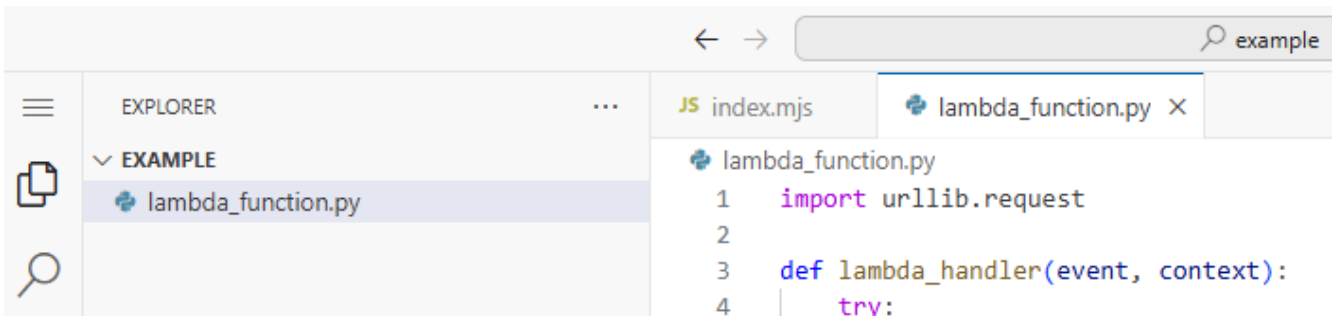
Se la funzione non riesce a raggiungere la rete Internet pubblica, ti verrà mostrato un messaggio di errore come questo:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12j1c-640a-8157-0249-9be825c2y110
Task timed out after 3.01 seconds"
}
```

## Python

1. Nel riquadro Codice sorgente della console Lambda, incolla il seguente codice seguente nel file `lambda_function.py`. La funzione effettua una richiesta HTTP GET a un endpoint pubblico e restituisce il codice di risposta HTTP per verificare se la funzione ha accesso alla rete Internet pubblica.

## Code source [Info](#)



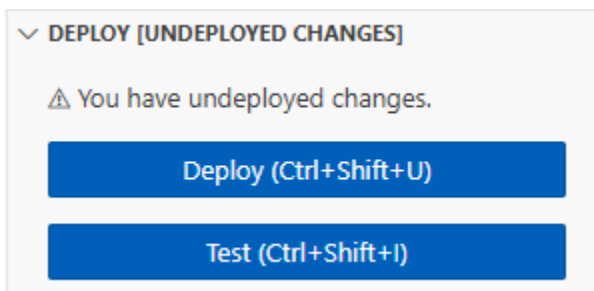
```

import urllib.request

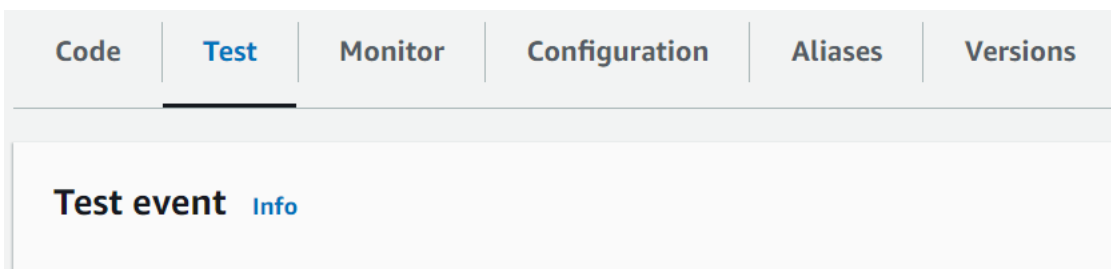
def lambda_handler(event, context):
    try:
        response = urllib.request.urlopen('https://aws.amazon.com')
        status_code = response.getcode()
        print('Response Code:', status_code)
        return status_code
    except Exception as e:
        print('Error:', e)
        raise e

```

2. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:

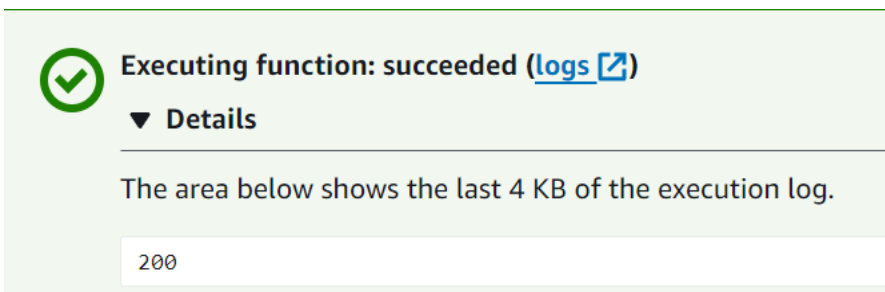


3. Seleziona la scheda Test.



4. Scegli Test (Esegui test).

- La funzione restituisce un codice di stato 200. Ciò significa che la funzione dispone di un accesso a Internet in uscita.



Se la funzione non riesce a raggiungere la rete Internet pubblica, ti verrà mostrato un messaggio di errore come questo:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
Task timed out after 3.01 seconds"
}
```

Ho già un VPC

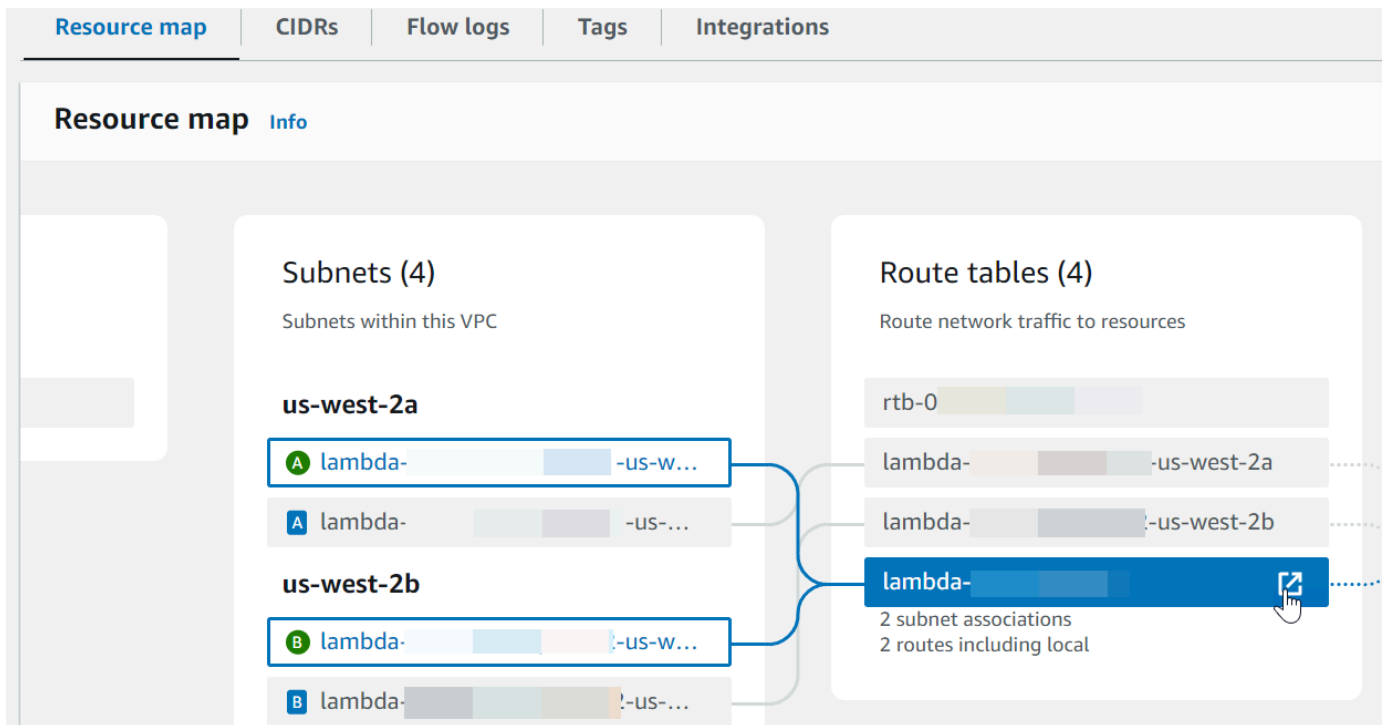
Se disponi già di un VPC ma devi configurare l'accesso pubblico a Internet per una funzione Lambda, completa questa procedura. Questa procedura presuppone che il VPC abbia almeno due sottoreti. Se non disponi di due sottoreti, consulta [Creare una sottorete](#) nella Guida per l'utente di Amazon VPC.

Verificare la configurazione della tabella di routing

- Apri la console Amazon VPC all'indirizzo <https://console.aws.amazon.com/vpc/>.
- Scegli l'ID VPC.

| Your VPCs (3) <a href="#">Info</a>  |                 |                       |           |  |
|-------------------------------------|-----------------|-----------------------|-----------|--|
| <input type="text" value="Search"/> |                 |                       |           |  |
| <input type="checkbox"/>            | Name            | VPC ID                | State     |  |
| <input type="checkbox"/>            | -               | <a href="#">vpc-2</a> | Available |  |
| <input type="checkbox"/>            | lambda-test-vpc | <a href="#">vpc-0</a> | Available |  |

- Scorri verso il basso fino alla sezione Mappa delle risorse. Prendi nota delle mappature delle tabelle di routing. Apri ogni tabella di routing mappata a una sottorete.



4. Scorri verso il basso fino alla scheda Route. Esamina i routing per determinare se il tuo VPC ha entrambe le seguenti tabelle di routing. Ciascuno di questi requisiti deve essere soddisfatto da una tabella di routing separata.
  - Internet-bound il traffico ( $0.0.0.0/0$  per IPv4,  $::/0$  per IPv6) viene indirizzato a un gateway Internet (). `igw-xxxxxxxxxx` Ciò significa che la sottorete associata alla tabella di routing è una sottorete pubblica.

#### Note

Se la sottorete non ha un blocco CIDR IPv6, sarà visualizzata solo la route IPv4 ( $0.0.0.0/0$ ).

## Example tabella di routing della sottorete pubblica

| Routes                                                                    | Subnet associations   | Edge associations | Route propagation | Tags |
|---------------------------------------------------------------------------|-----------------------|-------------------|-------------------|------|
| <b>Routes (4)</b> <div> <input type="text" value="Filter routes"/> </div> |                       |                   |                   |      |
| Destination                                                               | Target                | Status            |                   |      |
| ::/0                                                                      | <a href="#">igw-0</a> | Active            |                   |      |
| ::/56                                                                     | local                 | Active            |                   |      |
| 0.0.0.0/0                                                                 | <a href="#">igw-0</a> | Active            |                   |      |
| /16                                                                       | local                 | Active            |                   |      |

- Internet-bound il traffico per IPv4 (0.0.0.0/0) viene indirizzato a un gateway NAT (nat-xxxxxxxxxx) associato a una sottorete pubblica. Ciò significa che la sottorete è una sottorete privata che può accedere a Internet tramite il gateway NAT.

### Note

Se la sottorete ha un blocco CIDR IPv6, la tabella di routing deve instradare il traffico IPv6 vincolato a Internet (::/0) a un gateway Internet di sola uscita (eigw-xxxxxxxxxx). Se la sottorete non ha un blocco CIDR IPv6, sarà visualizzata solo la route IPv4 (0.0.0.0/0).

## Example tabella di routing della sottorete privata

| Routes                                                                    | Subnet associations    | Edge associations | Route propagation | Tags |
|---------------------------------------------------------------------------|------------------------|-------------------|-------------------|------|
| <b>Routes (4)</b> <div> <input type="text"/> Filter routes         </div> |                        |                   |                   |      |
| Destination                                                               | Target                 | Status            |                   |      |
| ::/0                                                                      | <a href="#">eigw-0</a> | ✓ Active          |                   |      |
| ::/56                                                                     | local                  | ✓ Active          |                   |      |
| 0.0.0.0/0                                                                 | <a href="#">nat-0</a>  | ✓ Active          |                   |      |
| /16                                                                       | local                  | ✓ Active          |                   |      |

- Ripeti il passaggio precedente fino a quando non avrai esaminato ogni tabella di routing associata a una sottorete nel tuo VPC e avrai confermato di avere una tabella di routing con un gateway Internet e una tabella di routing con un gateway NAT.

Se non disponi di due tabelle di routing, una con una route verso un gateway Internet e una con una route verso un gateway NAT, segui questi passaggi per creare le risorse mancanti e le voci della tabella di routing.

### Creazione di una tabella di routing

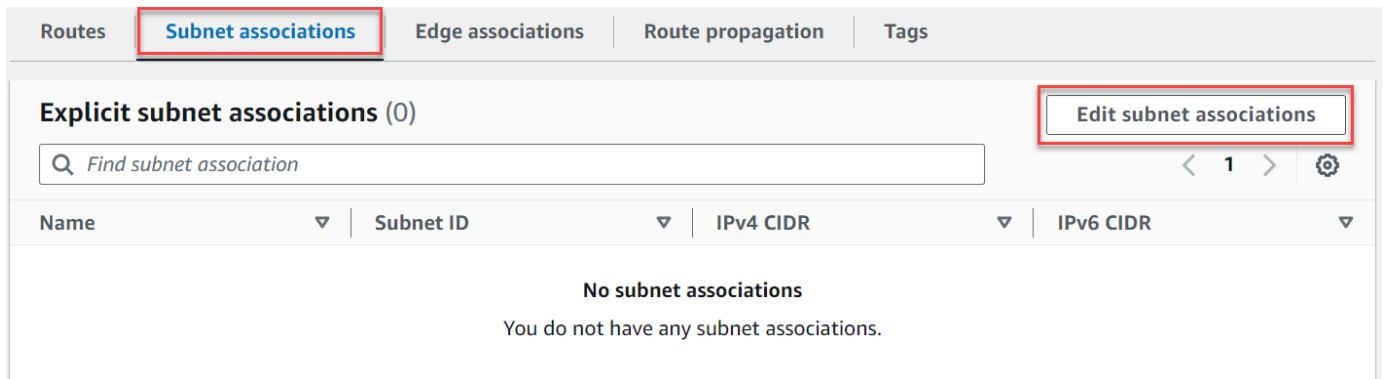
Completa la seguente procedura per creare una tabella di routing e associarla a una sottorete.

Per creare una tabella di routing personalizzata utilizzando la console Amazon VPC

- Apri la console Amazon VPC all'indirizzo <https://console.aws.amazon.com/vpc/>.
- Nel riquadro di navigazione, seleziona Tabelle di routing.
- Selezionare Create route table (Crea tabella di instradamento).
- (Facoltativo) In Name (Nome), inserisci un nome per la tabella di instradamento.
- In VPC, seleziona il VPC.
- (Facoltativo) Per aggiungere un tag, scegli Add new tag (Aggiungi nuovo tag) e inserisci la chiave e il valore del tag.



7. Selezionare Create route table (Crea tabella di instradamento).
8. Nella scheda Associazioni sottorete scegli Modifica associazioni sottorete.



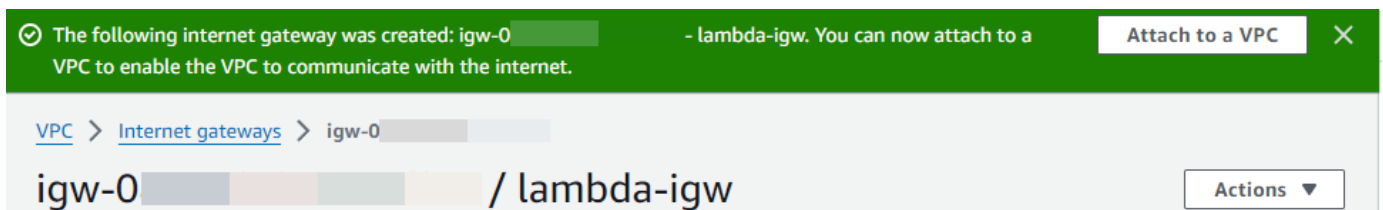
9. Seleziona la casella di controllo per la sottorete da associare alla tabella di instradamento.
10. Scegli Salva associazioni.

### Creazione di un Internet Gateway

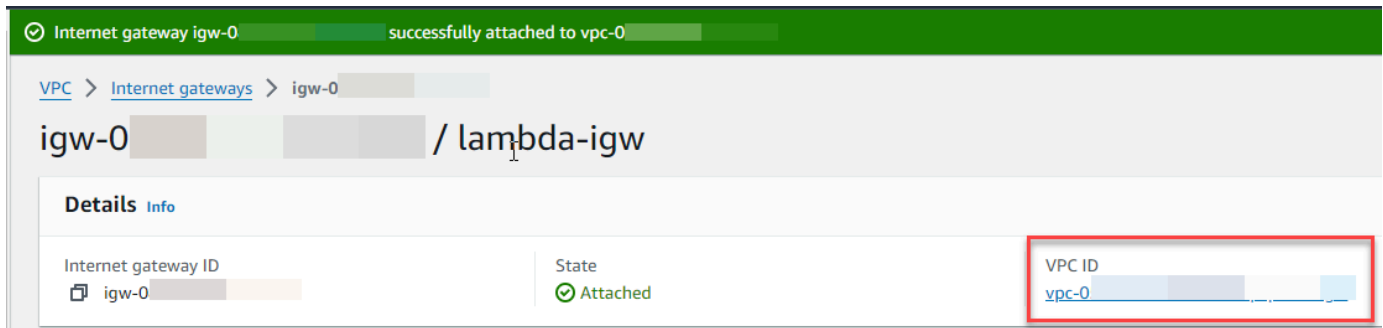
Segui questi passaggi per creare un gateway Internet, collegarlo al tuo VPC e aggiungerlo alla tabella di routing della sottorete pubblica.

### Come creare un gateway Internet

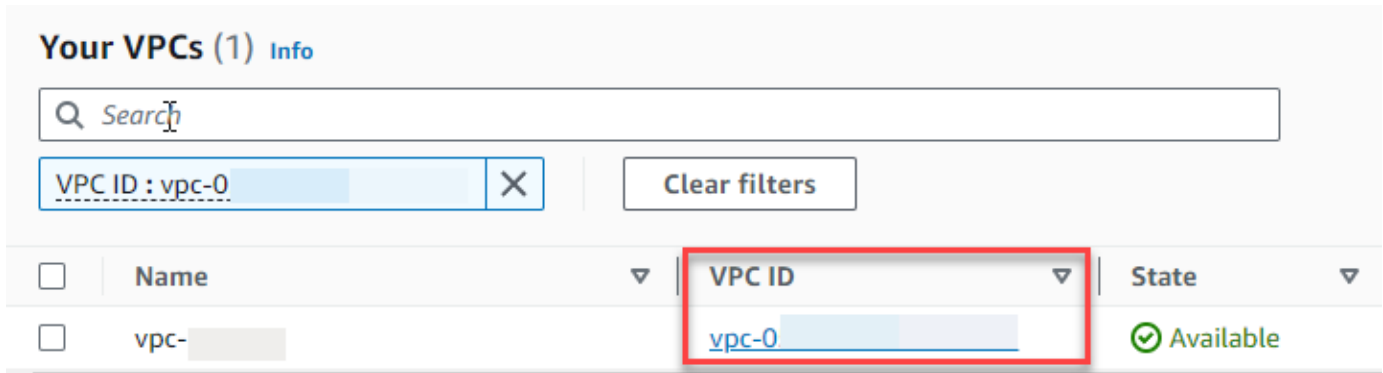
1. Apri la console Amazon VPC all'indirizzo <https://console.aws.amazon.com/vpc/>.
2. Nel riquadro di navigazione, seleziona Gateway Internet.
3. Scegliere Crea gateway Internet.
4. (Facoltativo) Inserisci un nome per il gateway Internet.
5. (Facoltativo) Per aggiungere un tag, scegli Add new tag (Aggiungi nuovo tag) e immetti la chiave e il valore del tag.
6. Scegliere Crea gateway Internet.
7. Scegli Collega a un VPC dal banner nella parte superiore dello schermo, seleziona un VPC disponibile e scegli Collega un gateway Internet.



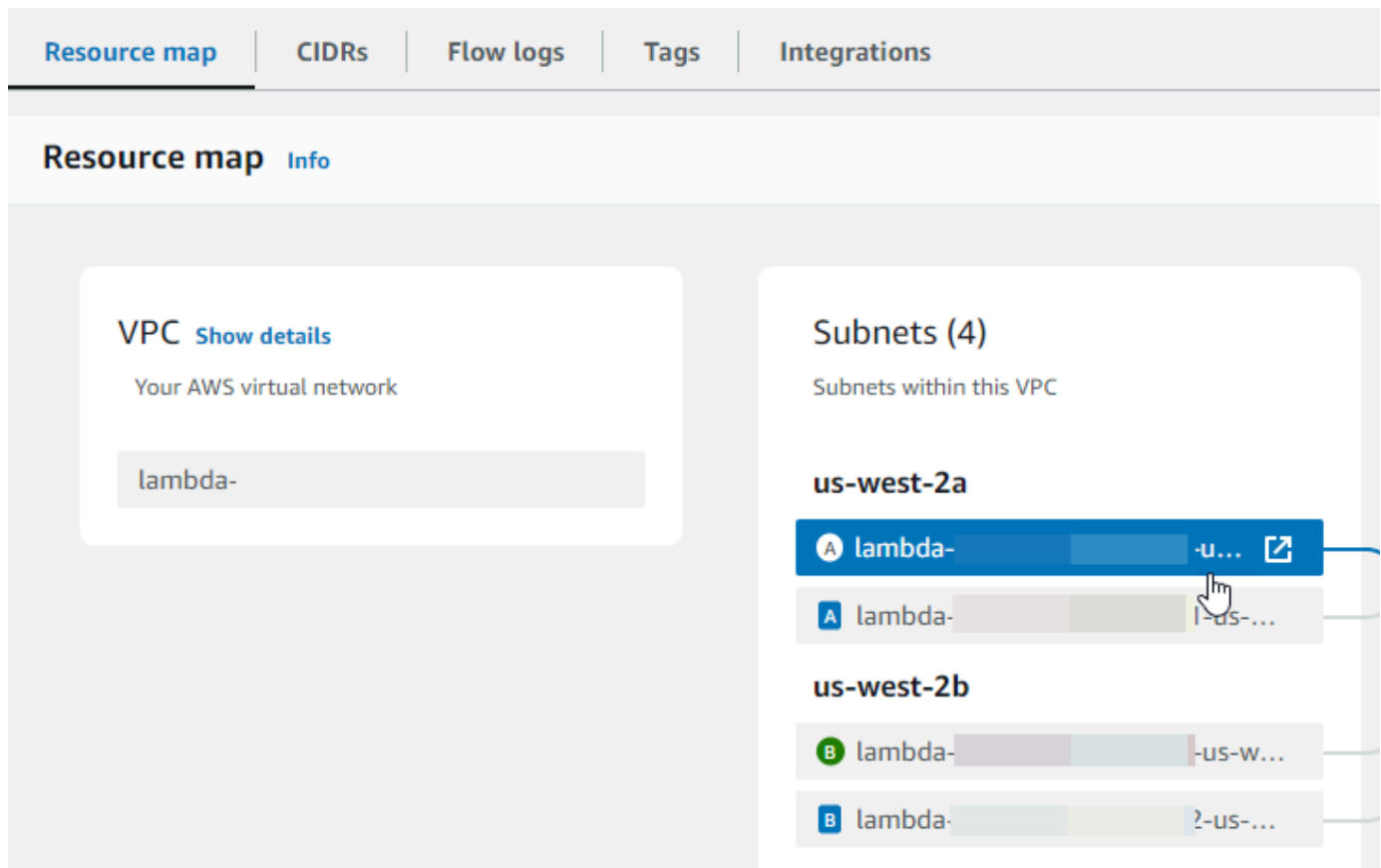
8. Scegli l'ID VPC.



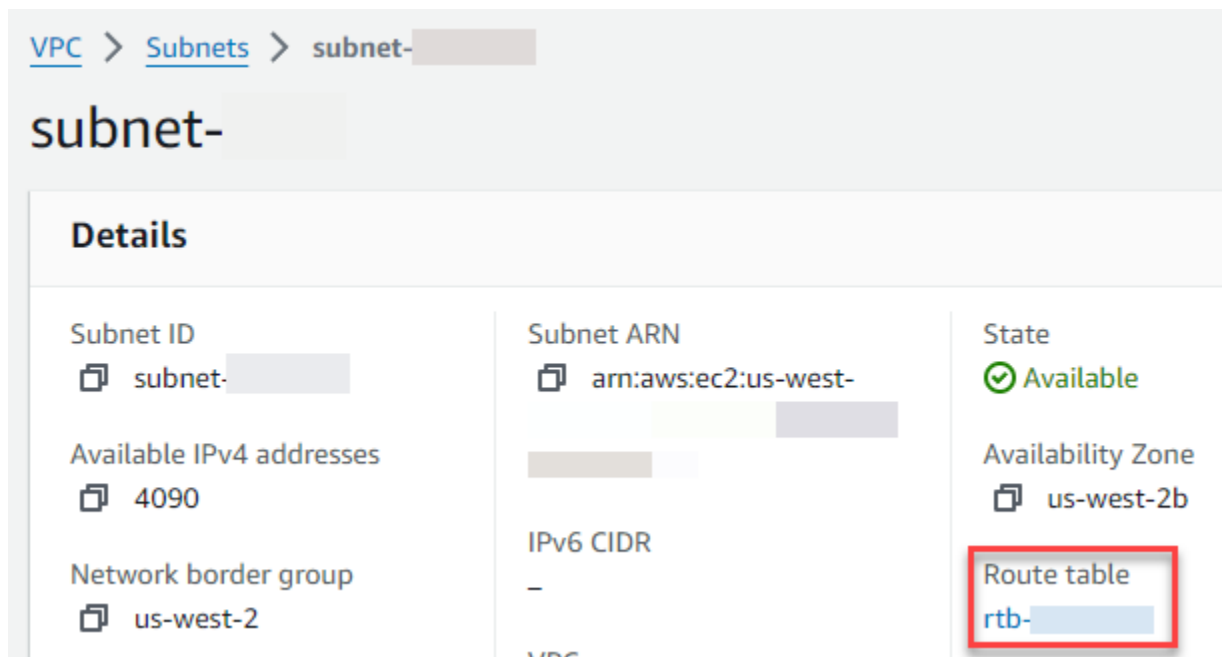
9. Scegli di nuovo l'ID VPC per aprire la pagina dei dettagli del VPC.



10. Scorri verso il basso fino alla sezione Mappa delle risorse, quindi scegli una sottorete. I dettagli della sottorete vengono visualizzati in una nuova scheda.



11. Scegli il link sotto Tabella di routing.



12. Seleziona ID tabella di routing per aprire la pagina dei dettagli della tabella di routing.

**Route tables (1)** Info

Find resources by attribute or tag

Route table ID : rtb-0 X Clear filters

| Name | Route table ID |
|------|----------------|
| -    | rtb-0          |

13. In Route, seleziona Modifica route.

Routes Subnet associations Edge associations Route propagation Tags

**Routes (1)** Both Edit routes

Filter routes

| Destination | Target | Status |
|-------------|--------|--------|
| 10.0.0.0/24 | local  | Active |

14. Scegli Aggiungi route, quindi inserisci 0.0.0.0/0 nella casella Destinazione.

**Edit routes**

| Destination | Target | Status |
|-------------|--------|--------|
| 10.0.0.0/24 | local  | Active |

Search for Destination: 0.0.0.0/0, 0.0.0.0/8, 0.0.0.0/16

Search for Target: local

15. Per Destinazione, seleziona Internet gateway, quindi scegli il gateway Internet creato in precedenza. Se la sottorete ha un blocco CIDR IPv6, devi anche aggiungere una route per `:::/0` per lo stesso gateway Internet.

## Edit routes

| Destination                              | Target                                                                                                                                                                                     |
|------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10.0.0.0/24                              | local                                                                                                                                                                                      |
| <input type="text" value="0.0.0.0/0"/>   | <input type="text" value="local"/>                                                                                                                                                         |
| <input type="button" value="Add route"/> | <div><div>Carrier Gateway</div><div>Core Network</div><div>Egress Only Internet Gateway</div><div>Gateway Load Balancer Endpoint</div><div>Instance</div><div>Internet Gateway</div></div> |

16. Scegli Save changes (Salva modifiche).

### Creazione di un gateway NAT

Segui questi passaggi per creare un gateway NAT, associarlo a una sottorete pubblica e aggiungerlo alla tabella di routing della sottorete privata.

Per creare un gateway NAT e associarlo a una sottorete pubblica

1. Nel riquadro di navigazione, scegli Gateway NAT.
2. Scegli Crea gateway NAT.
3. (Facoltativo) Inserisci un nome per il gateway NAT.
4. In Sottorete seleziona una sottorete pubblica del VPC. Una sottorete pubblica è una sottorete che ha una route diretta a un gateway Internet nella sua tabella di routing.

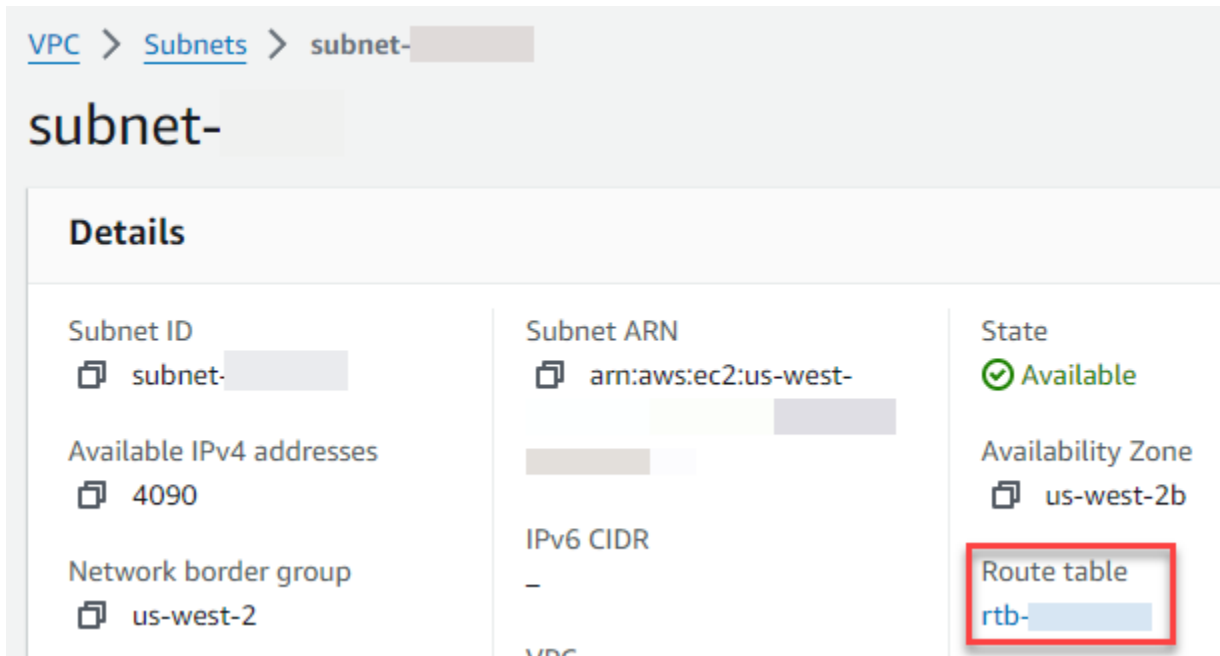
#### Note

I gateway NAT sono associati a una sottorete pubblica, ma la voce della tabella di routing si trova nella sottorete privata.

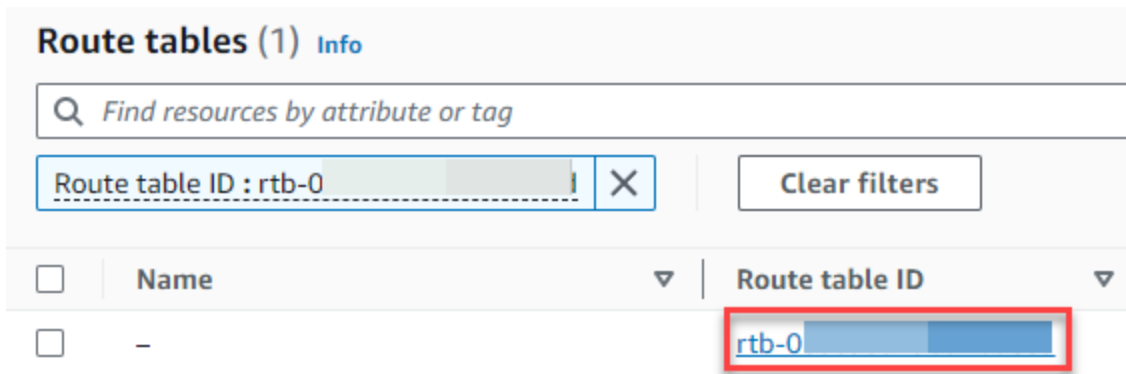
5. Per ID allocazione IP elastico, seleziona un indirizzo IP elastico o scegli Alloca IP elastico.
6. Scegli Crea gateway NAT.

Per aggiungere una route al gateway NAT nella tabella di routing della sottorete privata

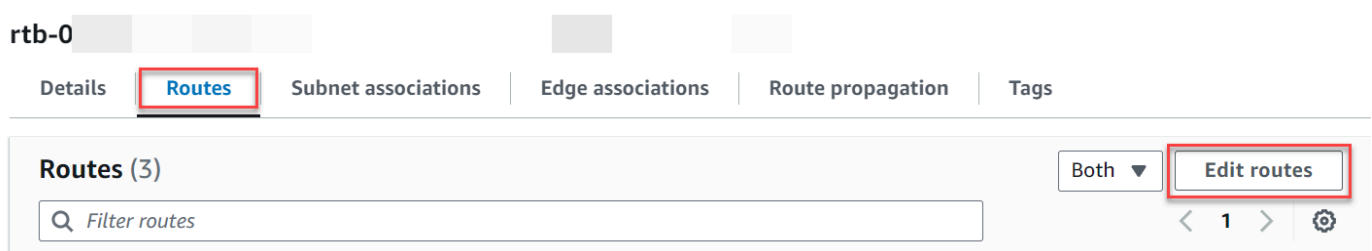
1. Nel riquadro di navigazione, seleziona Sottoreti.
2. Seleziona una sottorete privata nel VPC. Una sottorete privata è una sottorete che non dispone di una route a un gateway Internet nella sua tabella di routing.
3. Scegli il link sotto Tabella di routing.



4. Seleziona ID tabella di routing per aprire la pagina dei dettagli della tabella di routing.



5. Scorri verso il basso e seleziona la scheda Route, quindi scegli Modifica route.



6. Scegli Aggiungi route, quindi inserisci `0.0.0.0/0` nella casella Destinazione.

**Edit routes**

| Destination | Target | Status |
|-------------|--------|--------|
| 10.0.0.0/24 | local  | Active |

Search for destinations:

- 0.0.0.0/0
- 0.0.0.0/8
- 0.0.0.0/16

7. Per Destinazione, seleziona Gateway NAT, quindi scegli il gateway NAT creato in precedenza.

[VPC](#) > [Route tables](#) > [rtb-](#) > [Edit routes](#)

**Edit routes**

| Destination | Target |
|-------------|--------|
| 0.0.0.0/0   | local  |

Search for destinations:

Target dropdown options:

- Carrier Gateway
- Core Network
- Egress Only Internet Gateway
- Gateway Load Balancer Endpoint
- Instance
- Internet Gateway
- local
- NAT Gateway**
- Network Interface

8. Scegli Save changes (Salva modifiche).

Creare un gateway Internet di sola uscita (solo IPv6)

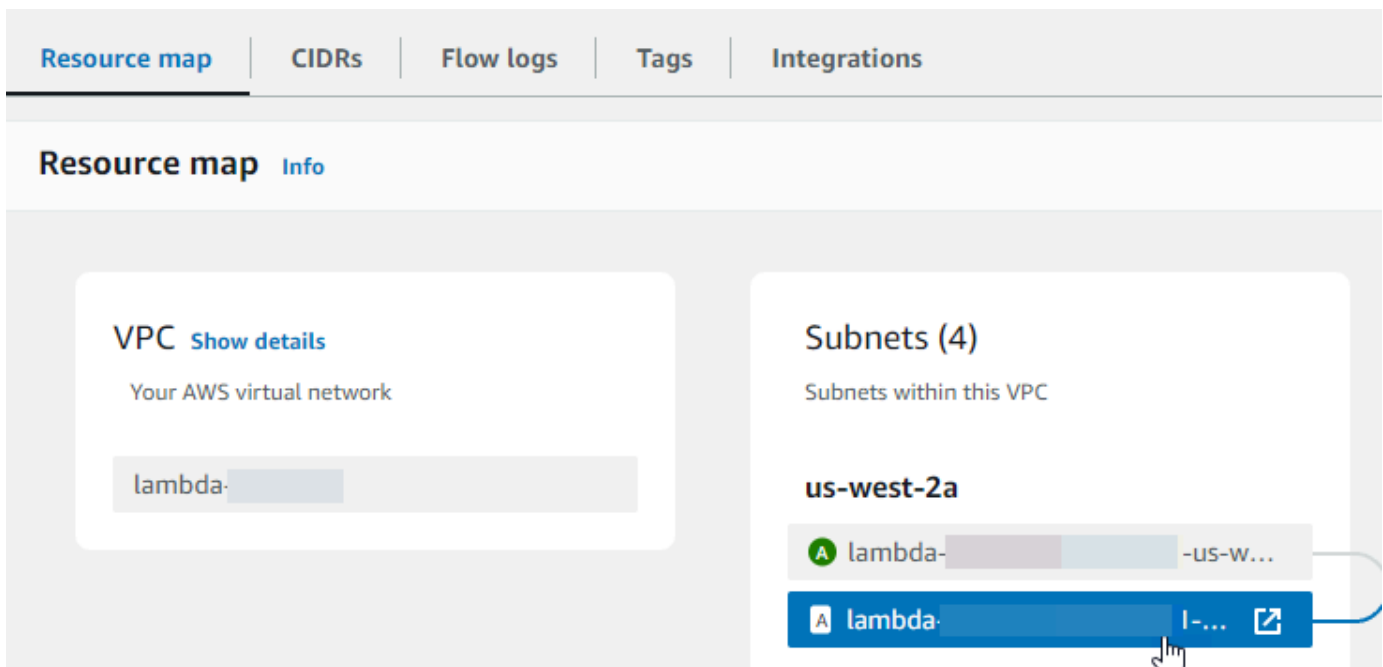
Segui questi passaggi per creare un gateway Internet solo in uscita e aggiungerlo alla tabella di routing della tua sottorete privata.

## Per creare un gateway internet egress-only

1. Nel pannello di navigazione, scegli i gateway Egress-only Internet.
2. Seleziona Crea gateway Internet solo in uscita.
3. (Facoltativo) Immetti un nome.
4. Selezionare il VPC nel quale creare l'Internet Gateway egress-only.
5. Seleziona Crea gateway Internet solo in uscita.
6. Scegli il link sotto ID VPC collegato.



7. Scegliere il link sotto ID VPC per aprire la pagina dei dettagli del VPC.
8. Scorri verso il basso fino alla sezione Mappa delle risorse, quindi scegli una sottorete privata. Una sottorete privata è una sottorete che non dispone di una route a un gateway Internet nella sua tabella di routing. I dettagli della sottorete vengono visualizzati in una nuova scheda.



9. Scegli il link sotto Tabella di routing.



**subnet-0** -subnet-private1-us-west-2a

**Details**

|                                   |                                    |                                 |
|-----------------------------------|------------------------------------|---------------------------------|
| Subnet ID<br>subnet-              | Subnet ARN<br>arn:aws:ec2:us-west- | State<br>Available              |
| Available IPv4 addresses<br>4090  | IPv6 CIDR<br>::/64                 | Availability Zone<br>us-west-2a |
| Network border group<br>us-west-2 | VPC<br>vpc-                        | Route table<br>rtb-0<br>west-2a |
| Default subnet<br>No              | Auto-assign public IPv4 address    | Auto-assign IPv6 address        |

10. Seleziona ID tabella di routing per aprire la pagina dei dettagli della tabella di routing.

**Route tables (1)** Info

Find resources by attribute or tag

Route table ID : rtb-0 X Clear filters

| Name | Route table ID |
|------|----------------|
| -    | rtb-0          |

11. In Route, seleziona Modifica route.

**Routes (1)** Both Edit routes

Filter routes

| Destination | Target | Status |
|-------------|--------|--------|
| 10.0.0.0/24 | local  | Active |

12. Scegli Aggiungi route, quindi inserisci ::/0 nella casella Destinazione.

**Edit routes**

| Destination | Target | Status |
|-------------|--------|--------|
| 10.0.0.0/24 | local  | Active |
| 0.0.0.0/0   |        |        |
| 0.0.0.0/8   |        |        |
| 0.0.0.0/16  |        |        |

13. Per Destinazione, seleziona Gateway Internet solo in uscita, quindi scegli il gateway creato in precedenza.

### Edit routes

| Destination   | Target                       | Status   |
|---------------|------------------------------|----------|
| ::/56         | local                        | ✓ Active |
|               | Q local X                    |          |
| 10.0.0.0/16   | local                        | ✓ Active |
|               | Q local X                    |          |
| Q 0.0.0.0/0 X | NAT Gateway                  | ✓ Active |
|               | Q nat- X                     |          |
| Q ::/0 X      | Egress Only Internet Gateway | ✓ Active |
|               | Q eigw- X                    |          |

14. Scegli Save changes (Salva modifiche).

## Configura la funzione Lambda

Per configurare un VPC quando si crea una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli Crea funzione.
3. In Informazioni di base, immettere un nome per la funzione in Nome funzione.
4. Espandere Advanced settings (Impostazioni avanzate).
5. Seleziona Abilita VPC, quindi scegli un VPC.
6. (Facoltativo) Per consentire il [traffico IPv6 in uscita](#), selezionare Consenti il traffico IPv6 per sottoreti dual-stack.
7. Per Sottoreti, seleziona tutte le sottoreti private. Le sottoreti private possono accedere a Internet attraverso un gateway NAT. La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet.

#### Note

Se hai selezionato Consenti il traffico IPv6 per sottoreti dual-stack, tutte le sottoreti selezionate devono avere un blocco CIDR IPv4 e un blocco CIDR IPv6.

8. Per Gruppi di sicurezza, seleziona un gruppo di sicurezza che consenta il traffico in uscita.
9. Scegli Crea funzione.

Lambda crea automaticamente un ruolo di esecuzione con la policy

[AWSLambdaVPCLambdaAccessExecutionRole](#) AWS gestita. Le autorizzazioni in questa policy sono necessarie solo per creare interfacce di rete elastiche per la configurazione VPC, non per richiamare la funzione. Per applicare le autorizzazioni con privilegi minimi, puoi rimuovere la [AWSLambdaVPCLambdaAccessExecutionRole](#) policy dal tuo ruolo di esecuzione dopo aver creato la funzione e la configurazione del VPC. Per ulteriori informazioni, consulta [Autorizzazioni IAM richieste](#).

Per configurare un VPC per una funzione esistente

Per aggiungere una configurazione VPC a una funzione esistente, il ruolo di esecuzione della funzione deve disporre dell'[autorizzazione per creare e gestire interfacce di rete elastiche](#). La policy [AWSLambdaVPCLambdaAccessExecutionRole](#) AWS gestita include le autorizzazioni richieste. Per applicare le autorizzazioni con privilegi minimi, puoi rimuovere la [AWSLambdaVPCLambdaAccessExecutionRole](#) policy dal tuo ruolo di esecuzione dopo aver creato la configurazione VPC.

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegli la scheda Configurazione, quindi scegli VPC.
4. In VPC, scegli Modifica.
5. Seleziona il VPC
6. (Facoltativo) Per consentire il [traffico IPv6 in uscita](#), selezionare Consenti il traffico IPv6 per sottoreti dual-stack.
7. Per Sottoreti, seleziona tutte le sottoreti private. Le sottoreti private possono accedere a Internet attraverso un gateway NAT. La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet.

 Note

Se hai selezionato Consenti il traffico IPv6 per sottoreti dual-stack, tutte le sottoreti selezionate devono avere un blocco CIDR IPv4 e un blocco CIDR IPv6.

8. Per Gruppi di sicurezza, seleziona un gruppo di sicurezza che consenta il traffico in uscita.
9. Scegli Save (Salva).

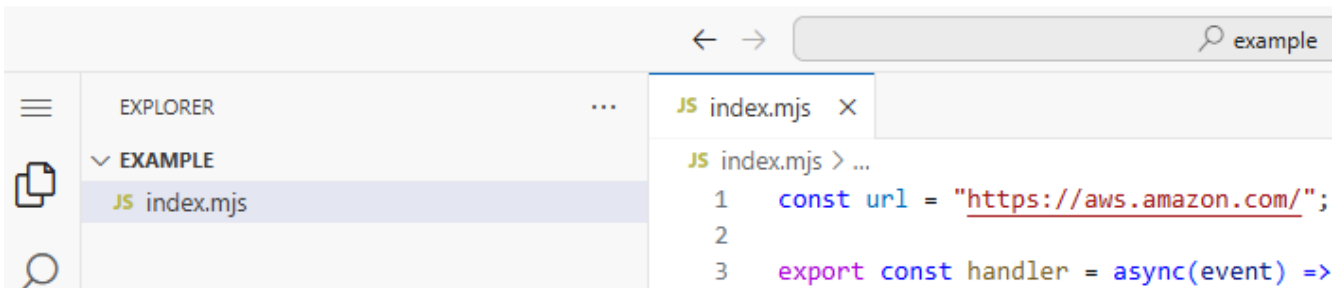
## Test della funzione

Utilizza il seguente codice di esempio per confermare che la tua VPC-connected funzione può raggiungere la rete Internet pubblica. In caso di successo, il codice restituisce un codice di stato 200. Se non ha esito positivo, la funzione scade.

### Node.js

1. Nel riquadro Codice sorgente della console Lambda, incolla il codice seguente nel file `index.mjs`. La funzione effettua una richiesta HTTP GET a un endpoint pubblico e restituisce il codice di risposta HTTP per verificare se la funzione ha accesso alla rete Internet pubblica.

#### Code source [Info](#)

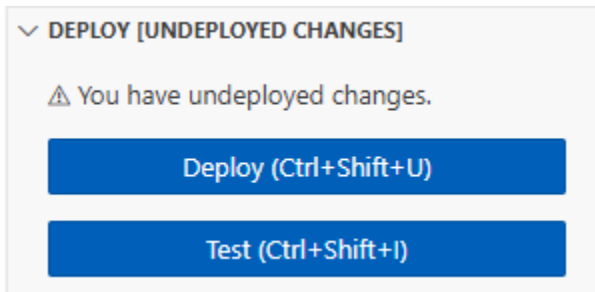


#### Example— Richiesta HTTP con `async/await`

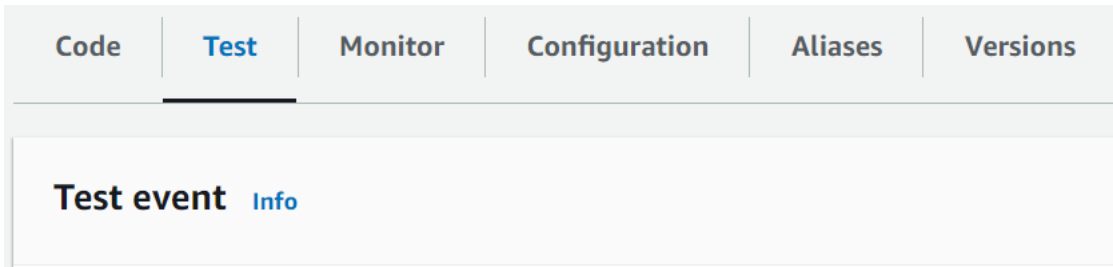
```
const url = "https://aws.amazon.com/";

export const handler = async(event) => {
  try {
    const res = await fetch(url);
    console.info("status", res.status);
    return res.status;
  }
  catch (e) {
    console.error(e);
    return 500;
  }
};
```

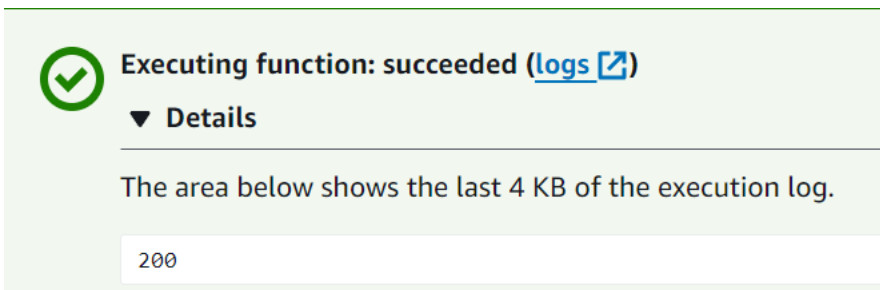
2. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



3. Seleziona la scheda Test.



4. Scegli Test (Esegui test).
5. La funzione restituisce un codice di stato 200. Ciò significa che la funzione dispone di un accesso a Internet in uscita.



Se la funzione non riesce a raggiungere la rete Internet pubblica, ti verrà mostrato un messaggio di errore come questo:

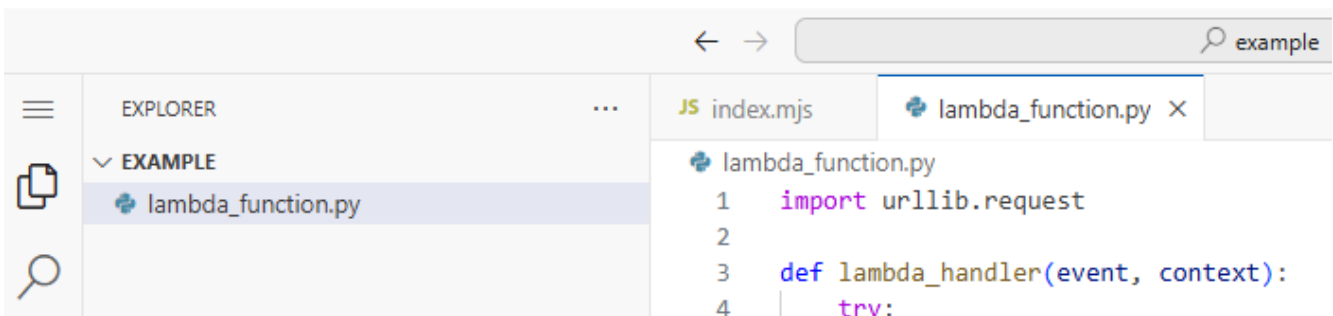
```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
Task timed out after 3.01 seconds"
}
```

## Python

1. Nel riquadro Codice sorgente della console Lambda, incolla il seguente codice seguente nel file `lambda_function.py`. La funzione effettua una richiesta HTTP GET a un endpoint pubblico

e restituisce il codice di risposta HTTP per verificare se la funzione ha accesso alla rete Internet pubblica.

## Code source [Info](#)



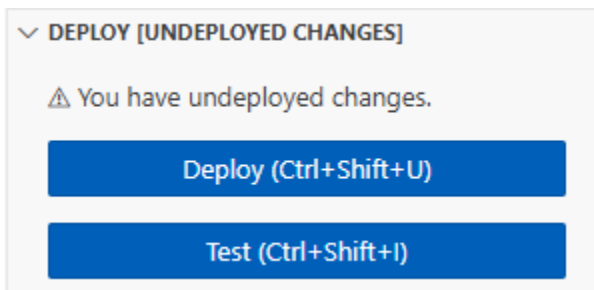
```

import urllib.request

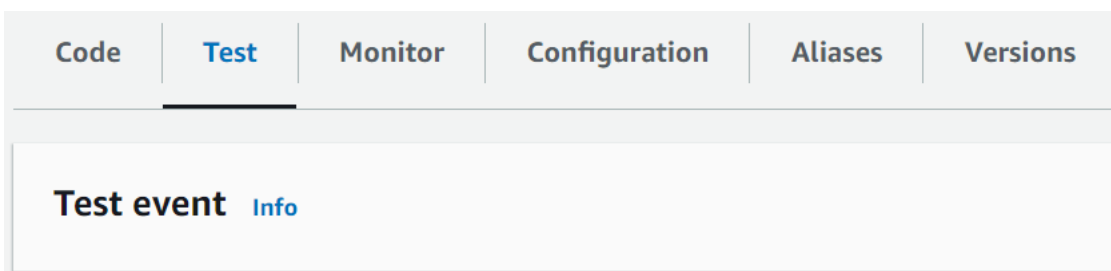
def lambda_handler(event, context):
    try:
        response = urllib.request.urlopen('https://aws.amazon.com')
        status_code = response.getcode()
        print('Response Code:', status_code)
        return status_code
    except Exception as e:
        print('Error:', e)
        raise e

```

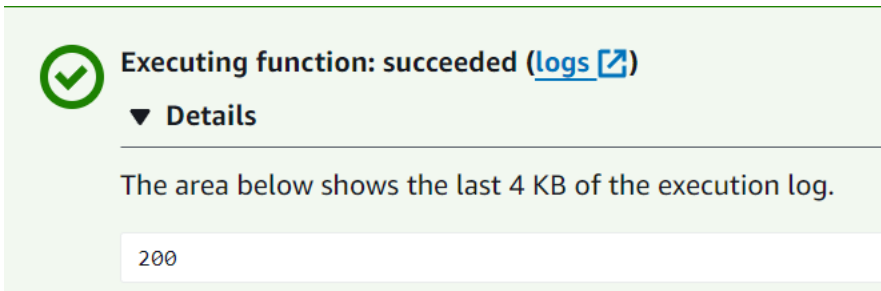
2. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



3. Seleziona la scheda Test.



4. Scegli Test (Esegui test).
5. La funzione restituisce un codice di stato 200. Ciò significa che la funzione dispone di un accesso a Internet in uscita.



Se la funzione non riesce a raggiungere la rete Internet pubblica, ti verrà mostrato un messaggio di errore come questo:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
Task timed out after 3.01 seconds"
}
```

## Connessione di endpoint VPC con interfaccia in entrata per Lambda

Se utilizzi Amazon Virtual Private Cloud (Amazon VPC) per ospitare AWS le tue risorse, puoi stabilire una connessione tra il tuo VPC e Lambda. È possibile utilizzare questa connessione per richiamare la propria funzione Lambda senza attraversare l'Internet pubblico.

È possibile stabilire una connessione privata tra il VPC e Lambda creando un [endpoint VPC di interfaccia](#). Endpoint di interfaccia con tecnologia [AWS PrivateLink](#). Con AWS PrivateLink, puoi accedere privatamente alle API Lambda senza un gateway Internet, un dispositivo NAT, una connessione VPN o una connessione. AWS Direct Connect Le istanze presenti nel VPC non richiedono indirizzi IP pubblici per comunicare con le API Lambda. Il traffico tra il tuo VPC e Lambda non esce dalla rete AWS .

Ogni endpoint di interfaccia è rappresentato da una o più [interfacce di rete elastiche](#) nelle sottoreti. Un'interfaccia di rete fornisce un indirizzo IP privato che funge da punto di ingresso per il traffico verso Lambda.

### Sections

- [Considerazioni per gli endpoint di interfaccia Lambda](#)
- [Creazione di un endpoint di interfaccia per Lambda](#)
- [Creazione di una policy degli endpoint di interfaccia per Lambda](#)

## Considerazioni per gli endpoint di interfaccia Lambda

Prima di impostare un endpoint di interfaccia per Lambda, esaminare l'argomento [Proprietà e limitazioni degli endpoint dell'interfaccia](#) nella Guida per l'utente di Amazon VPC.

Puoi chiamare qualsiasi operazione API Lambda dal tuo VPC. Ad esempio, è possibile richiamare la funzione Lambda invocando l'API Invoke dall'interno del VPC. Per l'elenco completo delle API Lambda, vedere [Azioni](#) nella Guida di riferimento alle API Lambda.

use1-az3 è una regione a capacità limitata per le funzioni Lambda VPC. L'utilizzo di sottoreti con le funzioni Lambda in questa zona di disponibilità non è consigliato perché, in caso di interruzione del servizio, potrebbe provocare una riduzione della ridondanza zonale.

### Keep-alive per connessioni permanenti

Lambda elimina le connessioni inattive nel tempo, quindi occorre utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Se si tenta di riutilizzare una connessione inattiva quando



si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, consulta [Reusing Connections with Keep-Alive in Node.js](#) nella AWS SDK per JavaScript Developer Guide.

### Considerazioni sulla fatturazione

Non vi è alcun costo aggiuntivo per accedere a una funzione Lambda tramite un endpoint di interfaccia. Per ulteriori informazioni sui prezzi di Lambda, consulta [Prezzi di AWS Lambda](#).

Il prezzo standard per AWS PrivateLink si applica agli endpoint di interfaccia per Lambda. All' AWS account viene fatturato ogni ora di provisioning di un endpoint di interfaccia in ciascuna zona di disponibilità e per i dati elaborati tramite l'endpoint di interfaccia. Per ulteriori informazioni sui prezzi degli endpoint di interfaccia, consulta [Prezzi di AWS PrivateLink](#).

### Considerazioni sul peering VPC

È possibile connettere altri VPC al VPC con endpoint di interfaccia utilizzando [Peering VPC](#). Il peering VPC è una connessione di rete tra due VPC. Puoi creare una connessione peering VPC tra i tuoi VPC oppure con un VPC in un altro account AWS . I VPC possono anche trovarsi in due regioni diverse. AWS

Il traffico tra VPC peered rimane sulla AWS rete e non attraversa la rete Internet pubblica. Una volta eseguito il peering, risorse come le istanze Amazon Elastic Compute Cloud (Amazon EC2), le istanze di Amazon Relational Database Service (Amazon RDS) o le funzioni Lambda in entrambi i VPC possono accedere all'API VPC-enabled Lambda tramite endpoint di interfaccia creati in uno dei VPC.

### Creazione di un endpoint di interfaccia per Lambda

Puoi creare un endpoint di interfaccia per Lambda utilizzando la console Amazon VPC o (). AWS Command Line Interface AWS CLI Per ulteriori informazioni, consultare [Creazione di un endpoint di interfaccia](#) nella Guida per l'utente di Amazon VPC.

Per creare un endpoint di interfaccia per Lambda (console)

1. Aprire la [pagina Endpoint](#) della console Amazon VPC.
2. Scegliere Create Endpoint (Crea endpoint).
3. In Categoria del servizio, assicurati che sia selezionato Servizi AWS.
4. Per Service Name, scegli com.amazonaws. **region**.lambda. Verificare che il tipo sia Interfaccia.
5. Creazione di un VPC e delle sottoreti

6. Per abilitare il DNS privato per l'endpoint di interfaccia, in Enable DNS Name (Abilita nome DNS), selezionare la casella di controllo. Ti consigliamo di abilitare nomi host DNS privati per gli endpoint VPC per Servizi AWS. Ciò garantisce che le richieste che utilizzano gli endpoint del servizio pubblico, come le richieste effettuate tramite un AWS SDK, vengano risolte nell'endpoint VPC.
7. Per gruppo di sicurezza, scegliere uno o più gruppi di sicurezza.
8. Seleziona Crea endpoint.

Per utilizzare l'opzione DNS privato, è necessario impostare `enableDnsHostnames` e `enableDnsSupportattributes` del VPC. Per ulteriori informazioni, consulta [Visualizzazione e aggiornamento del supporto DNS per il VPC](#) nella Guida per l'utente di Amazon VPC. Se si abilita il DNS privato per l'endpoint di interfaccia, è possibile effettuare richieste API verso Lambda utilizzando il nome DNS predefinito per la regione, ad esempio `lambda.us-east-1.amazonaws.com`. Per ulteriori endpoint di servizio, vedere [Endpoint e quote del servizio](#) in Riferimenti generali di AWS.

Per ulteriori informazioni, consultare [Accesso a un servizio tramite un endpoint di interfaccia](#) in Guida per l'utente di Amazon VPC.

Per informazioni sulla creazione e la configurazione di un endpoint utilizzando CloudFormation, consulta la risorsa [AWS: :EC2: :VPC Endpoint nella Guida per l'utente](#). AWS CloudFormation

Per creare un endpoint di interfaccia per Lambda (AWS CLI)

Usa il [comando](#) `create-vpc-endpoint` e specifica l'ID VPC, il tipo di endpoint VPC (interfaccia), il nome del servizio, le sottoreti che utilizzano l'endpoint e i gruppi di sicurezza da associare alle interfacce di rete dell'endpoint. Ad esempio:

```
aws ec2 create-vpc-endpoint
  --vpc-id vpc-ec43eb89
  --vpc-endpoint-type Interface
  --service-name com.amazonaws.us-east-1.lambda
  --subnet-id subnet-abababab
  --security-group-id sg-1a2b3c4d
```

## Creazione di una policy degli endpoint di interfaccia per Lambda

Per controllare chi può utilizzare l'endpoint di interfaccia e a quali funzioni Lambda l'utente può accedere, è possibile allegare una policy endpoint all'endpoint. La policy specifica le informazioni riportate di seguito:

- Il principale che può eseguire operazioni.
- Le azioni che l'entità può eseguire.
- Le risorse su cui l'utente/gruppo/ruolo può eseguire azioni.

Per ulteriori informazioni, consulta [Controllo degli accessi ai servizi con endpoint VPC](#) nella Guida per l'utente di Amazon VPC.

Esempio: policy endpoint di interfaccia per le azioni Lambda

Di seguito è riportato un esempio di una policy endpoint per Lambda. Quando è collegata a un endpoint, questa policy consente all'utente MyUser di richiamare la funzione my-function.

 Note

Nella risorsa è necessario includere l'ARN della funzione qualificata e non qualificata.

```
{
  "Statement": [
    {
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/MyUser"
      },
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": [
        "arn:aws:lambda:us-east-2:123456789012:function:my-function",
        "arn:aws:lambda:us-east-2:123456789012:function:my-function:*"
      ]
    }
  ]
}
```

Per consentire alla tua funzione di accedere alle risorse in un VPC che appartiene a un altro, consulta [Account AWS the section called “Collegare funzioni alle risorse di un altro account”](#)

## Configurazione dell'accesso al file system per le funzioni Lambda

È possibile configurare una funzione Lambda per montare un file system in una directory locale. Lambda supporta i seguenti tipi di file system:

- [Amazon Elastic File System \(Amazon EFS\)](#): file system serverless che si adatta automaticamente ai carichi di lavoro.
- [Amazon S3 Files: file](#) system senza server per il montaggio del bucket Amazon S3. Amazon S3 Files fornisce l'accesso ai tuoi oggetti Amazon S3 come file utilizzando operazioni di file system standard come lettura e scrittura sul percorso di montaggio locale.

### Note

Una funzione Lambda può utilizzare Amazon EFS o Amazon S3 Files, ma non entrambi. Se la tua funzione è già configurata con un tipo di file system, devi rimuoverla prima di configurare l'altro.

## Configurazione dell'accesso al file system Amazon EFS

È possibile configurare una funzione per montare un file system Amazon Elastic File System (Amazon EFS) in una directory locale. Amazon EFS è un file system serverless che si adatta automaticamente ai carichi di lavoro. Con Amazon EFS, il codice della funzione può accedere e modificare le risorse condivise in modo sicuro e ad alta concorrenza.

### Sections

- [Autorizzazioni del ruolo di esecuzione e dell'utente](#)
- [Configurazione di un file system e di un punto di accesso](#)
- [Connessione a un file system \(console\)](#)
- [Regioni supportate](#)

### Autorizzazioni del ruolo di esecuzione e dell'utente

Se il file system non dispone di una policy configurata dall'utente AWS Identity and Access Management (IAM), EFS utilizza una policy predefinita che concede l'accesso completo a qualsiasi client in grado di connettersi al file system utilizzando una destinazione di montaggio del file system.

Se il file system dispone di una Policy IAM configurata dall'utente, il ruolo di esecuzione della funzione deve disporre delle autorizzazioni `elasticfilesystem` corrette.

#### Autorizzazioni del ruolo di esecuzione

- file system elastico: `ClientMount`
- `elasticfilesystem`: (non richiesto per connessioni di sola lettura) `ClientWrite`

Queste autorizzazioni sono incluse nella politica gestita.

`AmazonElasticFileSystemClientReadWriteAccess` Inoltre, il ruolo di esecuzione deve avere le [autorizzazioni necessarie per connettersi al VPC del file system](#).

Quando si configura un file system, Lambda utilizza le autorizzazioni dell'utente per verificare le destinazioni di montaggio. Per configurare una funzione affinché possa connettersi a un file system, l'utente deve disporre delle seguenti autorizzazioni:

#### Autorizzazioni degli utenti

- file system elastico: `DescribeMountTargets`

#### Configurazione di un file system e di un punto di accesso

Creare un file system in Amazon EFS con una destinazione di montaggio in ogni zona di disponibilità a cui la funzione si connette. Per prestazioni e resilienza, utilizzare almeno due zone di disponibilità. Ad esempio, in una configurazione semplice è possibile avere un VPC con due sottoreti private in zone di disponibilità separate. La funzione si connette a entrambe le sottoreti e in ciascuna di esse è disponibile una destinazione di montaggio. Assicurarsi che il traffico NFS (porta 2049) sia consentito dai gruppi di sicurezza utilizzati dalla funzione e dalle destinazioni di montaggio.

#### Note

Quando si crea un file system, si sceglie una modalità di prestazioni che non può essere modificata in un secondo momento. La modalità Uso generale ha una latenza inferiore e la modalità I/O max supporta throughput e IOPS massimi più elevati. Per assistenza nella scelta, consulta [Prestazioni di Amazon EFS](#) nella Amazon Elastic File System User Guide.

Un punto di accesso collega ogni istanza della funzione alla destinazione di montaggio corretta per la zona di disponibilità a cui si connette. Per ottenere prestazioni ottimali, creare un punto di

accesso con un percorso non root e limitare il numero di file creati in ogni directory. Nell'esempio seguente viene creata una directory denominata `my-function` nel file system e viene impostato l'ID proprietario su 1001 con autorizzazioni di directory standard (755).

#### Example Configurazione del punto di accesso

- Nome – `files`
- ID utente: 1001
- ID gruppo: 1001
- Percorso – `/my-function`
- Autorizzazioni: 755
- ID utente proprietario – 1001
- ID utente gruppo – 1001

Quando una funzione utilizza il punto di accesso, riceve l'ID utente 1001 e ha accesso completo alla directory.

Per ulteriori informazioni, consulta i seguenti argomenti nella Amazon Elastic File System User Guide.

- [Creazione di risorse per Amazon EFS](#)
- [Utilizzo di utenti, gruppi e autorizzazioni](#)

#### Connessione a un file system (console)

Una funzione si connette a un file system tramite la rete locale in un VPC. Le sottoreti a cui si connette la funzione possono essere le stesse sottoreti che contengono punti di montaggio per il file system o sottoreti nella stessa zona di disponibilità che possono instradare il traffico NFS (porta 2049) al file system.

#### Note

Se la funzione non è già connessa a un VPC, consulta [Consentire alle funzioni Lambda l'accesso alle risorse in un Amazon VPC](#).

Per configurare l'accesso al file system EFS

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi scegliere File systems (File system).
4. In File system, scegliere Aggiungi file system.
5. Seleziona EFS.
6. Configurare le proprietà seguenti:
  - File system EFS: il punto di accesso per un file system nello stesso VPC.
  - Percorso di montaggio locale: la posizione in cui il file system è montato sulla funzione Lambda, a partire da /mnt/.

#### Prezzi

Amazon EFS prevede addebiti per lo storage e il throughput, con tariffe che variano in base alla classe di storage. Per informazioni dettagliate, consulta [Prezzi di Amazon EFS](#).

Lambda prevede addebiti per il trasferimento dei dati tra VPC. Questo vale solo se il VPC della funzione è collegato in peering a un altro VPC con un file system. Le tariffe sono le stesse del trasferimento dati di Amazon EC2 VPCs nella stessa regione. Per informazioni dettagliate, consulta [Prezzi di Lambda](#).

#### Regioni supportate

Amazon EFS per Lambda è disponibile in tutte le [regioni commerciali](#) ad eccezione di Asia Pacifico (Nuova Zelanda), Asia Pacifico (Taipei), Asia Pacifico (Malesia), Asia Pacifico (Tailandia) e Canada occidentale (Calgary).

#### Configurazione dell'accesso ai file Amazon S3

Amazon S3 Files offre un file system condiviso che collega qualsiasi risorsa di AWS calcolo direttamente ai tuoi dati in Amazon S3. Amazon S3 Files fornisce l'accesso ai tuoi oggetti Amazon S3 come file utilizzando operazioni di file system standard come lettura e scrittura sul percorso di montaggio locale. Scopri di più su [Amazon S3 Files](#).

#### Sections

- [Prerequisiti e configurazione](#)
- [Autorizzazioni del ruolo di esecuzione e dell'utente](#)
- [Connessione a un file system \(console\)](#)
- [Configurazione delle letture dirette](#)

## Prerequisiti e configurazione

Prima di configurare Amazon S3 Files con la funzione Lambda, assicurati di avere quanto segue:

- Un file system Amazon S3 e obiettivi di montaggio in stato di disponibilità nello stesso account e nella funzione Regione AWS Lambda.
- Una funzione Lambda nello stesso VPC del target di montaggio. È necessario disporre di un target di montaggio in ogni sottorete in cui è distribuita la funzione.
- Gruppi di sicurezza che consentono il traffico NFS (porta 2049) tra la funzione Lambda e i target di montaggio. [Scopri di più sulla configurazione dei gruppi di sicurezza.](#)

Per ulteriori informazioni, consulta i seguenti argomenti nella Guida per l'utente di Amazon S3:

- [Guida introduttiva ad Amazon S3 Files](#)
- [Prerequisiti per Amazon S3 Files](#)
- [Le migliori pratiche di Amazon S3 Files](#)

## Autorizzazioni del ruolo di esecuzione e dell'utente

Il ruolo di esecuzione della tua funzione deve disporre delle seguenti autorizzazioni per accedere a un file system Amazon S3 Files:

### Autorizzazioni del ruolo di esecuzione

- s3files: ClientMount — Necessario per montare il file system.
- s3files: ClientWrite — Richiesto per l'accesso in lettura/scrittura. Non necessario per le connessioni di sola lettura.

Queste autorizzazioni sono incluse nella policy gestita [AmazonS3FilesClientReadWriteAccess](#). Inoltre, il ruolo di esecuzione deve avere le [autorizzazioni necessarie per connettersi al VPC del file system](#).



 Note

Amazon S3 Files ottimizza il throughput trasmettendo in streaming le richieste di lettura idonee direttamente dal tuo bucket Amazon S3. Le letture dirette sono abilitate di default per le funzioni configurate con 512 MB o più di memoria. Per ulteriori informazioni, consulta [Configurazione delle letture dirette](#).

La tua funzione richiede anche le seguenti autorizzazioni per leggere direttamente da Amazon S3:

- s3: GetObject
- s3: GetObjectVersion

Per ulteriori informazioni sulle autorizzazioni richieste, consulta le autorizzazioni [IAM per i file Amazon S3](#) nella Amazon S3 User Guide.

Quando configuri un file system nella console, Lambda utilizza le tue autorizzazioni per verificare le destinazioni di montaggio e i punti di accesso. Per configurare una funzione affinché possa connettersi a un file system, l'utente deve disporre delle seguenti autorizzazioni:

#### Autorizzazioni degli utenti

- file s3: ListFileSystems
- file s3: ListAccessPoints
- file s3: GetFileSystem
- file s3: GetAccessPoint
- s3files: CreateAccessPoint — Necessario se si collega il file system alla funzione dalla console.

La seguente policy di esempio concede le autorizzazioni relative al ruolo di esecuzione della funzione per montare un file system Amazon S3 con accesso in lettura/scrittura e lettura direttamente da Amazon S3.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "S3FilesLambdaAccess",
```

```

        "Effect": "Allow",
        "Action": [
            "s3files:ClientMount",
            "s3files:ClientWrite"
        ],
        "Resource": "*"
    },
    {
        "Sid": "S3DirectRead",
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:GetObjectVersion"
        ],
        "Resource": "arn:aws:s3:::bucket-name/*"
    },
    {
        "Sid": "S3FilesConsoleSetup",
        "Effect": "Allow",
        "Action": [
            "s3files:ListFileSystems",
            "s3files:ListAccessPoints",
            "s3files:GetFileSystem",
            "s3files:GetAccessPoint",
            "s3files:CreateAccessPoint"
        ],
        "Resource": "*"
    }
]
}

```

## Connessione a un file system (console)

Una funzione si connette a un file system tramite la rete locale in un VPC. Le sottoreti a cui si connette la funzione possono essere le stesse sottoreti che contengono punti di montaggio per il file system o sottoreti nella stessa zona di disponibilità che possono instradare il traffico NFS (porta 2049) al file system.

### Note

Se la funzione non è già connessa a un VPC, consulta [Consentire alle funzioni Lambda l'accesso alle risorse in un Amazon VPC](#).

## Per configurare l'accesso ai file S3

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegli Configurazione, quindi scegli File system.
4. Scegli Aggiungi file system (o Modifica per modificare una configurazione esistente).
5. Seleziona File S3.
6. Configurare le proprietà seguenti:
  - File system S3: scegli un file system dal menu a discesa.
  - Punto di accesso (opzionale): scegli un punto di accesso. Se il file system non dispone di punti di accesso, Lambda ne crea automaticamente uno al momento del salvataggio (UID/GID 1000:1000, directory principale, permessi 755/lambda). Se esistono punti di accesso, è necessario selezionarne uno.
  - Percorso di montaggio locale: la posizione in cui il file system è montato sulla funzione Lambda, a partire da /mnt/.
  - Configurazione avanzata
    - Letture S3 dirette: controlla se Lambda è in grado di eseguire lo streaming di letture idonee direttamente dal bucket Amazon S3 per un throughput più elevato. Per ulteriori informazioni, consulta [Configurazione delle letture dirette](#).
7. Selezionare Salva.

Il file system viene collegato la prossima volta che richiami la funzione Lambda.

### Configurazione delle letture dirette

Le letture dirette ottimizzano il throughput trasmettendo i dati direttamente dal bucket Amazon S3. Lambda utilizza letture dirette per letture di grandi dimensioni e quando lo storage ad alte prestazioni del file system non contiene i dati del file. Per ulteriori informazioni, vedi [Come S3 Files offre prestazioni](#). Questa configurazione opzionale fa parte della `FileSystemConfigs` tua funzione e si applica solo ai file system Amazon S3. L'`DirectS3Read` impostazione accetta i seguenti valori:

- `AUTO`(impostazione predefinita): consente la lettura diretta per le funzioni configurate con 512 MB o più di memoria.
- `ENABLED`— Consente la lettura diretta per la funzione, incluse le funzioni con meno di 512 MB di memoria.

- **DISABLED**— Instrada tutte le letture attraverso l'archiviazione ad alte prestazioni del file system.

 **Note**

Se una lettura diretta ha esito negativo, Lambda torna automaticamente alla lettura del file system.

Per utilizzare le letture dirette, il ruolo di esecuzione della funzione deve disporre delle autorizzazioni richieste. Per ulteriori informazioni, consulta le politiche sui prerequisiti di [Amazon S3 Files](#).

## Creare un alias per una funzione Lambda

È possibile creare alias per la funzione Lambda. Un alias Lambda è un puntatore a una versione della funzione che è possibile aggiornare. Gli utenti possono accedere alla versione della funzione utilizzando l'alias nome della risorsa Amazon (ARN). Quando si distribuisce una nuova versione, è possibile aggiornare l'alias per utilizzare la nuova versione o dividere il traffico tra due versioni. Per ulteriori informazioni sull'utilizzo degli alias nelle applicazioni, vedere [the section called “Utilizzo di alias”](#).

### Console

Per creare un alias tramite la console

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegliere Aliases (Alias) e quindi Create alias (Crea alias).
4. Nella pagina Create alias (Crea alias), eseguire le operazioni seguenti:
  - a. Immettere un Name (Nome) per la query.
  - b. (Facoltativo) Immettere una Description (Descrizione) per l'alias.
  - c. Per Version (Versione), scegliere una versione della funzione a cui si desidera puntare l'alias.
  - d. (Facoltativo) Per configurare il routing sull'alias, espandere Weighted alias (Alias ponderato). Per ulteriori informazioni, consulta [Implementare le implementazioni canary Lambda utilizzando un alias ponderato](#).
  - e. Scegli Save (Salva).

### AWS CLI

Per creare un alias utilizzando AWS Command Line Interface (AWS CLI), utilizzate il comando <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/create-alias.html> create-alias.

```
aws lambda create-alias \  
  --function-name my-function \  
  --name alias-name \  
  --function-version version-number \  
  --
```

```
--description " "
```

Per modificare un alias in modo che punti a una nuova versione della funzione, utilizza il comando [update-alias](#).

```
aws lambda update-alias \  
  --function-name my-function \  
  --name alias-name \  
  --function-version version-number
```

Per eliminare un alias, utilizza il comando [delete-alias](#).

```
aws lambda delete-alias \  
  --function-name my-function \  
  --name alias-name
```

I AWS CLI comandi nei passaggi precedenti corrispondono alle seguenti operazioni dell'API Lambda:

- [CreateAlias](#)
- [UpdateAlias](#)
- [DeleteAlias](#)

## Utilizzo degli alias Lambda nelle origini eventi e nelle policy di autorizzazioni

Ogni alias ha un ARN univoco. Un alias può puntare solo a una versione della funzione, non a un altro alias. È possibile aggiornare un alias in modo che punti a una nuova versione della funzione.

Le origini eventi, ad esempio Amazon Simple Storage Service (Amazon S3), richiamano la funzione Lambda. Queste origini eventi mantengono un mapping che identifica la funzione da richiamare quando si verificano gli eventi. Se si specifica un alias di funzione Lambda nella configurazione del mapping, non è necessario aggiornare il mapping quando cambia la versione della funzione. Per ulteriori informazioni, consulta [In che modo Lambda elabora i record provenienti da origini eventi basate su flussi e code](#).

In una policy di risorsa, è possibile concedere le autorizzazioni per le origini eventi per utilizzare la funzione Lambda. Se si specifica un ARN di alias nella policy, non è necessario aggiornare la policy quando cambia la versione della funzione.

### Policy delle risorse

È possibile utilizzare una [policy basata sulle risorse](#) per concedere a un servizio, a una risorsa o a un account l'accesso alla funzione. L'ambito di tale autorizzazione dipende dal fatto che venga applicata a un alias, a una versione o all'intera funzione. Ad esempio, se si utilizza un nome alias (ad esempio `helloworld:PROD`), è possibile richiamare la `helloworld` funzione utilizzando l'alias ARN (`helloworld:PROD`).

Se si tenta di richiamare la funzione senza un alias o una versione specifica, viene visualizzato un errore di autorizzazione. Questo errore di autorizzazione si verifica anche se si tenta di richiamare direttamente la versione della funzione associata all'alias.

Ad esempio, il AWS CLI comando seguente concede ad Amazon S3 le autorizzazioni per richiamare l'alias `PROD` della funzione quando Amazon S3 agisce per conto di `helloworld`. `amzn-s3-demo-bucket`

```
aws lambda add-permission \
  --function-name helloworld \
  --qualifier PROD \
  --statement-id 1 \
  --principal s3.amazonaws.com \
  --action lambda:InvokeFunction \
  --source-arn arn:aws:s3:::amzn-s3-demo-bucket \
  --source-account 123456789012
```

Per ulteriori informazioni sull'utilizzo dei nomi delle risorse nelle policy, consulta [Fine-tuning le sezioni Risorse e condizioni delle politiche](#).

## Implementare le implementazioni canary Lambda utilizzando un alias ponderato

È possibile utilizzare un alias ponderato per suddividere il traffico tra due diverse [versioni](#) della stessa funzione. Con questo approccio, puoi testare nuove versioni delle tue funzioni con una piccola percentuale di traffico e ripristinarle rapidamente se necessario. Questa operazione è nota come [implementazione canary](#). Le implementazioni di Canary si differenziano dalle blue/green distribuzioni perché espongono la nuova versione solo a una parte delle richieste anziché cambiare tutto il traffico contemporaneamente.

È possibile puntare un alias a un massimo di due versioni della funzione Lambda. Le versioni devono soddisfare i seguenti criteri:

- Entrambe le versioni devono disporre dello stesso [ruolo di esecuzione](#).
- Entrambe le versioni devono avere la stessa configurazione della [coda dead-letter](#) o nessuna configurazione della coda dead-letter.
- Entrambe le versioni devono essere pubblicate. L'alias non può puntare a \$LATEST.

### Note

Lambda utilizza un modello probabilistico semplice per distribuire il traffico tra le due versioni delle funzioni. A livelli di traffico bassi, è possibile che si verifichi una variazione elevata tra la percentuale di traffico configurata e quella effettiva in ciascuna versione. Se la tua funzione utilizza la concorrenza con provisioning, puoi evitare [invocazioni spillover](#) configurando un numero maggiore di istanze di concorrenza sottoposte a provisioning durante il periodo in cui il routing degli alias è attivo.



## Creare un alias ponderato

### Console

Per configurare il routing utilizzando la console

#### Note

Verificare che la funzione abbia almeno due versioni pubblicate. Per creare versioni aggiuntive, seguire le istruzioni in [Creazione di versioni delle funzioni](#).

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegliere Aliases (Alias) e quindi Create alias (Crea alias).
4. Nella pagina Create alias (Crea alias), eseguire le operazioni seguenti:
  - a. Immettere un Name (Nome) per la query.
  - b. (Facoltativo) Immettere una Description (Descrizione) per l'alias.
  - c. Per Version (Versione), scegliere la prima versione della funzione a cui si desidera puntare l'alias.
  - d. Espandere Weighted alias (Alias ponderato).
  - e. In Additional version (Versione aggiuntiva), scegliere la seconda versione della funzione a cui si desidera puntare l'alias.
  - f. Per Weight (%) (Ponderazione %), digitare un valore di ponderazione per la funzione. Weight (Ponderazione) è la percentuale di traffico assegnata alla versione quando l'alias viene invocato. La prima versione riceve la ponderazione residua. Ad esempio se si specifica il 10 per cento per Additional version (Versione aggiuntiva), alla prima versione viene automaticamente assegnato il 90 per cento.
  - g. Scegli Save (Salva).

### AWS CLI

Utilizza i comandi [create-alias](#) e [update-alias](#) AWS CLI per configurare i pesi del traffico tra due versioni di funzioni. Quando crei o aggiorni l'alias, specifica il peso del traffico nel parametro `routing-config`.

Nell'esempio seguente viene creato un alias della funzione Lambda denominato `routing-alias` che punta alla versione 1 della funzione. La versione 2 della funzione riceve il 3% del traffico. Il restante 97 per cento del traffico viene instradato alla versione 1.

```
aws lambda create-alias \  
  --name routing-alias \  
  --function-name my-function \  
  --function-version 1 \  
  --routing-config AdditionalVersionWeights={"2":0.03}
```

Utilizza il comando `update-alias` per aumentare la percentuale di traffico in ingresso alla versione 2. Nell'esempio seguente, aumenti il traffico al 5%.

```
aws lambda update-alias \  
  --name routing-alias \  
  --function-name my-function \  
  --routing-config AdditionalVersionWeights={"2":0.05}
```

Per instradare tutto il traffico alla versione 2, utilizza il comando `update-alias` per modificare la proprietà `function-version` in modo che l'alias punti alla versione 2. Il comando reimposta anche la configurazione di routing.

```
aws lambda update-alias \  
  --name routing-alias \  
  --function-name my-function \  
  --function-version 2 \  
  --routing-config AdditionalVersionWeights={}
```

I AWS CLI comandi nei passaggi precedenti corrispondono alle seguenti operazioni dell'API Lambda:

- [CreateAlias](#)
- [UpdateAlias](#)

## Determinazione della versione richiamata

Quando configuri i pesi del traffico tra due versioni di funzioni, esistono due modi per determinare la versione della funzione Lambda invocata:

- CloudWatch Log: Lambda emette automaticamente una voce di START registro che contiene l'ID della versione richiamata per ogni chiamata di funzione. Esempio:

```
START RequestId: 1dh194d3759ed-4v8b-a7b4-1e541f60235f Version: 2
```

Per le invocazioni di alias Lambda utilizza la dimensione ExecutedVersion per filtrare i dati del parametro dalla versione richiamata. Per ulteriori informazioni, consulta [Visualizzazione di parametri per le funzioni Lambda](#).

- Payload della risposta (invocazioni sincrone) - Le risposte a invocazioni sincrone della funzione includono un'intestazione x-amz-executed-version per indicare quale versione della funzione è stata invocata.

Creare una implementazione continua con alias ponderati

Usa AWS CodeDeploy and AWS Serverless Application Model (AWS SAM) per creare una distribuzione progressiva che rilevi automaticamente le modifiche al codice funzione, distribuisca una nuova versione della funzione e aumenti gradualmente la quantità di traffico che fluisce verso la nuova versione. La quantità di traffico e la velocità dell'aumento sono parametri che è possibile configurare.

In una distribuzione progressiva, esegue le seguenti attività: AWS SAM

- Configura la funzione Lambda e crea un alias. La configurazione del routing con alias ponderati è la funzionalità che implementa l'implementazione in sequenza.
- Crea un' CodeDeploy applicazione e un gruppo di distribuzione. Il gruppo di distribuzione gestisce l'implementazione in sequenza e il rollback (se necessario).
- Rileva quando viene creata una nuova versione della funzione Lambda.
- Trigger CodeDeploy per avviare la distribuzione della nuova versione.

Esempio AWS SAM modello

Nell'esempio seguente viene illustrato un [modello AWS SAM](#) per una semplice distribuzione in sequenza.

```
AWSTemplateFormatVersion : '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Description: A sample SAM template for deploying Lambda functions
```

```
Resources:
# Details about the myDateTimeFunction Lambda function
myDateTimeFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: myDateTimeFunction.handler
    Runtime: nodejs24.x
# Creates an alias named "live" for the function, and automatically publishes when you
update the function.
  AutoPublishAlias: live
  DeploymentPreference:
# Specifies the deployment configuration
  Type: Linear10PercentEvery2Minutes
```

Questo modello definisce una funzione Lambda denominata `myDateTimeFunction` con le seguenti proprietà.

### AutoPublishAlias

La proprietà `AutoPublishAlias` crea un alias denominato `live`. Inoltre, il framework AWS SAM rileva automaticamente quando viene salvato il nuovo codice per la funzione. Il framework pubblica quindi una nuova versione di funzione e aggiorna l'alias `live` in modo che punti alla nuova versione.

### DeploymentPreference

La `DeploymentPreference` proprietà determina la velocità con cui l'applicazione CodeDeploy sposta il traffico dalla versione originale della funzione Lambda alla nuova versione. Il valore `Linear10PercentEvery2Minutes` sposta un ulteriore 10% del traffico verso la nuova versione ogni 2 minuti.

Per l'elenco delle configurazioni di distribuzione predefinite, consulta [Configurazioni di distribuzione](#).

Per ulteriori informazioni su come creare implementazioni continue con CodeDeploy e AWS SAM, consulta quanto segue:

- [Tutorial: implementa una funzione Lambda aggiornata con e CodeDeploy AWS Serverless Application Model](#)
- [Distribuzione graduale di applicazioni serverless con AWS SAM](#)

## Gestire le versioni delle funzioni Lambda

Puoi utilizzare le versioni per gestire l'implementazione delle funzioni. Ad esempio, è possibile pubblicare una nuova versione di una funzione per il test beta senza influire sugli utenti della versione di produzione stabile. Lambda crea una nuova versione della funzione ogni volta che pubblichi la funzione. La nuova versione è una copia della versione non pubblicata della funzione. La versione non pubblicata è denominata `$LATEST`.

È importante sottolineare che ogni volta che implementi il codice della funzione, sovrascrivi il codice corrente in `$LATEST`. Per salvare l'iterazione corrente di `$LATEST`, crea una nuova versione della funzione. Se `$LATEST` è identica a una versione pubblicata in precedenza, non potrai creare una nuova versione finché non implementerai le modifiche su `$LATEST`. Queste modifiche possono includere l'aggiornamento del codice o la modifica delle impostazioni di configurazione della funzione.

Dopo aver pubblicato la versione di una funzione, il codice, il runtime, l'architettura, la memoria, i livelli e la maggior parte delle altre impostazioni di configurazione saranno immutabili. Ciò significa che non potrai modificare queste impostazioni senza pubblicare una nuova versione da `$LATEST`. È possibile configurare i seguenti elementi per una versione pubblicata della funzione:

- [Trigger](#)
- [Destinazioni](#)
- [Concorrenza fornita](#)
- [Invocazione asincrona](#)
- [Connessioni a database e proxy](#)

### Note

Quando si utilizzano i [controlli di gestione del runtime](#) con la modalità Auto, la versione di runtime utilizzata dalla versione della funzione viene aggiornata automaticamente. Quando si utilizza la modalità Function update (Aggiornamento delle funzioni) o Manual (Manuale), la versione di runtime non viene aggiornata. Per ulteriori informazioni, consulta [the section called "Aggiornamenti della versione di runtime"](#).

## Sections

- [Creazione di versioni delle funzioni](#)

- [Utilizzo delle versioni](#)
- [Concessione di autorizzazioni](#)

## Creazione di versioni delle funzioni

Puoi modificare il codice della funzione e le impostazioni solo sulla versione non pubblicata di una funzione. Quando pubblichi una versione, il codice e la maggior parte delle impostazioni sono bloccati da Lambda per garantire un'esperienza coerente agli utenti di quella versione.

È possibile creare una versione della funzione utilizzando la console Lambda.

Per creare una nuova versione della funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli una funzione e quindi seleziona la scheda Versioni.
3. Nella pagina di configurazione delle versioni, scegliere Publish new version (Pubblica nuova versione).
4. (Facoltativo) Immettere una descrizione della versione.
5. Seleziona Pubblica.

In alternativa, puoi pubblicare una versione di una funzione utilizzando l'operazione [PublishVersion](#) API.

Il AWS CLI comando seguente pubblica una nuova versione di una funzione. La risposta restituisce le informazioni di configurazione sulla nuova versione, tra cui il numero della versione e l'ARN della funzione con il suffisso della versione.

```
aws lambda publish-version --function-name my-function
```

Verrà visualizzato l'output seguente:

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function:1",
  "Version": "1",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "function.handler",
  "Runtime": "nodejs24.x",
```

```
...  
}
```

### Note

Lambda assegna numeri di sequenza crescenti in modo monotono per il controllo delle versioni. Lambda non riutilizza mai i numeri di versione, anche dopo aver eliminato e ricreato una funzione.

## Utilizzo delle versioni

Puoi fare riferimento alla tua funzione Lambda usando un ARN qualificato o un ARN non qualificato.

- ARN qualificato – L'ARN della funzione con il suffisso della versione. L'esempio seguente fa riferimento alla versione 42 della funzione `helloworld`.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:42
```

- ARN non qualificato – L'ARN della funzione senza il suffisso della versione.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld
```

È possibile utilizzare un ARN qualificato o non qualificato in tutte le operazioni API pertinenti. Tuttavia, non è possibile utilizzare un ARN non qualificato per creare un alias.

Se decidi di non pubblicare le versioni di funzione, puoi richiamare la funzione utilizzando l'ARN qualificato o non qualificato nel [mapping dell'origine eventi](#). Quando richiami una funzione utilizzando un ARN non completo, Lambda richiama implicitamente `$LATEST`.

L'ARN qualificato per ogni versione di funzione Lambda è univoco. Dopo aver pubblicato una versione, non è possibile modificare l'ARN o il codice di funzione.

Lambda pubblica una nuova versione della funzione solo se il codice non è mai stato pubblicato o se il codice è cambiato rispetto all'ultima versione pubblicata. Se non ci sono modifiche, la versione della funzione rimane la versione pubblicata più di recente.

Quando pubblichi una versione, Lambda crea uno snapshot immutabile del codice e della configurazione della funzione. Non tutte le modifiche alla configurazione attivano la pubblicazione

di una nuova versione. Le seguenti modifiche alla configurazione qualificano una funzione per la pubblicazione della versione:

- Codice della funzione
- Variabili di ambiente
- Runtime
- Gestore
- Livelli
- Dimensioni della memoria
- Timeout
- Configurazione VPC
- Configurazione coda DLQ
- Ruolo IAM
- Description
- Architettura (x86\_64 o arm64)
- Dimensioni di archiviazione temporanea
- Tipo di pacchetto
- Modalità di archiviazione del codice (passaggio da uno storage [S3 all'altro Lambda-managed e autogestito](#))
- Configurazione della registrazione
- Configurazione del file system
- SnapStart
- Configurazione del tracciamento

Impostazioni operative come la [simultaneità riservata](#) non attivano la pubblicazione di una nuova versione quando vengono modificate.

## Concessione di autorizzazioni

È possibile utilizzare una [policy basata sulle risorse](#) o una [policy basata sull'identità](#) per concedere l'accesso alla funzione. L'ambito dell'autorizzazione dipende dal fatto che la policy venga applicata a una funzione o a una versione di una funzione. Per ulteriori informazioni sui nomi delle risorse delle funzioni nelle policy, consulta [Fine-tuning le sezioni Risorse e condizioni delle politiche](#).



Puoi semplificare la gestione delle fonti di eventi e delle policy AWS Identity and Access Management (IAM) utilizzando gli alias delle funzioni. Per ulteriori informazioni, consulta [Creare un alias per una funzione Lambda](#).

## Uso dei tag sulle funzioni Lambda

È possibile taggare le funzioni per organizzare e gestire le risorse. I tag sono coppie chiave-valore a forma libera associate alle risorse supportate su Servizi AWS. Per ulteriori informazioni sui casi d'uso dei tag, consulta Strategie di tagging [comuni](#) nella Guida alle AWS risorse sui tag e all'editor di tag.

I tag si applicano a livello di funzione, non a versioni o alias. I tag non fanno parte della configurazione specifica della versione che AWS Lambda crea un'istanza di quando si pubblica una versione. Per visualizzare e aggiornare i tag, puoi utilizzare l'API Lambda. Puoi anche visualizzare e aggiornare i tag mentre gestisci una funzione specifica nella console Lambda.

Puoi anche utilizzare i tag per controllare l'accesso alle tue funzioni con il controllo degli accessi basato sugli attributi (ABAC). Per ulteriori informazioni, consulta [the section called “Attribute-based controllo degli accessi”](#).

### Sections

- [Autorizzazioni necessarie per lavorare con i tag](#)
- [Utilizzo di tag con la console Lambda](#)
- [Usare i tag con AWS CLI](#)

## Autorizzazioni necessarie per lavorare con i tag

Per consentire a un'identità AWS Identity and Access Management (IAM) (utente, gruppo o ruolo) di leggere o impostare tag su una risorsa, concedile le autorizzazioni corrispondenti:

- `lambda: ListTags` —Quando una risorsa ha dei tag, concedi questa autorizzazione a chiunque debba richiamarla. `ListTags` Per le funzioni con tag, questa autorizzazione è necessaria anche per `GetFunction`.
- `lambda: TagResource` —Concedi questa autorizzazione a chiunque debba chiamare `TagResource` o eseguire un tag durante la creazione.

Facoltativamente, valuta la possibilità di concedere anche l'`UntagResource` autorizzazione `lambda:` per consentire `UntagResource` le chiamate alla risorsa.

Per ulteriori informazioni, consulta [Identity-based Politiche IAM per Lambda](#).

## Utilizzo di tag con la console Lambda

Puoi utilizzare la console Lambda per creare funzioni che hanno tag, aggiungere tag a funzioni esistenti e filtrare le funzioni in base ai tag aggiunti.

Per aggiungere tag durante la creazione di una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli Crea funzione.
3. Scegliere Author from scratch (Crea da zero) o Container image (Immagine di container).
4. In Informazioni di base, configura la tua funzione. Per ulteriori informazioni sulla configurazione delle funzioni, consulta [the section called "Configurazione delle funzioni"](#).
5. Espandi Advanced settings (Impostazioni avanzate), quindi seleziona Enable tags (Abilita tag).
6. Scegli Add new tag (Aggiungi nuovo tag) e completa i campi Key (Chiave) e facoltativamente Value (Valore). Ripetere questa fase per aggiungere altri tag.
7. Scegli Crea funzione.

Per aggiungere tag a una funzione esistente

1. Aprire la pagina [Functions](#) (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Scegli Configuration (Configurazione), quindi Tags (Tag).
4. In Tag, scegli Gestisci tag.
5. Scegli Add new tag (Aggiungi nuovo tag) e completa i campi Key (Chiave) e facoltativamente Value (Valore). Ripetere questa fase per aggiungere altri tag.
6. Selezionare Salva.

Per filtrare le funzioni con i tag

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la barra di ricerca per visualizzare un elenco di proprietà della funzione e chiavi di tag.
3. Scegli una chiave tag per visualizzare un elenco di valori in uso nella regione corrente. AWS
4. Seleziona Usa: "nome-tag" per vedere tutte funzioni taggate con questa chiave, oppure scegli un operatore per filtrare ulteriormente in base al valore.

5. Seleziona il valore del tag da filtrare in base a una combinazione di chiave e valore del tag.

La barra di ricerca supporta anche la ricerca di chiavi tag. Immetti tag per visualizzare solo un elenco di chiavi di tag oppure immetti il nome di una chiave per trovarla nell'elenco.

## Usare i tag con AWS CLI

Puoi aggiungere e rimuovere tag sulle risorse Lambda esistenti, incluse le funzioni, con l'API Lambda. È inoltre possibile aggiungere tag durante la creazione di una funzione, per mantenere una risorsa contrassegnata durante l'intero ciclo di vita.

### Aggiornamento dei tag con le API dei tag Lambda

Puoi aggiungere e rimuovere tag per le risorse Lambda supportate tramite le operazioni [TagResource](#) e [UntagResource](#) API.

Puoi chiamare queste operazioni tramite la AWS CLI. Per aggiungere i tag a una risorsa esistente, utilizza il comando `tag-resource`. Questo esempio aggiunge due tag, uno con la chiave *Department* e uno con la chiave *CostCenter*.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:resource-type:my-resource \
--tags Department=Marketing,CostCenter=1234ABCD
```

Per rimuovere i tag, utilizza il comando `untag-resource`. Questo esempio rimuove il tag con la chiave *Department*.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:resource-
type:resource-identifier \
--tag-keys Department
```

### Aggiunta di tag durante la creazione di una funzione

Per creare una nuova funzione Lambda con tag, usa l'operazione [CreateFunction](#) API. Specifica il parametro `Tags`. È possibile richiamare questa operazione con il comando della CLI `create-function` e l'opzione `--tags`.

Prima di utilizzare il parametro `tags` con `CreateFunction`, assicurati che il tuo ruolo disponga dell'autorizzazione per taggare le risorse oltre alle normali autorizzazioni necessarie per questa

operazione. Per ulteriori informazioni sulle autorizzazioni per il tagging, consulta [the section called “Autorizzazioni necessarie per lavorare con i tag”](#). Questo esempio aggiunge due tag, uno con la chiave *Department* e uno con la chiave *CostCenter*.

```
aws lambda create-function --function-name my-function
--handler index.js --runtime nodejs24.x \
--role arn:aws:iam::123456789012:role/lambda-role \
--tags Department=Marketing,CostCenter=1234ABCD
```

### Visualizzazione dei tag su una funzione

Per visualizzare i tag applicati a una risorsa Lambda specifica, utilizza l'operazione API `ListTags`. Per ulteriori informazioni, consulta [ListTags](#).

Puoi chiamare questa operazione con il `list-tags` AWS CLI comando fornendo un ARN (Amazon Resource Name).

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:resource-
type:resource-identifier
```

Puoi visualizzare i tag applicati a una risorsa specifica con l'operazione [GetFunction](#) API. Funzionalità comparabili non sono disponibili per altri tipi di risorse.

È possibile richiamare questa operazione utilizzando il comando `get-function` della CLI:

```
aws lambda get-function --function-name my-function
```

### Filtro delle risorse per tag

Puoi utilizzare l'operazione AWS Resource Groups Tagging API [GetResources](#) API per filtrare le tue risorse per tag. L'operazione `GetResources` riceve fino a 10 filtri, ognuno dei quali contenente una chiave di tag e un massimo di 10 valori di tag. Fornisci `GetResources` con un `ResourceType` per filtrare in base a tipi di risorse specifiche.

Puoi chiamare questa operazione usando il `get-resources` AWS CLI comando. Per esempi di utilizzo di `get-resources`, consulta [get-resources](#) nella Riferimento ai comandi CLI di AWS .

## Streaming di risposte per le funzioni Lambda

Le funzioni Lambda possono trasmettere nativamente i payload di risposta ai client tramite gli URL delle funzioni [Lambda](#) o utilizzando l'[InvokeWithResponseStream](#) API (tramite l' AWS SDK o chiamate API dirette). La tua funzione Lambda può anche trasmettere i payload di risposta tramite l'integrazione del proxy [Amazon API Gateway](#), che utilizza l'API per richiamare la tua funzione. [InvokeWithResponseStream](#) Lo streaming delle risposte può favorire le applicazioni sensibili alla latenza migliorando le prestazioni del time to first byte (TTFB). Questo perché consente di inviare risposte parziali al client non appena diventano disponibili. Inoltre, le funzioni di streaming di risposta possono restituire payload fino a 200 MB, rispetto al massimo di 6 MB per le risposte bufferizzate.

Lo streaming di una risposta significa anche che la funzione non deve contenere l'intera risposta in memoria. Per risposte molto grandi, ciò può ridurre la quantità di memoria necessaria per configurare la funzione. Per aiutarti a scegliere tra gli URL delle funzioni e API Gateway in base al tuo caso d'uso, consulta [the section called “URL delle funzioni e Amazon API Gateway”](#).

### Note

Lo streaming di risposte Lambda non è ancora disponibile in tutto. Regioni AWS Per la disponibilità delle funzionalità per regione, consulta [AWS Funzionalità per regione](#) del Builder Center.

La velocità con cui Lambda trasmette le tue risposte dipende dalla dimensione della risposta. La velocità di streaming per i primi 6 MB di risposta della funzione è illimitata. Per le risposte superiori a 6 MB, il resto della risposta è soggetto a un limite di larghezza di banda. Per ulteriori informazioni sulla larghezza di banda dello streaming, consulta la sezione [Limiti di larghezza di banda per lo streaming delle risposte](#).

Le risposte in streaming comportano dei costi e le risposte in streaming non vengono interrotte o interrotte quando la connessione del client richiamante si interrompe. Ai clienti viene addebitata l'intera durata della funzione, pertanto i clienti devono prestare attenzione quando configurano lunghi timeout di funzionamento.

Lambda supporta lo streaming delle risposte su runtime gestiti. Node.js Per altri linguaggi, incluso Python, puoi [utilizzare un runtime personalizzato con un'integrazione API Runtime personalizzata](#) per lo streaming delle risposte o utilizzare il [Lambda Web Adapter](#).

 Note

Quando testi la funzione tramite la console Lambda, vedrai sempre le risposte come memorizzate nel buffer.

Per imparare a scrivere funzioni di streaming, consulta [the section called “Funzioni di scrittura”](#). Per richiamare le funzioni di streaming tramite gli URL delle funzioni, consulta [the section called “Richiamo di funzioni”](#). Per la procedura dettagliata, consulta [the section called “Tutorial: creazione di una funzione di streaming delle risposte con un URL della funzione”](#).

## Argomenti

- [Limiti di larghezza di banda per lo streaming delle risposte](#)
- [Compatibilità VPC con streaming di risposta](#)
- [Scrittura di funzioni Lambda abilitate allo streaming delle risposte](#)
- [Richiamo di una funzione abilitata allo streaming di risposte utilizzando la funzione Lambda URLs](#)
- [Tutorial: creazione di una funzione Lambda di streaming delle risposte con un URL della funzione](#)

## Limiti di larghezza di banda per lo streaming delle risposte

I primi 6 MB del payload di risposta della funzione hanno una larghezza di banda illimitata. Dopo questa espansione iniziale, Lambda trasmette la tua risposta a una velocità massima di 2 MB/s. Se le risposte delle tue funzioni non superano mai i 6 MB, questo limite di larghezza di banda non verrà mai applicato.

 Note

I limiti di larghezza di banda si applicano solo al payload di risposta della funzione e non all'accesso alla rete da parte della funzione.

La velocità di larghezza di banda illimitata varia in base a una serie di fattori, inclusa la velocità di elaborazione della funzione. Normalmente ci si può aspettare una velocità superiore a 2 MBps per i primi 6 MB di risposta della funzione. Se la tua funzione è lo streaming di una risposta verso una destinazione esterna AWS, la velocità di streaming dipende anche dalla velocità della connessione Internet esterna.

## Compatibilità VPC con streaming di risposta

Quando utilizzi le funzioni Lambda in un ambiente VPC, per lo streaming di risposta è importante tenere in considerazione quanto segue:

- Gli URL delle funzioni Lambda non supportano lo streaming di risposta all'interno di un ambiente VPC.
- Puoi utilizzare lo streaming di risposte all'interno di un VPC richiamando la tua funzione Lambda tramite l' AWS SDK utilizzando l'API. `InvokeWithResponseStream` Ciò richiede la configurazione degli endpoint VPC appropriati per Lambda.
- Per gli ambienti VPC, dovrai creare un endpoint VPC di interfaccia per Lambda per abilitare la comunicazione tra le tue risorse nel VPC e il servizio Lambda.

Un'architettura tipica per lo streaming di risposta in un VPC potrebbe includere:

```
Client in VPC -> Interface VPC endpoint for Lambda -> Lambda function -> Response  
streaming back through the same path
```

## Scrittura di funzioni Lambda abilitate allo streaming delle risposte

La scrittura del gestore per le funzioni di streaming delle risposte è diversa dai modelli di gestore tipici. Quando scrivi funzioni di streaming, assicurati di completare le seguenti operazioni:

- Completa la funzione con il decoratore `awslambda.streamifyResponse()`. L'oggetto globale `awslambda` è fornito dall'ambiente di runtime Node.js di Lambda.
- Termina il flusso in modo corretto per assicurarti che tutta l'elaborazione dei dati sia completa.

### Configurazione di un gestore delle funzioni per lo streaming delle risposte

Per indicare al runtime che Lambda deve trasmettere in streaming le risposte della funzione, è necessario racchiudere la funzione con il decoratore `streamifyResponse()`. Questo indica al runtime di utilizzare il percorso logico corretto per lo streaming delle risposte e consente alla funzione di trasmettere le risposte.

Il decoratore `streamifyResponse()` accetta una funzione che accetta i seguenti parametri:

- `event`: fornisce informazioni sull'evento di chiamata dell'URL della funzione, ad esempio il metodo HTTP, i parametri della query e il corpo della richiesta.



- `responseStream`: fornisce un flusso scrivibile.
- `context`: fornisce i metodi e le proprietà con informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

L'oggetto `responseStream` è un [writableStream Node.js](#). Come con qualsiasi flusso di questo tipo, dovresti usare il metodo `pipeline()`.

#### Note

L'oggetto globale `awsLambda` viene fornito automaticamente dal runtime Node.js di Lambda e non è richiesta alcuna importazione.

### Example gestore abilitato allo streaming delle risposte

```
import { pipeline } from 'node:stream/promises';
import { Readable } from 'node:stream';

export const echo = awsLambda.streamifyResponse(async (event, responseStream, _context) => {
  // As an example, convert event to a readable stream.
  const requestStream = Readable.from(Buffer.from(JSON.stringify(event)));

  await pipeline(requestStream, responseStream);
});
```

Sebbene `responseStream` offra il metodo `write()` per scrivere sul flusso, ti consigliamo di utilizzare [pipeline\(\)](#) laddove possibile. L'utilizzo di `pipeline()` garantisce che il flusso scrivibile non venga sopraffatto da un flusso leggibile più veloce.

### Terminazione dello streaming

Assicurati di terminare correttamente il flusso prima che torni al gestore. Il metodo `pipeline()` gestisce questo aspetto automaticamente.

Per altri casi d'uso, chiama il metodo `responseStream.end()` per terminare correttamente un flusso. Questo metodo segnala che nel flusso non devono essere scritti altri dati. Questo metodo non è necessario se si scrive nel flusso con `pipeline()` o `pipe()`.

A partire da Node.js 24, Lambda non attende più che le promesse non risolte vengano completate dopo il ritorno del gestore o il termine del flusso di risposta. Se la tua funzione dipende da operazioni asincrone aggiuntive, come timer o recuperi, dovresti inserirle nel tuo gestore. `await`

Example Esempio di terminazione di un flusso con `pipeline()`

```
import { pipeline } from 'node:stream/promises';

export const handler = awslambda.streamifyResponse(async (event, responseStream,
  _context) => {
  await pipeline(requestStream, responseStream);
});
```

Example Esempio di terminazione di un flusso senza `pipeline()`

```
export const handler = awslambda.streamifyResponse(async (event, responseStream,
  _context) => {
  responseStream.write("Hello ");
  responseStream.write("world ");
  responseStream.write("from ");
  responseStream.write("Lambda!");
  responseStream.end();
});
```

## Richiamo di una funzione abilitata allo streaming di risposte utilizzando la funzione Lambda URLs

### Note

La tua funzione Lambda può ora trasmettere in streaming i payload di risposta tramite l'integrazione proxy di [Amazon API Gateway](#).

Puoi richiamare le funzioni abilitate allo streaming delle risposte modificando la modalità di richiamo dell'URL della funzione. La modalità di richiamo determina quale operazione API Lambda utilizza per richiamare la funzione. Le modalità di richiamo disponibili sono:

- **BUFFERED**: questa è l'opzione predefinita. Lambda richiama la funzione utilizzando l'operazione API Invoke. I risultati delle chiamate sono disponibili quando il payload è completo. La dimensione massima del payload è pari a 6 MB.

- **RESPONSE\_STREAM**: consente alla funzione di trasmettere in streaming i risultati del payload non appena diventano disponibili. Lambda richiama la funzione utilizzando l'operazione API `InvokeWithResponseStream`. La dimensione massima del payload di risposta è di 200 MB.

Puoi comunque richiamare la funzione senza lo streaming delle risposte chiamando direttamente l'operazione API `Invoke`. Tuttavia, Lambda trasmette in streaming tutti i payload di risposta per le chiamate che arrivano tramite l'URL della funzione fino a quando non si modifica la modalità di richiamo in **BUFFERED**.

## Console

### Creazione di un URL della funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione per la quale desideri impostare la modalità di richiamo.
3. Scegli la scheda Configurazione, quindi scegli URL della funzione.
4. Scegli Modifica, quindi scegli Impostazioni aggiuntive.
5. In Modalità di richiamo, scegli la modalità di richiamo desiderata.
6. Scegli Save (Salva).

## AWS CLI

### Per impostare la modalità di richiamo di un URL della funzione (AWS CLI)

```
aws lambda update-function-url-config \  
  --function-name my-function \  
  --invoke-mode RESPONSE_STREAM
```

## CloudFormation

### Per impostare la modalità di richiamo di un URL della funzione (CloudFormation)

```
MyFunctionUrl:  
  Type: AWS::Lambda::Url  
  Properties:  
    AuthType: AWS_IAM  
    InvokeMode: RESPONSE_STREAM
```

Per ulteriori informazioni sulla configurazione della funzione URLs, vedere Funzione [URLsLambda](#).

## Tutorial: creazione di una funzione Lambda di streaming delle risposte con un URL della funzione

In questo tutorial viene creata una funzione Lambda definita come archivio file .zip con un endpoint funzione URL che restituisce un flusso di risposte. Per ulteriori informazioni sulla configurazione della funzione URLs, vedere [URL delle funzioni](#).

### Prerequisiti

Questo tutorial presuppone una certa conoscenza delle operazioni di base di Lambda e della console relativa. Se non lo si è già fatto, seguire le istruzioni riportate in [Creare una funzione Lambda con la console](#) per creare la prima funzione Lambda.

Per completare i passaggi seguenti, è necessaria la [AWS CLI versione 2](#). I comandi e l'output previsto sono elencati in blocchi separati:

```
aws --version
```

Verrà visualizzato l'output seguente:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Per i comandi lunghi viene utilizzato un carattere di escape (\) per dividere un comando su più righe.

In Linux e macOS utilizzare la propria shell e il proprio programma di gestione dei pacchetti preferiti.

#### Note

Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, zip) non sono supportati dai terminali integrati del sistema operativo. Per ottenere una versione integrata su Windows di Ubuntu e Bash, [installa il sottosistema Windows per Linux](#). I comandi della CLI di esempio in questa guida utilizzano la formattazione Linux. Se si utilizza la CLI di Windows, i comandi che includono documenti JSON in linea dovranno essere riformattati.

## Creazione di un ruolo di esecuzione

Creare il [ruolo di esecuzione](#) che offre l'autorizzazione alla funzione Lambda per accedere alle risorse AWS .

## Creazione di un ruolo di esecuzione

1. Aprire la pagina [Roles \(Ruoli\)](#) della console IAM AWS Identity and Access Management .
2. Scegli Crea ruolo.
3. Creare un ruolo con le seguenti proprietà:
  - Tipo di entità affidabile: servizio di AWS
  - Caso d'uso: Lambda
  - Autorizzazioni: AWSLambdaBasicExecutionRole
  - Nome ruolo – **response-streaming-role**

La AWSLambdaBasicExecutionRolepolicy dispone delle autorizzazioni necessarie alla funzione per scrivere log su Amazon CloudWatch Logs. Una volta creato il ruolo, prendi nota del relativo nome della risorsa Amazon (ARN). Questo valore servirà nella fase successiva.

## Creazione di una funzione di streaming delle risposte (AWS CLI)

Crea una funzione Lambda di streaming delle risposte con un endpoint URL della funzione utilizzando l' AWS Command Line Interface (AWS CLI).

## Creazione di una funzione in grado di trasmettere le risposte

1. Copiare il codice di esempio seguente in un file denominato `index.js`. Questa funzione trasmette tre risposte, separate da 1 secondo.

```
exports.handler = awslambda.streamifyResponse(
  async (event, responseStream, _context) => {
    // Metadata is a JSON serializable JS object. Its shape is not defined here.
    const metadata = {
      statusCode: 200,
      headers: {
        "Content-Type": "application/json",
        "CustomHeader": "outerspace"
      }
    };
  });
```

```
// Assign to the responseStream parameter to prevent accidental reuse of the non-
wrapped stream.
responseStream = awslambda.HttpResponseStream.from(responseStream, metadata);

responseStream.write("Streaming with Helper \n");
await new Promise(r => setTimeout(r, 1000));
responseStream.write("Hello 0 \n");
await new Promise(r => setTimeout(r, 1000));
responseStream.write("Hello 1 \n");
await new Promise(r => setTimeout(r, 1000));
responseStream.write("Hello 2 \n");
await new Promise(r => setTimeout(r, 1000));
responseStream.end();
await responseStream.finished();
}
);
```

## 2. Crea un pacchetto di implementazione.

```
zip function.zip index.js
```

## 3. Creare una funzione Lambda con il comando `create-function`. Sostituisci il valore di `--role` con l'ARN del ruolo del passaggio precedente. Questo comando imposta il timeout della funzione su 10 secondi, il che consente alla funzione di trasmettere tre risposte.

```
aws lambda create-function \
  --function-name my-streaming-function \
  --runtime nodejs24.x \
  --zip-file fileb://function.zip \
  --handler index.handler \
  --timeout 10 \
  --role arn:aws:iam::123456789012:role/response-streaming-role
```

## Creazione di un URL della funzione

1. Aggiungi alla tua funzione una politica basata sulle risorse che concede e autorizzazioni. `lambda:InvokeFunctionUrl` `lambda:InvokeFunction` Ogni istruzione deve essere aggiunta in un comando separato. Sostituisci il valore di `--principal` con il tuo Account AWS ID.

```
aws lambda add-permission \  
  --function-name my-streaming-function \  
  --action lambda:InvokeFunctionUrl \  
  --statement-id UrlPolicyInvokeURL \  
  --principal 123456789012 \  
  --function-url-auth-type AWS_IAM
```

```
aws lambda add-permission \  
  --function-name my-streaming-function \  
  --action lambda:InvokeFunction \  
  --statement-id UrlPolicyInvokeFunction \  
  --principal 123456789012
```

2. Crea un endpoint URL per la funzione con il comando `create-function-url-config`.

```
aws lambda create-function-url-config \  
  --function-name my-streaming-function \  
  --auth-type AWS_IAM \  
  --invoke-mode RESPONSE_STREAM
```

#### Note

Se ricevi un errore in merito `--invoke-mode`, potrebbe essere necessario eseguire l'aggiornamento a una [versione più recente di AWS CLI](#).

Verifica l'endpoint URL della funzione

Testa l'integrazione richiamando la tua funzione. Puoi aprire l'URL della funzione in un browser oppure puoi usare curl.

```
curl --request GET "https://abcdefghijklm7nop7qrs740abcd.lambda-url.us-east-1.on.aws/"  
--user "AKIAIOSFODNN7EXAMPLE" --aws-sigv4 "aws:amz:us-east-1:lambda" --no-buffer
```

La nostra funzione URL utilizza il tipo di autenticazione `IAM_AUTH`. Ciò implica che devi firmare le richieste sia con la [chiave di accesso AWS che con la chiave segreta](#). Nel comando precedente, sostituiscilo `AKIAIOSFODNN7EXAMPLE` con l'ID della chiave di AWS accesso. Inserisci la tua chiave AWS segreta quando richiesto. Se non disponi della chiave AWS segreta, puoi invece [utilizzare AWS credenziali temporanee](#).

Dovresti ottenere una risposta simile a questa:

```
Streaming with Helper  
Hello 0  
Hello 1  
Hello 2
```

## Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili a tuo carico. Account AWS

### Come eliminare il ruolo di esecuzione

1. Aprire la pagina [Ruoli](#) della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

### Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **confirm** nel campo di immissione testo e scegli Delete (Elimina).



## Utilizzo dell'endpoint dei metadati Lambda

Con l'endpoint di metadati Lambda, le tue funzioni possono scoprire in quale zona di disponibilità (AZ) sono in esecuzione. Puoi utilizzarlo per ottimizzare la latenza indirizzando verso risorse della stessa AZ come gli endpoint Amazon ElastiCache e Amazon RDS e per implementare modelli di resilienza. AZ-aware

L'endpoint restituisce i metadati in un semplice formato JSON tramite un'API HTTP localhost all'interno dell'ambiente di esecuzione ed è accessibile sia ai runtime che alle estensioni.

### Sections

- [Nozioni di base](#)
- [Comprensione degli ID delle zone di disponibilità](#)
- [Guida di riferimento alle API](#)

### Nozioni di base

[Powertools for AWS Lambda](#) fornisce un'utilità per accedere all'endpoint dei metadati Lambda in Python, Java e .NET. TypeScript L'utilità memorizza nella cache la risposta dopo la prima chiamata e gestisce automaticamente l'invalidazione della cache. SnapStart

Utilizza l'utilità di metadati Powertools for AWS Lambda o chiama direttamente l'endpoint dei metadati

### Python

Installa il pacchetto Powertools:

```
pip install "aws-lambda-powertools"
```

Usa l'utilità per i metadati nel tuo gestore:

Example Recupero dell'ID AZ con Powertools (Python)

```
from aws_lambda_powertools.utilities.lambda_metadata import get_lambda_metadata

def handler(event, context):
    metadata = get_lambda_metadata()
    az_id = metadata.availability_zone_id # e.g., "use1-az1"

    return {"az_id": az_id}
```

## TypeScript

Installa il pacchetto Powertools:

```
npm install @aws-lambda-powertools/commons
```

Usa l'utilità per i metadati nel tuo gestore:

Example Recupero dell'ID AZ con Powertools () TypeScript

```
import { getMetadata } from '@aws-lambda-powertools/commons/utils/metadata';

const metadata = await getMetadata();

export const handler = async () => {
  const { AvailabilityZoneID: azId } = metadata;
  return azId;
};
```

## Java

Aggiungi la dipendenza Powertools al tuo: pom.xml

```
<dependencies>
  <dependency>
    <groupId>software.amazon.lambda</groupId>
    <artifactId>powertools-lambda-metadata</artifactId>
    <version>2.10.0</version>
  </dependency>
</dependencies>
```

Usa il client di metadati nel tuo gestore:

Example Recupero dell'ID AZ con Powertools (Java)

```
import software.amazon.lambda.powertools.metadata.LambdaMetadata;
import software.amazon.lambda.powertools.metadata.LambdaMetadataClient;

public class App implements RequestHandler<Object, String> {

  @Override
  public String handleRequest(Object input, Context context) {
    LambdaMetadata metadata = LambdaMetadataClient.get();
```

```
        String azId = metadata.getAvailabilityZoneId(); // e.g., "use1-az1"

        return "{\"azId\": \"" + azId + "\"}";
    }
}
```

## .NET

Installa il pacchetto Powertools:

```
dotnet add package AWS.Lambda.Powertools.Metadata
```

Usa la classe di metadati nel tuo gestore:

Example Recupero dell'ID AZ con Powertools (.NET)

```
using AWS.Lambda.Powertools.Metadata;

public class Function
{
    public string Handler(object input, ILambdaContext context)
    {
        var azId = LambdaMetadata.AvailabilityZoneId;
        return $"Running in AZ: {azId}";
    }
}
```

## All Runtimes

Tutti i runtime Lambda supportano l'endpoint dei metadati, inclusi i runtime personalizzati e le immagini dei contenitori. Usa il seguente esempio per accedere all'API dei metadati direttamente dalla tua funzione utilizzando le variabili di ambiente che Lambda imposta automaticamente nell'ambiente di esecuzione.

Example Accesso diretto all'endpoint dei metadati

```
# Variables are automatically set by Lambda
METADATA_ENDPOINT="http://${AWS_LAMBDA_METADATA_API}/2026-01-15/metadata/execution-
environment"

# Make the request
RESPONSE=$(curl -s -H "Authorization: Bearer ${AWS_LAMBDA_METADATA_TOKEN}"
"$METADATA_ENDPOINT")
```

```
# Parse the AZ ID
AZ_ID=$(echo "$RESPONSE" | jq -r '.AvailabilityZoneID')

echo "Function is running in AZ ID: $AZ_ID"
```

## Comprensione degli ID delle zone di disponibilità

Gli ID AZ (ad esempio `use1-az1`) si riferiscono sempre alla stessa posizione fisica in tutti AWS gli account, mentre i nomi AZ (ad esempio `us-east-1a`) potrebbero essere associati a infrastrutture fisiche diverse in ciascun AWS account in determinate regioni. Per ulteriori informazioni, consulta [ID AZ per la coerenza](#) tra più account.

Conversione dell'ID AZ in nome AZ:

Per convertire un ID AZ in un nome AZ, utilizza l'API Amazon EC2 [DescribeAvailabilityZones](#). Per utilizzare questa API, aggiungi `ec2:DescribeAvailabilityZones` autorizzazione al ruolo di esecuzione della tua funzione.

## Guida di riferimento alle API

### Variabili di ambiente

Lambda imposta automaticamente le seguenti variabili di ambiente in ogni ambiente di esecuzione:

- `AWS_LAMBDA_METADATA_API`— L'indirizzo del server di metadati nel formato `{ipv4_address}:{port}` (ad esempio, `169.254.100.1:9001`)
- `AWS_LAMBDA_METADATA_TOKEN`— Un token di autenticazione univoco per l'ambiente di esecuzione corrente. Lambda genera questo token automaticamente al momento dell'inizializzazione. Includilo in tutte le richieste API di metadati.

### Endpoint

GET `http://${AWS_LAMBDA_METADATA_API}/2026-01-15/metadata/execution-environment`

### Richiesta

Intestazioni obbligatorie:

- **Authorization**— Il valore del token derivante dalla variabile di ambiente `AWS_LAMBDA_METADATA_TOKEN` con lo schema `Bearer: Bearer <token>`. Questa autenticazione basata su token fornisce una protezione approfondita contro le vulnerabilità di Server-Side Request Forgery (SSRF). Ogni ambiente di esecuzione riceve un token univoco generato casualmente al momento dell'inizializzazione.

## Risposta

Stato: 200 OK

Content-Type: application/json

Cache-Control: private, max-age=43200, immutable

La risposta è immutabile all'interno di un ambiente di esecuzione. I client devono memorizzare la risposta nella cache e rispettare il `Cache-Control` TTL. Per quanto riguarda SnapStart le funzioni, il TTL viene ridotto durante l'inizializzazione in modo che i client aggiornino i metadati dopo il ripristino quando l'ambiente di esecuzione potrebbe trovarsi in una AZ diversa. Se si utilizza Powertools, la memorizzazione nella cache e l'invalidazione vengono gestiti automaticamente. SnapStart

Testo:

```
{
  "AvailabilityZoneID": "use1-az1"
}
```

Il `AvailabilityZoneID` campo contiene l'identificatore univoco della zona di disponibilità in cui è in esecuzione l'ambiente di esecuzione.

### Note

Ulteriori campi potrebbero essere aggiunti alla risposta negli aggiornamenti futuri. I client devono ignorare i campi sconosciuti e non fallire se vengono visualizzati nuovi campi.

## Risposte agli errori

- 401 Non autorizzato: l'`Authorization` intestazione è mancante o contiene un token non valido. Verifica di aver superato il test. `Bearer ${AWS_LAMBDA_METADATA_TOKEN}`
- Metodo 405 non consentito: il metodo di richiesta non GET lo è.

- Errore interno del server 500: errore di Server-side elaborazione.

## Comprensione dei metodi di invocazione della funzione Lambda

Dopo aver implementato la tua funzione Lambda, puoi richiamarla in diversi modi:

- La [console Lambda](#): utilizza la console Lambda per creare rapidamente un evento di test per richiamare la tua funzione.
- L'[AWS SDK](#): utilizza l'SDK per richiamare AWS a livello di codice la tua funzione.
- L'API [Invoke](#): utilizza l'API Lambda Invoke per richiamare direttamente la tua funzione.
- The [AWS Command Line Interface \(AWS CLI\)](#): usa il `aws lambda invoke` AWS CLI comando per richiamare direttamente la tua funzione dalla riga di comando.
- Un [endpoint HTTP\(S\) della funzione URL](#): utilizza le funzioni URL per creare un endpoint HTTP(S) dedicato che è possibile utilizzare per richiamare la funzione.

Tutti questi metodi sono modi diretti per richiamare la tua funzione. In Lambda, un caso d'uso comune consiste nel richiamare la funzione in base a un evento che si verifica altrove nell'applicazione. Alcuni servizi possono richiamare una funzione Lambda con ogni nuovo evento.

Si chiama [trigger](#). Per i servizi basati su flussi e code, Lambda richiama la funzione con batch di record. Ciò è detto [strumento di mappatura dell'origine degli eventi](#).

Quando si invoca una funzione, è possibile scegliere di invocarla in modo sincrono o asincrono. Con l'[invocazione sincrona](#), è necessario attendere che la funzione elabori l'evento e restituisca una risposta. Con l'[invocazione asincrona](#), Lambda accoda l'evento per l'elaborazione e restituisce una risposta immediatamente. Il [parametro della richiesta InvocationType nell'API Invoke](#) determina il modo in cui Lambda richiama la tua funzione. Un valore di `RequestResponse` indica una invocazione sincrona mentre un valore `Event` indica una invocazione asincrona.

Per richiamare la tua funzione su IPv6, usa gli endpoint dual-stack pubblici di Lambda. <https://docs.aws.amazon.com/general/latest/gr/rande.html#dual-stack-endpoints> Dual-stack Gli endpoint supportano sia IPv4 che IPv6. Gli endpoint dual-stack Lambda utilizzano la seguente sintassi:

```
protocol://lambda.us-east-1.api.aws
```

È possibile utilizzare gli [URL della funzione Lambda](#) anche per richiamare la funzione su IPv6. Gli endpoint URL della funzione hanno il formato seguente:

```
https://url-id.lambda-url.us-east-1.on.aws
```

Se l'invocazione della funzione genera un errore, per le invocazioni sincrone, visualizza il messaggio di errore nella risposta e riprova a richiamare manualmente. Per l'invocazione asincrona, Lambda gestisce i nuovi tentativi e può inviare i record di invocazione a una [destinazione](#).

## Richiamare una funzione Lambda in modo sincrono

Quando si invoca una funzione in modo sincrono, Lambda esegue la funzione e attende una risposta. Quando l'esecuzione di una funzione termina, Lambda restituisce la risposta dal codice della funzione con dati aggiuntivi, ad esempio la versione della funzione invocata. Per richiamare una funzione in modo sincrono con, usa il AWS CLI comando. `invoke`

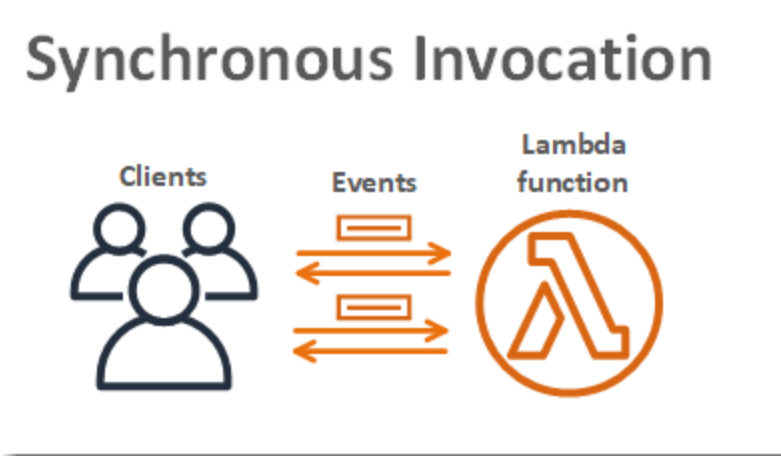
```
aws lambda invoke --function-name my-function \  
  --cli-binary-format raw-in-base64-out \  
  --payload '{ "key": "value" }' response.json
```

L'`cli-binary-format` opzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

Il seguente diagramma mostra i client che richiamano una funzione Lambda in modo sincrono. Lambda invia gli eventi direttamente alla funzione e invia la risposta della funzione all'invoker.



Il `payload` è una stringa che contiene un evento in formato JSON. Il nome del file in cui l' AWS CLI scrive la risposta dalla funzione è `response.json`. Se la funzione restituisce un oggetto o un errore,



il corpo della risposta è l'oggetto o l'errore in formato JSON. Se la funzione termina senza errori, il corpo della risposta è `null`.

#### Note

Lambda non attende il completamento delle estensioni esterne prima di inviare la risposta. Le estensioni esterne vengono eseguite come processi indipendenti nell'ambiente di esecuzione e continuano l'esecuzione dopo che la chiamata della funzione è stata completata. Per ulteriori informazioni, consulta [Aumentare le funzioni Lambda utilizzando le estensioni Lambda](#).

L'output del comando, che viene visualizzato nel terminale, include informazioni dalle intestazioni nella risposta da Lambda. Sono incluse la versione che ha elaborato l'evento (utile quando si utilizzano gli [alias](#)) e il codice di stato restituito da Lambda. Se Lambda è stato in grado di eseguire la funzione, il codice di stato è 200, anche se la funzione ha restituito un errore.

#### Note

Per le funzioni con un lungo timeout, il client potrebbe essere scollegato durante l'invocazione sincrona mentre è in attesa della risposta. Configurare il client HTTP, l'SDK, il firewall o il sistema operativo per consentire le connessioni lunghe con timeout o le impostazioni keep-alive.

Se Lambda non è in grado di eseguire la funzione, l'errore viene visualizzato nell'output.

```
aws lambda invoke --function-name my-function \  
  --cli-binary-format raw-in-base64-out \  
  --payload value response.json
```

Verrà visualizzato l'output seguente:

```
An error occurred (InvalidRequestContentException) when calling the Invoke operation:  
Could not parse request body into json: Unrecognized token 'value': was expecting  
( 'true', 'false' or 'null' )  
at [Source: (byte[]) "value"; line: 1, column: 11]
```

AWS CLI È uno strumento open source che puoi utilizzare per interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della [AWS CLI versione 2](#).

È possibile utilizzare [AWS CLI](#) per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "LogResult":
    "U1RBULQgUmVxdWVzdElkOiA4N2QwNDRIOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2l1vb...",
  "ExecutedVersion": "$LATEST"
}
```

Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'`base64` utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'`cli-binary-format` opzione è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed
Duration: 80 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

L'utilità `base64` è disponibile su Linux, macOS e [Ubuntu su Windows](#). Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

Per ulteriori informazioni sulle API `Invoke`, incluso un elenco completo di parametri, intestazioni ed errori, consulta [Invoke \(Invoca\)](#).

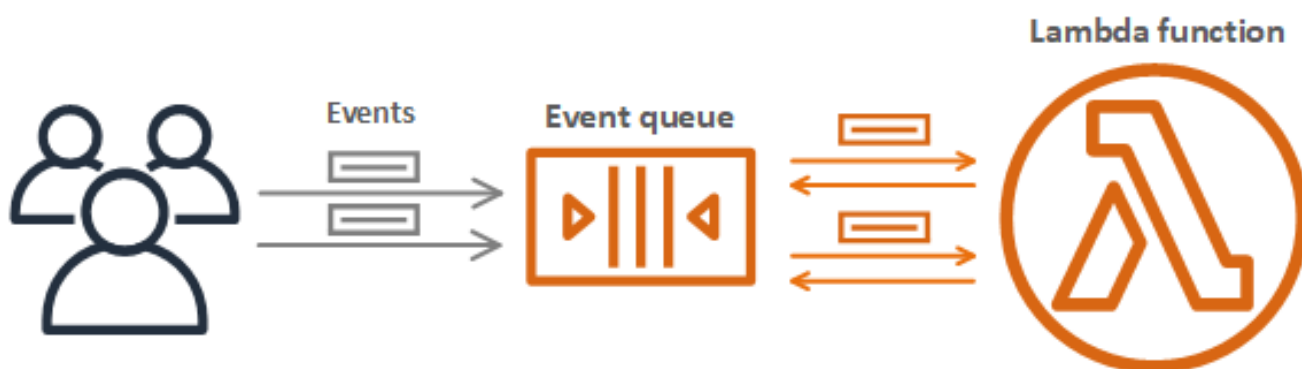
Quando si invoca una funzione direttamente, è possibile controllare la risposta agli errori e riprovare. L'SDK AWS CLI e l'AWS SDK riprovano inoltre automaticamente in caso di timeout, limitazione ed errori di servizio del client. Per ulteriori informazioni, consulta [Informazioni sul comportamento relativo ai nuovi tentativi in Lambda](#).

## Richiamo di una funzione Lambda in modo sincrono

Diversi Servizi AWS, come Amazon Simple Storage Service (Amazon S3) e Amazon Simple Notification Service (Amazon SNS), richiamano le funzioni in modo asincrono per elaborare gli eventi. Puoi anche richiamare una funzione Lambda in modo asincrono utilizzando () o uno degli SDK. AWS Command Line Interface AWS CLI AWS Quando si richiama una funzione in modo asincrono, non si attende una risposta dal codice della funzione. Si passa l'evento a Lambda e Lambda si occupa del resto. Puoi configurare il modo in cui Lambda gestisce gli errori e inviare i record di chiamata a una risorsa downstream come Amazon Simple Queue Service (Amazon SQS) o Amazon () per concatenare i componenti della tua applicazione. EventBridge EventBridge

Il seguente diagramma mostra i client che richiamano una funzione Lambda in modo asincrono. Lambda accoda gli eventi prima di inviarli alla funzione.

### Asynchronous Invocation



Per l'invocazione asincrona, Lambda inserisce l'evento in una coda e restituisce una risposta di esito positivo senza ulteriori informazioni. Un processo separato legge gli eventi dalla coda ed esegue la funzione.

Per richiamare una funzione Lambda in modo asincrono utilizzando AWS Command Line Interface (AWS CLI) o uno degli SDK, imposta il parametro su. AWS [InvocationType](#) Event L'esempio seguente mostra un comando per richiamare una funzione AWS CLI .

```
aws lambda invoke \
  --function-name my-function \
  --invocation-type Event \
  --cli-binary-format raw-in-base64-out \
```

```
--payload '{ "key": "value" }' response.json
```

Verrà visualizzato l'output seguente:

```
{  
  "StatusCode": 202  
}
```

L'opzione `cli-binary-format` è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Il file di output (`response.json`) non contiene informazioni, ma è comunque creato quando si esegue il comando. Se Lambda non è in grado di aggiungere l'evento alla coda, il messaggio di errore viene visualizzato nell'output del comando.

## In che modo Lambda gestisce gli errori e i nuovi tentativi con una invocazione asincrona

Lambda gestisce la coda di eventi asincroni della funzione ed effettua nuovi tentativi in caso di errori. Se la funzione restituisce un errore, per impostazione predefinita Lambda prova a eseguirla altre due volte, con un minuto di attesa tra i primi due tentativi e due minuti tra il secondo e il terzo. Gli errori di funzione includono gli errori restituiti dal codice della funzione e gli errori restituiti dal runtime della funzione, ad esempio `timeout`.

Se la funzione non dispone di sufficiente concorrenza per elaborare tutti gli eventi, ulteriori richieste saranno sottoposte a `throttling`. Per gli errori di limitazione (429) e gli errori di sistema (serie 500), Lambda restituisce l'evento alla coda e prova a eseguire nuovamente la funzione per un massimo di 6 ore. L'intervallo tra i tentativi aumenta esponenzialmente da 1 secondo dopo il primo tentativo a un massimo di 5 minuti. Se la coda contiene molte voci, Lambda aumenta l'intervallo dei tentativi e riduce la velocità con cui legge gli eventi dalla coda.

Anche se la funzione non restituisce un errore, è possibile che riceva lo stesso evento da Lambda più volte perché la coda stessa alla fine è coerente. Se la funzione non è in grado di seguire gli eventi in entrata, gli eventi possono anche essere eliminati dalla coda senza essere inviati alla funzione. Verificare che il codice della funzione gestisca normalmente gli eventi duplicati e che si disponga di sufficiente concorrenza per gestire tutte le invocazioni.

Quando la coda è molto lunga, i nuovi eventi potrebbero diventare datati prima che Lambda abbia la possibilità di inviarli alla funzione. Quando un evento scade o fallisce tutti i tentativi di elaborazione, Lambda lo scarta. È possibile [configurare la gestione degli errori](#) per una funzione per ridurre il numero di tentativi eseguiti da Lambda o per eliminare più rapidamente gli eventi non elaborati. Configurare una coda DLQ per conservare gli eventi scartati. Per registrare le registrazioni delle chiamate non riuscite (ad esempio timeout o errori di runtime), [create](#) una destinazione in caso di errore.

## Configurazione delle impostazioni di gestione degli errori per le invocazioni asincrone Lambda

Usa le seguenti impostazioni per configurare il modo in cui Lambda gestisce gli errori e i nuovi tentativi per le invocazioni di funzioni asincrone:

- [MaximumEventAgeInSeconds](#): il tempo massimo in secondi per cui Lambda conserva un evento nella coda degli eventi asincroni prima di eliminarlo.
- [MaximumRetryAttempts](#): il numero massimo di tentativi che Lambda effettua quando la funzione restituisce un errore.

Utilizza la console Lambda o AWS CLI per configurare le impostazioni di gestione degli errori per una funzione, una versione o un alias.

### Console

Per configurare la gestione degli errori

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione), quindi scegli Asynchronous invocation (Chiamata asincrona).
4. In Asynchronous invocation (Chiamata asincrona), scegliere Edit (Modifica).
5. Configura le impostazioni seguenti.
  - Maximum age of event (Età massima dell'evento): il tempo massimo per cui Lambda conserva un evento nella coda degli eventi asincroni, fino a 6 ore.
  - Retry attempts (Nuovi tentativi): il numero di tentativi che Lambda effettua quando la funzione restituisce un errore, tra 0 e 2.

## 6. Selezionare Salva.

### AWS CLI

Per configurare l'invocazione asincrona con la AWS CLI, utilizza il comando [put-function-event-invoke-config](#). Nell'esempio seguente viene configurata una funzione con una durata massima dell'evento di 1 ora e senza ulteriori tentativi.

```
aws lambda put-function-event-invoke-config \  
  --function-name error \  
  --maximum-event-age-in-seconds 3600 \  
  --maximum-retry-attempts 0
```

Il comando `put-function-event-invoke-config` sovrascrive qualsiasi configurazione esistente sulla funzione, versione o alias. Per configurare un'opzione senza reimpostarne altre, utilizza [update-function-event-invoke-config](#). Nell'esempio seguente Lambda viene configurato per inviare un record a una coda SQS standard denominata `destination` quando non è possibile elaborare un evento.

```
aws lambda update-function-event-invoke-config \  
  --function-name my-function \  
  --destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-  
east-1:123456789012:destination"}}'
```

Verrà visualizzato l'output seguente:

```
{  
  "LastModified": 1573686021.479,  
  "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:  
$LATEST",  
  "MaximumRetryAttempts": 0,  
  "MaximumEventAgeInSeconds": 3600,  
  "DestinationConfig": {  
    "OnSuccess": {},  
    "OnFailure": {}  
  }  
}
```

Quando un evento di invocazione supera l'età massima o fallisce tutti i nuovi tentativi, Lambda lo scarta. Per conservare una copia degli eventi eliminati, configurare una [destinazione](#) dell'evento non riuscito.

## Acquisizione dei record delle invocazioni asincrone Lambda

Lambda può inviare record di chiamate asincrone a uno dei seguenti. Servizi AWS

- Amazon SQS: una coda SQS standard
- Amazon SNS: un argomento SNS standard
- Amazon S3: un bucket Amazon S3 (solo in caso di errore)
- AWS Lambda: una funzione Lambda
- EventBridgeAmazon EventBridge : un bus per eventi

Il record di invocazione contiene dettagli sulla richiesta e la risposta in formato JSON. È possibile configurare destinazioni separate per gli eventi che vengono elaborati correttamente e per quelli che restituiscono un errore a ogni tentativo di elaborazione. In alternativa, è possibile configurare una coda standard di Amazon SQS o un argomento standard di Amazon SNS come coda DLQ per gli eventi scartati. Per le code DLQ, Lambda invia solo il contenuto dell'evento, senza dettagli sulla risposta.

Se Lambda non è in grado di inviare un record a una destinazione che hai configurato, invia una `DestinationDeliveryFailures` metrica ad Amazon CloudWatch. Ciò può verificarsi se la configurazione include un tipo di destinazione non supportato, ad esempio una coda FIFO di Amazon SQS o un argomento FIFO di Amazon SNS. Gli errori di recapito possono verificarsi anche a causa di errori di autorizzazioni e limiti di dimensione. Per ulteriori informazioni sui parametri di invocazione Lambda, consulta [the section called “Parametri di invocazione”](#).

### Note

Per impedire l'attivazione di una funzione, è possibile impostare la simultaneità riservata della funzione su zero. Quando si imposta la simultaneità riservata su zero per una funzione chiamata in modo asincrono, Lambda inizia a inviare i nuovi eventi alla [coda DLQ](#) configurata o alla [destinazione degli eventi](#) in caso di errore, senza nuovi tentativi. Per elaborare gli eventi inviati mentre la simultaneità riservata era impostata su zero, è necessario utilizzare gli eventi dalla coda DLQ o dalla destinazione degli eventi in caso di errore.



## Aggiunta di una destinazione

Per mantenere i record delle chiamate asincrone, aggiungi una destinazione alla funzione. È possibile scegliere di inviare a una destinazione le chiamate riuscite o non riuscite. Ogni funzione può avere più destinazioni, quindi è possibile configurare destinazioni separate per eventi riusciti e non riusciti. Ogni record inviato alla destinazione è un documento JSON con i dettagli relativi alla chiamata. Come per le impostazioni di gestione degli errori, è possibile impostare le destinazioni su una funzione, una versione della funzione o un alias.

### Tip

Puoi anche conservare i record delle chiamate non riuscite per i seguenti tipi di mappatura delle sorgenti di eventi: [Amazon Kinesis](#), Amazon DynamoDB e Apache Kafka ([Amazon MSK e Apache Kafka autogestito](#)).

La tabella seguente elenca le destinazioni supportate per i record di chiamata asincrona. Affinché Lambda invii correttamente i record alla destinazione prescelta, assicurati che il [ruolo di esecuzione](#) della funzione disponga anche delle autorizzazioni pertinenti. La tabella descrive anche il modo in cui ogni tipo di destinazione riceve il record di chiamata JSON.

| Tipo di destinazione                      | Autorizzazione richiesta                                        | Destination-specific Formato JSON                                                                                                                                                              |
|-------------------------------------------|-----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Coda Amazon SQS                           | <a href="#">sqs: SendMessage</a>                                | Lambda passa il record di chiamata come Message alla destinazione.                                                                                                                             |
| Argomento Amazon SNS                      | <a href="#">sns: Publish</a>                                    | Lambda passa il record di chiamata come Message alla destinazione.                                                                                                                             |
| Bucket Amazon S3 (solo in caso di errore) | <a href="#">s3: PutObject</a><br><a href="#">s3: ListBucket</a> | <ul style="list-style-type: none"> <li>Lambda archivia il record di invocazione come oggetto JSON nel bucket di destinazione.</li> <li>Il nome dell'oggetto S3 utilizza la seguente</li> </ul> |

| Tipo di destinazione | Autorizzazione richiesta               | Destination-specific Formato JSON                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------|----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                      |                                        | <p>convenzione di denominazione:</p> <pre>aws/lambda/async/&lt;function-name&gt;/YYY Y/MM/DD/YYYY-MM-DD THH.MM.SS-&lt;Random UUID&gt;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| funzione Lambda      | <a href="#">lambda: InvokeFunction</a> | Lambda passa il record di chiamata come payload alla funzione.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| EventBridge          | <a href="#">eventi: PutEvents</a>      | <ul style="list-style-type: none"> <li>Lambda passa il record di chiamata come <code>detail</code> nella chiamata. <code>PutEvents</code></li> <li>Il valore per il campo di <code>event source</code> è <code>lambda</code>.</li> <li>Il valore per il campo dell'evento <code>detail-type</code> è "Risultato della chiamata della funzione Lambda - Successo" o "Risultato della chiamata della funzione Lambda - Errore".</li> <li>Il campo dell'evento <code>resource</code> contiene i nomi della risorsa Amazon (ARN) della funzione e della destinazione.</li> <li>Per altri campi relativi agli eventi, consulta <a href="#">Amazon events. EventBridge</a></li> </ul> |

 Note

Per le destinazioni Amazon S3, se hai abilitato la crittografia sul bucket utilizzando una chiave KMS, la tua funzione richiede anche l'autorizzazione [kms: GenerateDataKey](#)

 Important

Quando usi Amazon SNS come destinazione, tieni presente che Amazon SNS ha un limite massimo di dimensione dei messaggi di 256 KB. Se il payload di chiamata asincrona si avvicina a 1 MB, il record di chiamata (che include il payload originale più metadati aggiuntivi) potrebbe superare il limite di Amazon SNS e causare errori di consegna. Prendi in considerazione l'utilizzo di destinazioni Amazon SQS o Amazon S3 per payload più grandi.

La procedura seguente descrive come configurare una destinazione per una funzione che utilizza la console Lambda e la AWS CLI.

## Console

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. In Function overview (Panoramica delle funzioni), scegliere Add destination (Aggiungi destinazione).
4. Per Source (Origine), scegliere Asynchronous invocation (Chiamata asincrona).
5. Per Condition (Condizione) scegliere tra le seguenti opzioni:
  - In caso di errore: viene inviato un record quando l'evento fallisce tutti i tentativi di elaborazione o supera l'età massima.
  - On success (In caso di esito positivo): viene inviato un record quando la funzione elabora correttamente un'invocazione asincrona.
6. Per Destination type (Tipo di destinazione), scegliere il tipo di risorsa che riceve il record di invocazione.
7. Per Destination (Destinazione), scegliere una risorsa.
8. Scegli Save (Salva).

## AWS CLI

Per configurare una destinazione utilizzando il comando AWS CLI `update-function-event-invoke-config`. Nell'esempio seguente Lambda viene configurato per inviare un record a una coda SQS standard denominata `destination` quando non è possibile elaborare un evento.

```
aws lambda update-function-event-invoke-config \
  --function-name my-function \
  --destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-east-1:123456789012:destination"}}'
```

### Best practice di sicurezza per destinazioni Amazon S3

L'eliminazione di un bucket S3 configurato come destinazione senza rimuovere la destinazione dalla configurazione della funzione può creare un rischio per la sicurezza. Se un altro utente conosce il nome del bucket di destinazione, può ricreare il bucket nel proprio Account AWS. I record delle invocazioni non riuscite verranno inviati al relativo bucket, esponendo potenzialmente i dati della tua funzione.

#### Warning

Per assicurarti che i record di chiamata dalla tua funzione non possano essere inviati a un bucket S3 in un altro Account AWS, aggiungi una condizione al ruolo di esecuzione della funzione che limiti le autorizzazioni ai bucket del tuo account. `s3:PutObject`

Di seguito viene illustrato un esempio di policy IAM che limita le autorizzazioni `s3:PutObject` della funzione ai bucket presenti nell'account. Questa policy fornisce inoltre a Lambda l'autorizzazione `s3:ListBucket` necessaria per utilizzare un bucket S3 come destinazione.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "S3BucketResourceAccountWrite",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:ListBucket"
      ]
    }
  ]
}
```

```
    ],
    "Resource": [
      "arn:aws:s3:::*/*",
      "arn:aws:s3:::*"
    ],
    "Condition": {
      "StringEquals": {
        "s3:ResourceAccount": "111122223333"
      }
    }
  }
}
```

Per aggiungere una politica di autorizzazioni al ruolo di esecuzione della funzione utilizzando Console di gestione AWS or AWS CLI, consulta le istruzioni nelle seguenti procedure:

### Console

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Seleziona la funzione Lambda di cui si desidera modificare il ruolo di esecuzione.
3. Nella scheda Configurazione scegli Autorizzazioni.
4. Nella scheda Ruolo di esecuzione, seleziona il nome del ruolo della funzione per aprire la pagina della console IAM del ruolo.
5. Aggiungi una policy di autorizzazioni di base al ruolo completando le seguenti operazioni:
  - a. Nel riquadro Policy di autorizzazioni, scegli Aggiungi autorizzazioni, poi Crea policy in linea.
  - b. Nell'editor delle policy, seleziona JSON.
  - c. Incolla la policy che desideri aggiungere nell'editor (sostituendo il codice JSON esistente), quindi scegli Avanti.
  - d. In Dettagli della policy, specifica un nome per la policy.
  - e. Scegli Crea policy.

## AWS CLI

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (CLI)

1. Crea un documento di policy JSON con le autorizzazioni richieste e salvalo in una directory locale.
2. Utilizza il comando della CLI `put-role-policy` di IAM per aggiungere le autorizzazioni per il ruolo di esecuzione di una funzione. Esegui il comando seguente dalla directory in cui hai salvato il documento di policy JSON e sostituisci il nome del ruolo, il nome della policy e il documento di policy con i tuoi valori.

```
aws iam put-role-policy \  
--role-name my_lambda_role \  
--policy-name LambdaS3DestinationPolicy \  
--policy-document file://my_policy.json
```

### Record di invocazione di esempio

Quando un'invocazione corrisponde alla condizione, Lambda invia un [documento JSON](#) con i dettagli sull'invocazione alla destinazione. L'esempio seguente mostra un record di invocazione per un evento che non è stato possibile elaborare per tre volte a causa di un errore di funzione.

### Example

```
{  
  "version": "1.0",  
  "timestamp": "2019-11-14T18:16:05.568Z",  
  "requestContext": {  
    "requestId": "e4b46cbf-b738-xmpl-8880-a18cdf61200e",  
    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:  
$LATEST",  
    "condition": "RetriesExhausted",  
    "approximateInvokeCount": 3  
  },  
  "requestPayload": {  
    "ORDER_IDS": [  
      "9e07af03-ce31-4ff3-xmpl-36dce652cb4f",  
      "637de236-e7b2-464e-xmpl-baf57f86bb53",  
      "a81ddca6-2c35-45c7-xmpl-c3a03a31ed15"  
    ]  
  },  
}
```

```
"responseContext": {
  "statusCode": 200,
  "executedVersion": "$LATEST",
  "functionError": "Unhandled"
},
"responsePayload": {
  "errorMessage": "RequestId: e4b46cbf-b738-xmpl-8880-a18cdf61200e Process exited
before completing request"
}
}
```

Il record di invocazione contiene dettagli sull'evento, la risposta e il motivo per cui il record è stato inviato.

### Tracciamento delle richieste verso le destinazioni

È possibile utilizzare AWS X-Ray per visualizzare una vista connessa di ogni richiesta mentre viene messa in coda, elaborata da una funzione Lambda e inviata al servizio di destinazione. Quando attivi la X-Ray traccia per una funzione o un servizio che richiama una funzione, Lambda aggiunge un'intestazione alla richiesta e passa l' X-Ray intestazione al servizio di destinazione. Le tracce dei servizi upstream vengono automaticamente collegate alle tracce delle funzioni Lambda downstream e dei servizi di destinazione, creando una visione end-to-end dell'intera applicazione. Per ulteriori informazioni sul tracciamento, consulta [Visualizza le invocazioni delle funzioni Lambda utilizzando AWS X-Ray](#).

### Aggiunta di una coda DLQ

In alternativa a una [destinazione in caso di errore](#), è possibile configurare la funzione con una coda DLQ per salvare gli eventi eliminati per ulteriori elaborazioni. Una coda DLQ agisce allo stesso modo di una destinazione in caso di errore in quanto viene utilizzata quando un evento non riesce a tutti i tentativi di elaborazione o scade senza essere elaborato. Tuttavia, è possibile aggiungere o rimuovere una coda DLQ solo a livello di funzione. Le versioni delle funzioni utilizzano le stesse impostazioni relative alla coda di lettere morte della versione non pubblicata (\$LATEST). On-failure le destinazioni supportano anche destinazioni aggiuntive e includono dettagli sulla risposta della funzione nel record di chiamata.

Per rielaborare gli eventi in una coda DLQ, è possibile impostarla come [origine eventi](#) per la funzione Lambda. In alternativa, è possibile recuperare gli eventi manualmente.

È possibile scegliere una coda standard di Amazon SQS o un argomento standard di Amazon SNS per la coda DLQ. Le code FIFO e gli argomenti FIFO di Amazon SNS non sono supportati.

- [Coda Amazon SQS](#): una coda conserva gli eventi non riusciti finché non vengono richiamati. Scegli una coda standard di Amazon SQS se prevedi che una singola entità, ad esempio una funzione Lambda o un CloudWatch allarme, elabori l'evento fallito. Per ulteriori informazioni, consulta [Utilizzo di Lambda con Amazon SQS](#).
- [Argomento Amazon SNS](#) – Un argomento invia gli eventi non riusciti a una o più destinazioni. Scegli un argomento standard di Amazon SNS se ti aspetti che più entità agiscano su un evento non riuscito. Ad esempio, puoi configurare un argomento per inviare eventi a un indirizzo email, una funzione Lambda, un endpoint HTTP. and/or Per ulteriori informazioni, consulta [Richiamo di funzioni Lambda mediante notifiche Amazon SNS](#).

Per inviare gli eventi a una coda o argomento, la funzione necessita di autorizzazioni aggiuntive. Aggiungi una policy con le [autorizzazioni necessarie](#) per il [ruolo di esecuzione](#) di una funzione. Se la coda o l'argomento di destinazione è crittografato con una AWS KMS chiave gestita dal cliente, assicurati che sia il ruolo di esecuzione della funzione che la policy basata sulle [risorse della chiave contengano le autorizzazioni pertinenti](#).

Dopo aver creato il target e l'aggiornamento del ruolo di esecuzione della funzione, aggiungere la coda DLQ alla funzione. È possibile configurare più funzioni per l'invio di eventi allo stesso oggetto.

## Console

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione), quindi scegli Asynchronous invocation (Chiamata asincrona).
4. In Asynchronous invocation (Chiamata asincrona), scegliere Edit (Modifica).
5. Imposta il servizio di Dead-letter coda su Amazon SQS o Amazon SNS.
6. Scegliere l'argomento o la coda target.
7. Scegli Save (Salva).

## AWS CLI

Per configurare una coda di lettere morte con, usa il comando update-function-configuration. AWS CLI <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/update-function-configuration.html>



```
aws lambda update-function-configuration \  
  --function-name my-function \  
  --dead-letter-config TargetArn=arn:aws:sns:us-east-1:123456789012:my-topic
```

Lambda invia l'evento alla coda DLQ così com'è, con ulteriori informazioni negli attributi. Queste informazioni possono essere utilizzate per identificare l'errore restituito dalla funzione o correlare l'evento ai log o a una traccia AWS X-Ray .

Dead-letter attributi dei messaggi di coda

- RequestID (String) – L'ID della richiesta di invocazione. Gli ID della richiesta appaiono nei log della funzione. Puoi anche utilizzare l' X-Ray SDK per registrare l'ID della richiesta su un attributo nella traccia. È quindi possibile cercare le tracce in base all'ID della richiesta nella X-Ray console.
- ErrorCode(Numero): il codice di stato HTTP.
- ErrorMessage(String) — Il primo 1 KB del messaggio di errore.

Se Lambda non è in grado di inviare un messaggio alla coda delle lettere morte, elimina l'evento ed emette la metrica. [DeadLetterErrors](#). Questo può accadere a causa di mancanza di autorizzazioni oppure se le dimensioni totali del messaggio superano il limite per la coda o l'argomento target. Ad esempio, supponiamo che una notifica Amazon SNS con un corpo di dimensioni vicine a 1 MB attivi una funzione che genera un errore. In tal caso, i dati relativi a eventi aggiunti da Amazon SNS, combinati con gli attributi aggiunti da Lambda, possono far sì che il messaggio superi le dimensioni massime consentite nella coda DLQ.

Se si utilizza Amazon SQS come origine eventi, configurare una coda DLQ sulla coda Amazon SQS stessa e non sulla funzione Lambda. Per ulteriori informazioni, consulta [Utilizzo di Lambda con Amazon SQS](#).

## In che modo Lambda elabora i record provenienti da origini eventi basate su flussi e code

Uno strumento di mappatura dell'origine degli eventi è una risorsa Lambda che legge gli elementi da servizi basati su flussi o code e richiama una funzione con batch di record. All'interno di uno strumento di mappatura dell'origine degli eventi, risorse chiamate poller di eventi cercano attivamente nuovi messaggi e richiamano funzioni. Per impostazione predefinita, Lambda scala automaticamente i poller di eventi, ma per determinati tipi di origini eventi, puoi utilizzare la [modalità provisioning](#) per controllare il numero minimo e massimo di poller di eventi dedicati allo strumento di mappatura dell'origine degli eventi.

I seguenti servizi utilizzano gli strumenti di mappatura dell'origine degli eventi per richiamare le funzioni Lambda:

- [Amazon DocumentDB \(compatibile con MongoDB\) \(Amazon DocumentDB\)](#)
- [Amazon DynamoDB](#)
- [Amazon Kinesis](#)
- [Amazon MQ](#)
- [Amazon Managed Streaming for Apache Kafka \(Amazon MSK\)](#)
- [Self-managed Apache Kafka](#)
- [Amazon Simple Queue Service \(Amazon SQS\)](#)

### Warning

Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per saperne di più, vedi [Come posso rendere idempotente](#) la mia funzione Lambda nel Knowledge Center. AWS

## In che modo gli strumenti di mappatura dell'origine degli eventi differiscono dai trigger diretti

Alcuni Servizi AWS possono richiamare direttamente le funzioni Lambda utilizzando i trigger. Questi servizi inviano eventi a Lambda e la funzione viene richiamata immediatamente quando si

verifica l'evento specificato. I trigger sono adatti per eventi discreti ed elaborazione in tempo reale. Quando si [crea un trigger utilizzando la console Lambda](#), la console interagisce con il AWS servizio corrispondente per configurare la notifica degli eventi su quel servizio. Il trigger viene effettivamente archiviato e gestito dal servizio che genera gli eventi, non da Lambda. Ecco alcuni esempi di servizi che utilizzano i trigger per richiamare le funzioni Lambda:

- Amazon Simple Storage Service (Amazon S3): richiama una funzione quando un oggetto viene creato, eliminato o modificato in un bucket. Per ulteriori informazioni, consulta [Tutorial: uso di un trigger Amazon S3 per richiamare una funzione Lambda](#).
- Amazon Simple Notification Service (Amazon SNS): richiama una funzione quando un messaggio è pubblicato in un argomento SNS. Per ulteriori informazioni, consulta [Tutorial: Utilizzo AWS Lambda con Amazon Simple Notification Service](#).
- Gateway Amazon API: richiama una funzione quando viene effettuata una richiesta API a un endpoint specifico. Per ulteriori informazioni, consulta [Richiamo di funzione Lambda utilizzando un endpoint Gateway Amazon API](#).

Gli strumenti di mappatura dell'origine degli eventi sono risorse Lambda create e gestite all'interno del servizio Lambda. Gli strumenti di mappatura dell'origine degli eventi sono progettate per l'elaborazione di dati o messaggi in streaming ad alto volume dalle code. L'elaborazione dei record da un flusso o da una coda in batch è più efficiente rispetto all'elaborazione dei record singolarmente.

## Comportamento di batching

Per impostazione predefinita, una mappatura delle origini eventi raggruppa i registri in un unico payload che Lambda invia alla funzione. Per ottimizzare il comportamento del batch, è possibile configurare una finestra di batch ([MaximumBatchingWindowInSeconds](#)) e una dimensione del batch ([BatchSize](#)). Una finestra di batch è il tempo massimo per la raccolta dei registri in un singolo payload. La dimensione del batch è il numero massimo di registri in un singolo batch. Lambda richiama la funzione in presenza dei tre criteri seguenti:

- La finestra di dosaggio raggiunge il valore massimo. Il comportamento predefinito della finestra di batch varia in base alla specifica origine eventi.
- Per le origini eventi Kinesis, DynamoDB e Amazon SQS: la finestra di batch di default è 0 secondi. Ciò significa che Lambda richiama la funzione non appena i record sono disponibili. Per impostare una finestra di batch, configura `MaximumBatchingWindowInSeconds`. È possibile impostare questo parametro su qualsiasi valore da compreso tra 0 e 300 secondi con incrementi

di 1 secondo. Se si configura una la finestra di batch, la finestra successiva inizia non appena viene completato il precedente richiamo della funzione.

- Per le origini degli eventi di Amazon MSK, Apache Kafka autogestito, Amazon MQ e Amazon DocumentDB la finestra di batch predefinita è 500 ms. È possibile configurare `MaximumBatchingWindowInSeconds` su qualsiasi valore da 0 secondi a 300 secondi con incrementi di secondi. In modalità provvisoria per la mappatura delle sorgenti degli eventi di Kafka, quando si configura una finestra di batch, la finestra successiva inizia non appena il batch precedente viene completato. Nelle mappature delle sorgenti di eventi Kafka senza provisioning, quando si configura una finestra di batch, la finestra successiva inizia non appena viene completata la precedente invocazione della funzione.

Per ridurre al minimo la latenza quando si utilizzano le mappature delle sorgenti di eventi Kafka in modalità provisioning, impostate su 0. `MaximumBatchingWindowInSeconds` Questa impostazione garantisce che Lambda inizi a elaborare il batch successivo subito dopo aver completato l'invocazione della funzione corrente. Per ulteriori informazioni sull'elaborazione a bassa latenza, vedere. [Apache Kafka a bassa latenza](#)

- Per le sorgenti di eventi Amazon MQ e Amazon DocumentDB: la finestra di batch predefinita è di 500 ms. È possibile configurare `MaximumBatchingWindowInSeconds` su qualsiasi valore da 0 secondi a 300 secondi con incrementi di secondi. Una finestra di batch inizia non appena arriva il primo registro.

#### Note

Perché è possibile modificare `MaximumBatchingWindowInSeconds` solo in incrementi di secondi, non puoi tornare alla finestra di batch predefinita di 500 ms dopo averla modificata. Per ripristinare la finestra di batch predefinita, è necessario creare una nuova mappatura dell'origine eventi.

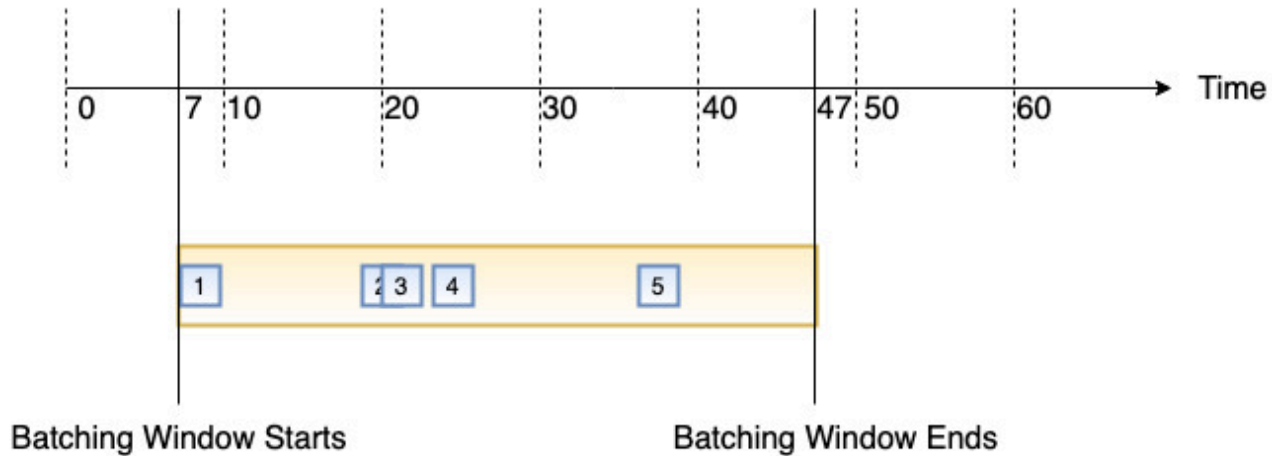
- Le dimensioni del batch sono soddisfatte. La dimensione minima del batch è 1. La dimensione predefinita e massima del batch dipendono dall'origine eventi. Per ulteriori informazioni su questi valori, consulta le specifiche di [BatchSize](#) per l'operazione API `CreateEventSourceMapping`.
- La dimensione del payload raggiunge [6 MB](#). Tale limite non è modificabile.

Il diagramma seguente illustra queste tre configurazioni. Supponiamo che una finestra di batch inizi a  $t = 7$  secondi. Nel primo scenario, la finestra di batch raggiunge il suo massimo di 40 secondi a  $t = 47$  secondi dopo aver accumulato 5 registri. Nel secondo scenario, la dimensione del batch

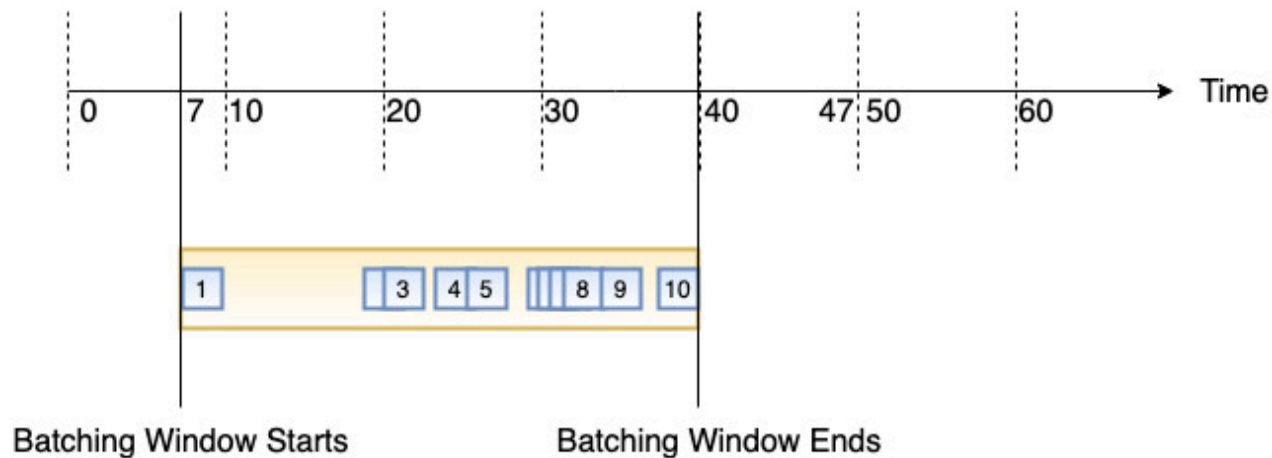
raggiunge 10 prima della scadenza della finestra di batch, quindi la finestra di batch termina in anticipo. Nel secondo scenario, il valore massimo del payload viene raggiunta prima della scadenza della finestra di batch, quindi la finestra di batch termina in anticipo.

Max Batching Window = 40 Seconds  
Max Batch Size = 10  
Max Batch Size in Bytes = 6 MB

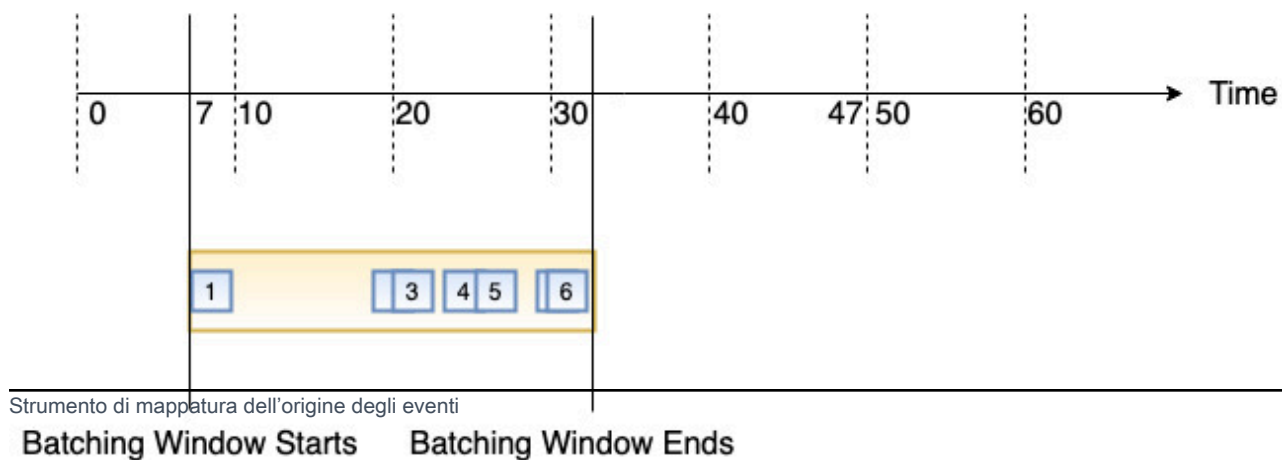
(1) Batching Window Expires



(2) Batching Size is reached



(3) Batch Size in bytes is reached



Consigliamo di eseguire il test con dimensioni di batch e record diverse in modo che la frequenza di polling di ogni origine eventi sia regolata in base alla velocità con cui la funzione è in grado di completare l'attività. Il [CreateEventSourceMapping](#) BatchSize parametro controlla il numero massimo di record che possono essere inviati alla funzione con ogni richiamo. Una dimensione di batch maggiore può spesso assorbire in modo più efficiente il costo associato all'invocazione in un set di record più grande, aumentando in tal modo il throughput.

Lambda non attende il completamento di alcuna [estensione](#) prima di inviare il batch successivo per l'elaborazione. In altre parole, le estensioni possono continuare a funzionare mentre Lambda elabora il successivo batch di record. Ciò può causare problemi di limitazione in caso di violazione delle impostazioni o dei limiti di [simultaneità](#). Per rilevare se si tratta di un potenziale problema, monitora le tue funzioni e verifica se i [parametri di simultaneità](#) per lo strumento di mappatura dell'origine degli eventi sono superiori al previsto. A causa degli intervalli ridotti tra le invocazioni, Lambda potrebbe segnalare brevemente un utilizzo della simultaneità maggiore rispetto al numero di partizioni. Ciò può essere vero anche per le funzioni Lambda senza estensioni.

Per impostazione predefinita, se la funzione restituisce un errore, l'intera mappatura del batch viene rielaborato fino a quando la funzione non restituisce esito positivo o gli elementi nel batch scadono. Per garantire l'ordine di elaborazione, l'elaborazione della mappatura delle fonti eventi per gli shard interessati viene messa in pausa finché l'errore non viene risolto. Per le origini di flusso (DynamoDB e Kinesis), è possibile configurare il numero massimo di tentativi che Lambda effettua quando la funzione restituisce un errore. Gli errori di servizio o le limitazioni per cui il batch non raggiunge la funzione non vengono conteggiati ai fini dei tentativi ripetuti. È inoltre possibile configurare lo strumento di mappatura dell'origine degli eventi per inviare un record di invocazione a una [destinazione](#) quando elimina un batch di eventi.

## Modalità provisioning

Gli strumenti di mappatura dell'origine degli eventi Lambda utilizzano i poller di eventi per interrogare l'origine eventi alla ricerca di nuovi messaggi. Per impostazione predefinita, Lambda gestisce la scalabilità automatica di questi poller in base al volume dei messaggi. Quando il traffico di messaggi aumenta, Lambda aumenta automaticamente il numero di poller di eventi per gestire il carico e li riduce quando il traffico diminuisce.

In modalità provisioning, puoi ottimizzare il throughput della mappatura delle sorgenti degli eventi definendo limiti minimi e massimi per le risorse di polling dedicate che rimangono pronte a gestire i modelli di traffico previsti. Queste risorse si scalano automaticamente 3 volte più velocemente per gestire picchi improvvisi nel traffico degli eventi e offrono una capacità 16 volte superiore per

elaborare milioni di eventi. Questo ti aiuta a creare carichi di lavoro basati sugli eventi altamente reattivi con requisiti prestazionali rigorosi.

In Lambda, un event poller è un'unità di calcolo con capacità di throughput che variano in base al tipo di origine dell'evento. Per Amazon MSK e Apache Kafka autogestito, ogni poller di eventi può gestire fino al 5% del throughput o fino a 5 chiamate simultanee. MB/sec Ad esempio, se la sorgente dell'evento produce un payload medio di 1 MB e la durata media della funzione è di 1 secondo, un singolo poller di eventi Kafka può supportare 5 throughput e 5 invocazioni Lambda simultanee (supponendo che non vi sia alcuna trasformazione del payload). MB/sec Per Amazon SQS, ogni poller di eventi può gestire fino all'1% del throughput o fino a 10 chiamate simultanee. MB/sec L'utilizzo della modalità provisioned comporta costi aggiuntivi in base all'utilizzo del poller di eventi. Per i dettagli sui prezzi, vedere [Prezzi di AWS Lambda](#).

La modalità provisioned è disponibile per Amazon MSK, Apache Kafka autogestito e sorgenti di eventi Amazon SQS. Mentre le impostazioni di simultaneità consentono di controllare la scalabilità della funzione, la modalità provisioning consente di controllare il throughput dello strumento di mappatura dell'origine degli eventi. Per garantire le massime prestazioni, potrebbe essere necessario regolare entrambe le impostazioni in modo indipendente.

La modalità provisioned è ideale per applicazioni in tempo reale che richiedono una latenza costante nell'elaborazione degli eventi, come le società di servizi finanziari che elaborano i feed di dati di mercato, le piattaforme di e-commerce che forniscono consigli personalizzati in tempo reale e le società di giochi che gestiscono le interazioni con i giocatori dal vivo.

Ogni event poller supporta diverse capacità di throughput:

- Per Amazon MSK e Apache Kafka autogestito: fino al 5% MB/sec del throughput o fino a 5 richiami simultanei
- Per Amazon SQS: fino all'1% MB/sec del throughput o fino a 10 richiami simultanei o fino a 10 chiamate API di polling SQS al secondo.

Per le mappature delle sorgenti degli eventi di Amazon SQS, puoi impostare il numero minimo di poller tra 2 e 200 con un valore predefinito di 2 e il numero massimo tra 2 e 10.000 con un valore predefinito di 200. Lambda scala il numero di poller di eventi tra il minimo e il massimo configurati, sommando rapidamente fino a 1.000 simultanee al minuto per fornire un'elaborazione coerente e a bassa latenza degli eventi.

Per le mappature delle sorgenti degli eventi di Kafka, puoi impostare il numero minimo di poller tra 1 e 200 con un valore predefinito di 1 e il numero massimo tra 1 e 2.000 con un valore predefinito di 200.



Lambda scala il numero di poller di eventi tra il minimo e il massimo configurati in base al backlog di eventi nell'argomento per fornire un'elaborazione a bassa latenza degli eventi.

Tieni presente che per le sorgenti di eventi di Amazon SQS, l'impostazione di concorrenza massima non può essere utilizzata con la modalità provisioned. Quando si utilizza la modalità provisioned, è possibile controllare la concorrenza tramite l'impostazione del numero massimo di poller di eventi.

Per informazioni dettagliate sulla configurazione della modalità provisioning, consulta le sezioni seguenti:

- [Configurazione della modalità provisioning per gli strumenti di mappatura dell'origine degli eventi Amazon MSK](#)
- [Configurazione della modalità provisioning per lo strumento di mappatura dell'origine degli eventi di Apache Kafka autogestito](#)
- [Utilizzo della modalità provisioned con le mappature delle sorgenti di eventi di Amazon SQS](#)

Per ridurre al minimo la latenza in modalità provisioning, impostate su 0.

`MaximumBatchingWindowInSeconds` Questa impostazione garantisce che Lambda inizi a elaborare il batch successivo subito dopo aver completato l'invocazione della funzione corrente. Per ulteriori informazioni sull'elaborazione a bassa latenza, vedere. [Apache Kafka a bassa latenza](#)

Dopo aver configurato la modalità provisioning, puoi osservare l'utilizzo dei poller di eventi per il tuo carico di lavoro monitorando il parametro `ProvisionedPollers`. Per ulteriori informazioni, consulta [the section called "Parametri dello strumento di mappatura dell'origine degli eventi"](#).

## API della mappatura dell'origine eventi

Per gestire un'origine eventi con la [AWS Command Line Interface \(AWS CLI\)](#) o un [SDK AWS](#), è possibile utilizzare le seguenti operazioni API:

- [CreateEventSourceMapping](#)
- [ListEventSourceMappings](#)
- [GetEventSourceMapping](#)
- [UpdateEventSourceMapping](#)
- [DeleteEventSourceMapping](#)

## Utilizzo di tag negli strumenti di mappatura dell'origine degli eventi

Puoi taggare gli strumenti di mappatura dell'origine degli eventi per organizzare e gestire le risorse. I tag sono coppie chiave-valore a forma libera associate alle risorse supportate su Servizi AWS. Per ulteriori informazioni sui casi d'uso dei tag, consulta Strategie di tagging [comuni](#) nella Guida alle AWS risorse sui tag e all'editor di tag.

Gli strumenti di mappatura dell'origine degli eventi sono associati a funzioni che possono avere i propri tag. Gli strumenti di mappatura dell'origine degli eventi non ereditano automaticamente i tag dalle funzioni. Puoi utilizzare l' AWS Lambda API per visualizzare e aggiornare i tag. Puoi anche visualizzare e aggiornare i tag mentre gestisci la mappatura della sorgente di un evento specifico nella console Lambda, inclusi quelli che utilizzano la modalità provvisoria per Amazon SQS.

### Autorizzazioni necessarie per lavorare con i tag

Per consentire a un'identità AWS Identity and Access Management (IAM) (utente, gruppo o ruolo) di leggere o impostare tag su una risorsa, concedile le autorizzazioni corrispondenti:

- `lambda: ListTags` —Quando una risorsa ha dei tag, concedi questa autorizzazione a chiunque debba richiamarla. `ListTags` Per le funzioni con tag, questa autorizzazione è necessaria anche per `GetFunction`.
- `lambda: TagResource` —Concedi questa autorizzazione a chiunque debba chiamare `TagResource` o eseguire un tag durante la creazione.

Facoltativamente, valuta la possibilità di concedere anche l'`UntagResource` autorizzazione `lambda:` per consentire `UntagResource` le chiamate alla risorsa.

Per ulteriori informazioni, consulta [Identity-based Politiche IAM per Lambda](#).

### Utilizzo di tag con la console Lambda

Puoi utilizzare la console Lambda per creare mappature delle sorgenti di eventi con tag, aggiungere tag alle mappature delle sorgenti di eventi esistenti e filtrare le mappature delle sorgenti di eventi per tag, comprese quelle configurate in modalità provvisoria per Amazon SQS.

Quando aggiungi un trigger per i servizi basati su flussi e code supportati tramite la console Lambda, Lambda crea automaticamente uno strumento di mappatura dell'origine degli eventi. Per ulteriori informazioni su queste origini eventi, consulta [the section called “Strumento di mappatura dell'origine degli eventi”](#). Per creare una mappatura delle sorgenti di eventi nella console, sono necessari i seguenti prerequisiti:

- Una funzione .
- Un'origine eventi proveniente da un servizio interessato.

È possibile aggiungere i tag come parte della stessa interfaccia utente utilizzata per creare o aggiornare i trigger.

Per aggiungere un tag durante la creazione di uno strumento di mappatura dell'origine degli eventi

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione .
3. In Panoramica delle funzioni, scegliere Aggiungi trigger.
4. In Configurazione trigger, nell'elenco a discesa, scegli il nome del servizio da cui proviene l'origine eventi.
5. Fornisci la configurazione di base per la tua origine eventi. Per ulteriori informazioni sulla configurazione dell'origine eventi, consulta la sezione per il servizio correlato in [the section called “Integrazione con altri servizi”](#).
6. In Configurazione dello strumento di mappatura dell'origine degli eventi, scegli Impostazioni aggiuntive.
7. In Tag, seleziona Aggiungi nuovo tag.
8. Nel campo Chiave, inserisci la chiave del tag. Per informazioni sulle restrizioni relative ai tag, consulta Limiti e requisiti per la denominazione dei [tag](#) nella Tagging AWS Resources and Tag Editor Guide.
9. Scegliere Aggiungi.

Per aggiungere tag a uno strumento di mappatura dell'origine degli eventi esistente

1. Apri [Strumenti di mappatura dell'origine degli eventi](#) nella console Lambda.
2. Dall'elenco delle risorse, scegli l'UUID per lo strumento di mappatura dell'origine degli eventi corrispondente alla funzione e all'ARN dell'origine eventi.
3. Dall'elenco delle schede sotto il riquadro di configurazione generale, scegli Tag.
4. Scegliere Gestisci tag.
5. Scegliere Aggiungi nuovo tag.

6. Nel campo Chiave, inserisci la chiave del tag. Per informazioni sulle restrizioni di etichettatura, consulta Limiti e requisiti per la denominazione dei [tag](#) nella AWS Tagging Resources and Tag Editor Guide.
7. Selezionare Salva.

Per filtrare gli strumenti di mappatura dell'origine degli eventi per tag

1. Apri [Strumenti di mappatura dell'origine degli eventi](#) nella console Lambda.
2. Scegli la barra di ricerca.
3. Dall'elenco a discesa, seleziona la chiave di tag sotto Tag.
4. Seleziona Usa: "nome-tag" per vedere tutti gli strumenti di mappatura dell'origine degli eventi etichettati con questa chiave, oppure scegli un operatore per filtrare ulteriormente in base al valore.
5. Seleziona il valore del tag da filtrare in base a una combinazione di chiave e valore del tag.

La barra di ricerca supporta anche la ricerca di chiavi di tag. Immetti il nome di una chiave per trovarla nell'elenco.

## Utilizzo dei tag con AWS CLI

Puoi aggiungere e rimuovere tag sulle risorse Lambda esistenti, inclusi gli strumenti di mappatura dell'origine degli eventi, con l'API Lambda. Puoi anche aggiungere tag durante la creazione di una mappatura della sorgente di eventi per mantenere una risorsa contrassegnata durante l'intero ciclo di vita.

## Aggiornamento dei tag con le API dei tag Lambda

Puoi aggiungere e rimuovere tag per le risorse Lambda supportate tramite le [TagResource](#) operazioni e API. [UntagResource](#)

Puoi chiamare queste operazioni tramite la AWS CLI. Per aggiungere i tag a una risorsa esistente, utilizza il comando `tag-resource`. Questo esempio aggiunge due tag, uno con la chiave *Department* e uno con la chiave *CostCenter*.

```
aws lambda tag-resource \  
--resource arn:aws:lambda:us-east-2:123456789012:resource-type:my-resource \  
--tags Department=Marketing, CostCenter=1234ABCD
```

Per rimuovere i tag, utilizza il comando `untag-resource`. Questo esempio rimuove il tag con la chiave *Department*.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:resource-  
type:resource-identifier \  
--tag-keys Department
```

Aggiunta di tag durante la creazione di uno strumento di mappatura dell'origine degli eventi

Per creare una nuova mappatura della sorgente di eventi Lambda con i tag, usa l'operazione [CreateEventSourceMapping](#) API. Specifica il parametro `Tags`. Puoi chiamare questa operazione con il `create-event-source-mapping` AWS CLI comando e l'`--tags` opzione. Per ulteriori informazioni sul comando della CLI, consulta [create-event-source-mapping](#) in Riferimento ai comandi della AWS CLI .

Prima di utilizzare il parametro `Tags` con `CreateEventSourceMapping`, assicurati che il tuo ruolo disponga dell'autorizzazione per etichettare le risorse oltre alle normali autorizzazioni necessarie per questa operazione. Per ulteriori informazioni sulle autorizzazioni per il tagging, consulta [the section called "Autorizzazioni necessarie per lavorare con i tag"](#).

Visualizzazione dei tag con le API dei tag Lambda

Per visualizzare i tag applicati a una risorsa Lambda specifica, utilizza l'operazione API `ListTags`. Per ulteriori informazioni, consulta [ListTags](#).

Puoi chiamare questa operazione con il `list-tags` AWS CLI comando fornendo un ARN (Amazon Resource Name).

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:resource-  
type:resource-identifier
```

Filtro delle risorse per tag

Puoi utilizzare l'operazione AWS Resource Groups Tagging API [GetResources](#) API per filtrare le tue risorse per tag. L'operazione `GetResources` riceve fino a 10 filtri, ognuno dei quali contenente una chiave di tag e un massimo di 10 valori di tag. Fornisci `GetResources` con un `ResourceType` per filtrare in base a tipi di risorse specifiche.

Puoi chiamare questa operazione usando il `get-resources` AWS CLI comando. Per esempi di utilizzo di `get-resources`, consulta [get-resources](#) nella Riferimento ai comandi CLI di AWS .

## Controllare gli eventi che Lambda invia alla funzione

Puoi utilizzare il filtraggio degli eventi per controllare quali record di un flusso o di una coda Lambda invia alla funzione. Ad esempio, puoi aggiungere un filtro in modo che la tua funzione elabori solo i messaggi Amazon SQS contenenti determinati parametri di dati. Il filtraggio degli eventi funziona solo con determinati strumenti di mappatura dell'origine degli eventi. È possibile aggiungere filtri alle mappature delle sorgenti degli eventi per quanto segue: Servizi AWS

- Amazon DynamoDB
- Flusso di dati Amazon Kinesis
- Amazon MQ
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Self-managed Apache Kafka
- Amazon Simple Queue Service (Amazon SQS)

Per informazioni specifiche sul filtraggio con origini eventi specifiche, consulta [the section called “Utilizzo di filtri con diversi Servizi AWS”](#). Lambda non supporta il filtraggio degli eventi per Amazon DocumentDB.

Per impostazione predefinita, è possibile definire fino a cinque filtri diversi per una singola mappatura dell'origine degli eventi. I filtri sono logicamente elaborati come alternative nel loro insieme. Se un record proveniente dall'origine dell'evento soddisfa uno o più filtri, Lambda include il record nell'evento successivo che invia alla funzione. Se nessuno dei filtri è soddisfatto, Lambda scarta il record.

### Note

Se devi definire più di cinque filtri per un'origine eventi, puoi richiedere un aumento della quota fino a 10 filtri per ciascuna origine eventi. Se si tenta di aggiungere più filtri rispetto a quelli consentiti dalla quota corrente, Lambda restituisce un errore quando si tenta di creare l'origine dell'evento.

### Argomenti

- [Nozioni di base sul filtraggio di eventi](#)
- [Gestione dei record che non soddisfano i criteri di filtraggio](#)

- [Sintassi delle regole di filtro](#)
- [Collegamento dei criteri di filtro a una mappatura dell'origine evento \(console\)](#)
- [Allegare criteri di filtro alla mappatura della sorgente di un evento \(AWS CLI\)](#)
- [Allegare criteri di filtro a una mappatura della sorgente di un evento \(AWS SAM\)](#)
- [Crittografia dei criteri di filtro](#)
- [Utilizzo di filtri con diversi Servizi AWS](#)

## Nozioni di base sul filtraggio di eventi

Un oggetto criterio di filtro (`FilterCriteria`) è una struttura costituita da un elenco di filtri (`Filters`). Ogni filtro è una struttura che definisce un modello di filtraggio degli eventi (`Pattern`). Un modello è una rappresentazione di stringa di una regola di filtraggio JSON. La struttura di un oggetto `FilterCriteria` è come segue.

```
{
  "Filters": [
    {
      "Pattern": "{ \"Metadata1\": [ rule1 ], \"data\": { \"Data1\": [ rule2 ] } }"
    }
  ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
  "Metadata1": [ rule1 ],
  "data": {
    "Data1": [ rule2 ]
  }
}
```

Il modello di filtraggio può includere proprietà dei metadati, proprietà dei dati o entrambe. I parametri dei metadati disponibili e il formato dei parametri dei dati variano a seconda del Servizio AWS che funge da origine dell'evento. Ad esempio, supponiamo che la mappatura dell'origine degli eventi riceva il seguente record da una coda Amazon SQS:

```
{
```

```

    "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
    "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
    "body": "{\\n \\\"City\\\": \\\"Seattle\\\",\\n \\\"State\\\": \\\"WA\\\",\\n \\\"Temperature\\\": \\\"46\\\"\\n}",
    "attributes": {
      "ApproximateReceiveCount": "1",
      "SentTimestamp": "1545082649183",
      "SenderId": "AIDAIENQZJOL023YVJ4V0",
      "ApproximateFirstReceiveTimestamp": "1545082649185"
    },
    "messageAttributes": {},
    "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
    "eventSource": "aws:sqs",
    "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
    "awsRegion": "us-east-2"
  }

```

- Le Proprietà dei metadati sono i campi contenenti informazioni sull'evento che ha creato il record. Nel record Amazon SQS di esempio, le proprietà dei metadati includono campi come `messageId`, `eventSourceArn` e `awsRegion`.
- Le Proprietà dei dati sono i campi del record contenenti i dati del flusso o della coda. Nell'esempio di evento Amazon SQS, la chiave per il campo dati è `body` e le proprietà dei dati sono i campi `City`, `State` e `Temperature`.

Diversi tipi di origine degli eventi utilizzano valori di chiave differenti per i rispettivi campi di dati. Per filtrare le proprietà dei dati, assicurati di utilizzare la chiave corretta nello schema del filtro. Per un elenco delle chiavi di filtraggio dei dati e per visualizzare esempi di modelli di filtro per ciascuna di esse supportate Servizio AWS, fai riferimento a [Utilizzo di filtri con diversi Servizi AWS](#).

Il filtraggio degli eventi è in grado di gestire il filtraggio JSON multi-livello. Considera, ad esempio, il seguente frammento di un record da un flusso DynamoDB:

```

"dynamodb": {
  "Keys": {
    "ID": {
      "S": "ABCD"
    }
    "Number": {
      "N": "1234"
    }
  },
  ...
}

```



```
}
```

Supponi di voler elaborare solo i record in cui il valore della chiave di ordinamento `Number` è 4567. In questo caso, l'oggetto `FilterCriteria` appare così:

```
{
  "Filters": [
    {
      "Pattern": "{ \"dynamodb\": { \"Keys\": { \"Number\": { \"N\": [ \"4567\" ] } } } }"
    }
  ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
  "dynamodb": {
    "Keys": {
      "Number": {
        "N": [ "4567" ]
      }
    }
  }
}
```

## Gestione dei record che non soddisfano i criteri di filtraggio

Il modo in cui Lambda gestisce i record che non soddisfano i criteri di filtro dipende dall'origine eventi.

- Per Amazon SQS, se un messaggio non soddisfa i criteri di filtraggio, Lambda rimuove automaticamente il messaggio dalla coda. Non è necessario eliminare manualmente questi messaggi in Amazon SQS.
- Per Kinesis e DynamoDB, una volta che i criteri di filtraggio valutano un record, l'iteratore di flussi passa oltre questo record. Se il registro non soddisfa i criteri di filtro, non è necessario eliminare manualmente il registro dall'origine evento. Dopo il periodo di conservazione, Kinesis e DynamoDB eliminano automaticamente questi vecchi record. Se vuoi che i record vengano eliminati prima, consulta [Modifica del periodo di conservazione dei dati](#).
- Per i messaggi Amazon MSK, Apache Kafka autogestiti e Amazon MQ, Lambda elimina i messaggi che non corrispondono a tutti i campi inclusi nel filtro. Per Amazon MSK e Apache

Kafka autogestito, Lambda esegue il commit degli offset per i messaggi corrispondenti e non corrispondenti dopo aver richiamato correttamente la funzione. Per Amazon MQ, Lambda riconosce i messaggi corrispondenti dopo aver richiamato con successo la funzione e riconosce i messaggi non corrispondenti quando li filtra.

## Sintassi delle regole di filtro

Per le regole di filtro, Lambda supporta le EventBridge regole Amazon e utilizza la stessa sintassi di EventBridge. Per ulteriori informazioni, consulta i modelli di EventBridge eventi di [Amazon](#) nella Amazon EventBridge User Guide.

Di seguito è riportato un riepilogo di tutti gli operatori di confronto disponibili per il filtro eventi Lambda.

| Operatore di confronto                  | Esempio                                           | Sintassi delle regole                                               |
|-----------------------------------------|---------------------------------------------------|---------------------------------------------------------------------|
| Null                                    | UserID è nullo                                    | "UserID": [ nullo ]                                                 |
| Empty                                   | LastName è vuoto                                  | "LastName": [ "" ]                                                  |
| Equals                                  | Il nome è "Alice"                                 | "Nome": [ "Alice" ]                                                 |
| Uguale a (ignora maiuscole e minuscole) | Il nome è "Alice"                                 | "Name": [ { "equals-ignore-case": "alice" } ]                       |
| And                                     | La posizione è "New York" e il giorno è "lunedì"  | "Luogo": [ "New York" ],<br>"Giorno": [ "lunedì" ]                  |
| Or                                      | PaymentType è «Credito» o «Debito»                | "PaymentType": [ «Credito», «Debito» ]                              |
| Or (campi multipli)                     | La posizione è "New York" o il giorno è "lunedì". | "\$or": [ { "Location": [ "New York" ] }, { "Day": [ "Monday" ] } ] |
| Not                                     | Il tempo è qualsiasi tranne "piovoso"             | "Meteo": [ { "anything-but": [ "Piove" ] } ]                        |
| Numerico (uguale)                       | Il prezzo è 100                                   | "Prezzo": [ { "numerico": [ "=", 100 ] } ]                          |

| Operatore di confronto | Esempio                                              | Sintassi delle regole                               |
|------------------------|------------------------------------------------------|-----------------------------------------------------|
| Numerico (intervallo)  | Il prezzo è superiore a 10 e inferiore o uguale a 20 | "Prezzo": [ { "numerico": [ ">", 10, "<=", 20 ] } ] |
| Exists                 | ProductName esiste                                   | "ProductName«: [{«exists»: true}]                   |
| Does not exist         | ProductName non esiste                               | "ProductName«: [{«exists»: false}]                  |
| Begins with            | La Regione è negli Stati Uniti                       | "Regione": [ { "prefisso": "us-" } ]                |
| Ends with              | FileName termina con un'estensione .png.             | "FileName«: [{«suffix»: «.png»}]                    |

#### Note

Ad esempio EventBridge, per le stringhe, Lambda utilizza una corrispondenza esatta carattere per carattere senza la piegatura delle maiuscole e minuscole o qualsiasi altra normalizzazione delle stringhe. Per i numeri Lambda utilizza anche la rappresentazione di stringhe. Ad esempio, 300, 300.0 e 3.0e2 non sono considerati uguali.

Tieni presente che l'operatore Exists funziona solo sui nodi foglia nell'origine eventi JSON. Non corrisponde ai nodi intermedi. Ad esempio, con il seguente codice JSON, lo schema di filtro { "person": { "address": [ { "exists": true } ] } } non troverebbe una corrispondenza perché "address" è un nodo intermedio.

```
{
  "person": {
    "name": "John Doe",
    "age": 30,
    "address": {
      "street": "123 Main St",
      "city": "Anytown",
      "country": "USA"
    }
  }
}
```

```
}  
}
```

## Collegamento dei criteri di filtro a una mappatura dell'origine evento (console)

Seguire questi passaggi per creare una nuova mappatura delle origini eventi con criteri di filtro utilizzando la console Lambda.

Per creare una nuova mappatura dell'origine evento con criteri di filtro (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere il nome di una funzione per la creazione di una mappatura dell'origine evento.
3. In Panoramica delle funzioni, scegliere Aggiungi trigger.
4. Per Trigger configuration (Configurazione trigger), scegliere un tipo di attivazione che supporta il filtraggio degli eventi. Per un elenco dei servizi supportati, consulta l'elenco all'inizio di questa pagina.
5. Espandere Additional settings (Impostazioni aggiuntive).
6. Alla voce Filter criteria (Criteri di filtro), seleziona Add (Aggiungi), quindi definisci e immetti i filtri. Ad esempio, è possibile inserire i seguenti valori.

```
{ "Metadata" : [ 1, 2 ] }
```

Ciò indica a Lambda di elaborare solo i registri in cui il campo Metadata è uguale a 1 o 2. Puoi continuare a selezionare Aggiungi per aggiungere altri filtri fino al numero massimo consentito.

7. Una volta completata l'aggiunta di filtri, scegli Salva.

Quando si inseriscono i criteri di filtraggio utilizzando la console, si inserisce solo il modello di filtraggio e non è necessario fornire la chiave Pattern o le virgolette di escape. Nel passaggio 6 delle istruzioni precedenti, { "Metadata" : [ 1, 2 ] } corrisponde ai FilterCriteria seguenti.

```
{  
  "Filters": [  
    {  
      "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"  
    }  
  ]  
}
```

```
}
```

Dopo aver creato la mappatura dell'origine eventi nella console, è possibile visualizzare il `FilterCriteria` formattato nei dettagli del trigger. Per altri esempi di creazione di filtri per gli eventi utilizzando la console, consulta la pagina [Utilizzo di filtri con diversi Servizi AWS](#).

## Allegare criteri di filtro alla mappatura della sorgente di un evento (AWS CLI)

Supponiamo che si desideri che una mappatura dell'origine eventi abbia i seguenti `FilterCriteria`:

```
{
  "Filters": [
    {
      "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
    }
  ]
}
```

Per creare una nuova mappatura della sorgente di eventi con questi criteri di filtro utilizzando AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"Metadata\" : [ 1, 2 ] }"}]}'
```

Il comando [create-event-source-mapping](#) crea una nuova mappatura dell'origine degli eventi Amazon SQS per la funzione `my-function` con i `FilterCriteria` specificati.

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"Metadata\" : [ 1, 2 ] }"}]}'
```

Si noti che per aggiornare una mappatura di origine eventi, è necessario il suo UUID. È possibile ottenere l'UUID tramite una chiamata [list-event-source-mappings](#). Lambda restituisce anche l'UUID nella risposta della CLI [create-event-source-mapping](#).

Per rimuovere i criteri di filtraggio da un'origine degli eventi, è possibile eseguire il comando [update-event-source-mapping](#) con un oggetto `FilterCriteria` vuoto.

```
aws lambda update-event-source-mapping \  
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \  
  --filter-criteria "{}"
```

Per altri esempi di creazione di filtri di eventi utilizzando il AWS CLI, vedere [Utilizzo di filtri con diversi Servizi AWS](#).

## Allegare criteri di filtro a una mappatura della sorgente di un evento (AWS SAM)

Supponiamo di voler configurare una fonte di eventi in modo AWS SAM da utilizzare i seguenti criteri di filtro:

```
{  
  "Filters": [  
    {  
      "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"  
    }  
  ]  
}
```

Per aggiungere questi criteri di filtro allo strumento di mappatura dell'origine degli eventi, inserisci il seguente frammento nel modello YAML delle origini eventi.

```
FilterCriteria:  
  Filters:  
    - Pattern: '{"Metadata": [1, 2]}'
```

Per ulteriori informazioni sulla creazione e la configurazione di un AWS SAM modello per la mappatura dell'origine di un evento, consulta la [EventSource](#) sezione della AWS SAM Developer Guide. Per altri esempi di creazione di filtri per eventi utilizzando i AWS SAM modelli, consulta [Utilizzo di filtri con diversi Servizi AWS](#)

## Crittografia dei criteri di filtro

Per impostazione predefinita, Lambda non crittografa l'oggetto dei criteri di filtro. Per i casi d'uso in cui potresti includere informazioni sensibili nell'oggetto dei criteri di filtro, puoi utilizzare la tua chiave [KMS](#) per crittografarle.

Dopo aver crittografato l'oggetto dei criteri di filtro, puoi visualizzarne la versione in testo normale utilizzando una chiamata API. [GetEventSourceMapping](#). È necessario disporre delle autorizzazioni `kms:Decrypt` per poter visualizzare correttamente i criteri di filtro in testo non crittografato.

#### Note

Se l'oggetto dei criteri di filtro è crittografato, Lambda oscura il valore del `FilterCriteria` campo nella risposta alle chiamate. [ListEventSourceMappings](#). Invece, questo campo viene visualizzato come `null`. Per vedere il vero valore di `FilterCriteria`, usa l'[GetEventSourceMapping](#) API.

Per visualizzare il valore decrittografato di `FilterCriteria` nella console, assicurati che il tuo ruolo IAM contenga le autorizzazioni per. [GetEventSourceMapping](#)

Puoi specificare la tua chiave KMS tramite la console,, o. API/CLI CloudFormation

Per crittografare i criteri di filtro con una chiave KMS di proprietà del cliente (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Selezionare Add trigger (Aggiungi trigger). Se hai già un trigger esistente, scegli la scheda Configurazione, quindi scegli Trigger. Seleziona il trigger esistente e scegli Modifica.
3. Seleziona la casella di controllo accanto a Crittografa con chiave KMS gestita dal cliente.
4. Per Scegli una chiave di crittografia KMS gestita dal cliente, seleziona una chiave abilitata esistente o crea una nuova chiave. A seconda dell'operazione, sono necessarie alcune o tutte le seguenti autorizzazioni: `kms:DescribeKey`, `kms:GenerateDataKey` e `kms:Decrypt`. Utilizza la policy delle chiavi KMS per concedere queste autorizzazioni.

Se utilizzi la tua chiave KMS, le seguenti operazioni API devono essere permesse nella [policy della chiave](#):

- `kms:Decrypt`: deve essere concesso al principale del servizio Lambda regionale (`lambda.AWS_region.amazonaws.com`). Ciò permette a Lambda per decrittografare i dati con questa chiave KMS.
- Per evitare un [problema di confused deputy tra servizi](#), la policy della chiave utilizza la chiave di condizione globale [aws:SourceArn](#). Il valore corretto della chiave `aws:SourceArn` è l'ARN della risorsa dello strumento di mappatura dell'origine degli eventi, quindi puoi aggiungerlo alla tua policy solo dopo averne rilevato l'ARN. Lambda inoltra inoltre le chiavi

`aws:lambda:FunctionArn` e `aws:lambda:EventSourceArn` e i rispettivi valori nel [contesto di crittografia](#) quando effettua una richiesta di decrittografia a KMS. Affinché la richiesta di decrittografia abbia successo, questi valori devono corrispondere alle condizioni specificate nella policy della chiave. Non è necessario includere `EventSourceArn` per Self-managed Kafka le sorgenti di eventi in quanto non dispongono di un `EventSourceArn`.

- `kms:Decrypt`— Deve essere concesso anche al principale che intende utilizzare la chiave per visualizzare i criteri di filtro in [GetEventSourceMapping](#) testo normale o le chiamate API. [DeleteEventSourceMapping](#)
- `kms:DescribeKey`: fornisce i dettagli della chiave gestita dal cliente per consentire al principale specificato di utilizzare la chiave.
- `kms:GenerateDataKey`: fornisce le autorizzazioni a Lambda per generare una chiave di dati per crittografare i criteri di filtro per conto del principale specificato ([crittografia a busta](#)).

Puoi utilizzarlo AWS CloudTrail per tenere traccia AWS KMS delle richieste effettuate da Lambda per tuo conto. Per esempi di CloudTrail eventi, consulta [???](#).

Consigliamo inoltre di utilizzare la chiave di condizione [kms:ViaService](#) per limitare l'uso della chiave KMS solo alle richieste provenienti da Lambda. Il valore di questa chiave è il principale del servizio Lambda regionale (`lambda.AWS_region.amazonaws.com`). Di seguito è riportata una policy della chiave di esempio che concede tutte le autorizzazioni pertinenti:

Example AWS KMS politica chiave

JSON

```
{
  "Version": "2012-10-17",
  "Id": "example-key-policy-1",
  "Statement": [
    {
      "Sid": "Allow Lambda to decrypt using the key",
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.us-east-1.amazonaws.com"
      },
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "*"
    }
  ]
}
```



```

        "Condition": {
            "ArnEquals" : {
                "aws:SourceArn": [
                    "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:<esm_uuid>"
                ]
            },
            "StringEquals": {

                "kms:EncryptionContext:aws:lambda:FunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:test-function",

                "kms:EncryptionContext:aws:lambda:EventSourceArn": "arn:aws:sqs:us-
east-1:123456789012:test-queue"
            }
        },
        {
            "Sid": "Allow actions by an AWS account on the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow use of the key to specific roles",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/ExampleRole"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals" : {
                    "kms:ViaService": "lambda.us-east-1.amazonaws.com"
                }
            }
        }
    ]
}

```

```
]
}
```

Per utilizzare la tua chiave KMS per crittografare i criteri di filtro, puoi anche usare il seguente [CreateEventSourceMapping](#) AWS CLI comando. Specifica l'ARN della chiave KMS con il flag `--kms-key-arn`.

```
aws lambda create-event-source-mapping --function-name my-function \
  --maximum-batching-window-in-seconds 60 \
  --event-source-arn arn:aws:sqs:us-east-1:123456789012:my-queue \
  --filter-criteria "{\"filters\": [{\"pattern\": \"{\\\"a\\\": [\\\"1\\\", \\\"2\\\"]}\" }]}\" \
  --kms-key-arn arn:aws:kms:us-east-1:123456789012:key/055efbb4-xmpl-4336-  
ba9c-538c7d31f599
```

Se disponi di una mappatura della sorgente degli eventi esistente, utilizza invece il [UpdateEventSourceMapping](#) AWS CLI comando. Specifica l'ARN della chiave KMS con il flag `--kms-key-arn`.

```
aws lambda update-event-source-mapping --function-name my-function \
  --maximum-batching-window-in-seconds 60 \
  --event-source-arn arn:aws:sqs:us-east-1:123456789012:my-queue \
  --filter-criteria "{\"filters\": [{\"pattern\": \"{\\\"a\\\": [\\\"1\\\", \\\"2\\\"]}\" }]}\" \
  --kms-key-arn arn:aws:kms:us-east-1:123456789012:key/055efbb4-xmpl-4336-  
ba9c-538c7d31f599
```

Questa operazione sovrascrive qualsiasi chiave KMS specificata in precedenza. Se specifichi il flag `--kms-key-arn` insieme a un argomento vuoto, Lambda smette di usare la chiave KMS per crittografare i criteri di filtro. Invece, per impostazione predefinita, Lambda utilizza una chiave Amazon-owned.

Per specificare la tua chiave KMS in un CloudFormation modello, utilizza la `KMSKeyArn` proprietà del tipo di risorsa `AWS::Lambda::EventSourceMapping`. Ad esempio, puoi inserire il seguente frammento di codice nel modello YAML dell'origine eventi.

```
MyEventSourceMapping:
  Type: AWS::Lambda::EventSourceMapping
  Properties:
    ...
    FilterCriteria:
      Filters:
```

```
- Pattern: '{"a": [1, 2]}'
KMSKeyArn: "arn:aws:kms:us-east-1:123456789012:key/055efbb4-xmpl-4336-
ba9c-538c7d31f599"
...
```

Per poter visualizzare i criteri di filtro crittografati in chiaro in una chiamata [GetEventSourceMapping](#) or [DeleteEventSourceMapping](#) API, devi disporre delle autorizzazioni. `kms:Decrypt`

A partire dal 6 agosto 2024, il `FilterCriteria` campo non viene più visualizzato nei AWS CloudTrail log di [CreateEventSourceMapping](#) e nelle chiamate [DeleteEventSourceMapping](#) API se la funzione non utilizza il filtro degli eventi. [UpdateEventSourceMapping](#) Se la tua funzione utilizza il filtro degli eventi, il campo `FilterCriteria` viene visualizzato come vuoto (`{}`). Puoi comunque visualizzare i criteri di filtro in chiaro nella risposta alle chiamate [GetEventSourceMapping](#) API se disponi delle `kms:Decrypt` autorizzazioni per la chiave KMS corretta.

Esempio di registrazione per CloudTrail le chiamate `Create/Update/DeleteEventSourceMapping`

Nel seguente AWS CloudTrail esempio di voce di registro per una `CreateEventSourceMapping` chiamata, `FilterCriteria` viene visualizzata come vuota (`{}`) perché la funzione utilizza il filtro degli eventi. Questo è il caso anche se l'oggetto `FilterCriteria` contiene criteri di filtro validi che la funzione utilizza attivamente. Se la funzione non utilizza il filtro degli eventi, CloudTrail non visualizzerà affatto il `FilterCriteria` campo nelle voci di registro.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAI23456789EXAMPLE:user1",
    "arn": "arn:aws:sts::123456789012:assumed-role/Example/example-role",
    "accountId": "123456789012",
    "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAI987654321EXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/User1",
        "accountId": "123456789012",
        "userName": "User1"
      },
      "webIdFederationData": {},
      "attributes": {
        "creationDate": "2024-05-09T20:35:01Z",
```

```

        "mfaAuthenticated": "false"
      },
    },
    "invokedBy": "AWS Internal"
  },
  "eventTime": "2024-05-09T21:05:41Z",
  "eventSource": "lambda.amazonaws.com",
  "eventName": "CreateEventSourceMapping20150331",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "AWS Internal",
  "userAgent": "AWS Internal",
  "requestParameters": {
    "eventSourceArn": "arn:aws:sqs:us-east-2:123456789012:example-queue",
    "functionName": "example-function",
    "enabled": true,
    "batchSize": 10,
    "filterCriteria": {},
    "kMSKeyArn": "arn:aws:kms:us-east-2:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
    "scalingConfig": {},
    "maximumBatchingWindowInSeconds": 0,
    "sourceAccessConfigurations": []
  },
  "responseElements": {
    "uUID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEEaaaaa",
    "batchSize": 10,
    "maximumBatchingWindowInSeconds": 0,
    "eventSourceArn": "arn:aws:sqs:us-east-2:123456789012:example-queue",
    "filterCriteria": {},
    "kMSKeyArn": "arn:aws:kms:us-east-2:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
    "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:example-
function",
    "lastModified": "May 9, 2024, 9:05:41 PM",
    "state": "Creating",
    "stateTransitionReason": "USER_INITIATED",
    "functionResponseTypes": [],
    "eventSourceMappingArn": "arn:aws:lambda:us-east-2:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb"
  },
  "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
  "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "readOnly": false,
  "eventType": "AwsApiCall",

```

```

    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "sessionCredentialFromConsole": "true"
  }

```

## Utilizzo di filtri con diversi Servizi AWS

Diversi tipi di origine degli eventi utilizzano valori di chiave differenti per i rispettivi campi di dati. Per filtrare le proprietà dei dati, assicurati di utilizzare la chiave corretta nello schema del filtro. La tabella seguente fornisce le chiavi di filtro per ciascuna delle chiavi supportate Servizio AWS.

| Servizio AWS              | Chiave di filtraggio |
|---------------------------|----------------------|
| DynamoDB                  | dynamodb             |
| Kinesis                   | data                 |
| Amazon MQ                 | data                 |
| MSK Amazon                | value                |
| Self-managed Apache Kafka | value                |
| Amazon SQS                | body                 |

Le sezioni seguenti forniscono esempi di modelli di filtraggio per diversi tipi di origini eventi. Forniscono inoltre definizioni dei formati di dati in entrata supportati e dei formati del corpo dei modelli di filtraggio per ciascun servizio supportato.

- [the section called “Filtro eventi”](#)
- [the section called “Filtro eventi”](#)
- [the section called “Filtro eventi”](#)
- [the section called “Filtro eventi”](#)
- [the section called “Filtro eventi”](#)

## Test delle funzioni Lambda nella console

È possibile testare la funzione Lambda nella console richiamando la funzione con un evento di test. Un evento di test è un input JSON per la funzione. Se la funzione non richiede input, l'evento può essere un documento ( `{}` ) vuoto.

Quando esegui un test nella console, Lambda invoca la funzione in modo sincrono con l'evento di test. Il runtime della funzione converte il JSON di evento in un oggetto e lo passa al metodo del gestore del codice per l'elaborazione.

### Creare un evento di test

Prima di poter eseguire il test nella console, devi creare un evento di test privato o condivisibile.

## Invocare funzioni con eventi di test

### Testare una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione che desideri testare.
3. Seleziona la scheda Test.
4. Sotto a Evento di test, scegli Crea nuovo evento o Modifica Evento salvato, quindi seleziona l'evento salvato da utilizzare.
5. Facoltativamente, scegli un Modello per l'evento JSON.
6. Scegli Test (Esegui test).
7. Per esaminare i risultati del test, in Execution result (Risultato esecuzione), espandi Details (Dettagli).

Per invocare la funzione senza salvare l'evento di test scegli Test prima di salvare. Questo crea un evento di test non salvato che Lambda conserverà solo per l'intera durata della sessione.

Per i runtime Node.js, Python e Ruby, puoi anche accedere ai tuoi eventi di test salvati e non salvati esistenti nella scheda Codice. Utilizza la sezione EVENTI DI TEST per creare, modificare ed eseguire test.

## Creazione di eventi di test privati

Gli eventi di test privati sono disponibili solo per il creatore dell'evento e non richiedono autorizzazioni aggiuntive per l'uso. Puoi creare fino a 10 eventi di test per ogni funzione.

Creare un evento di test

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione che desideri testare.
3. Seleziona la scheda Test.
4. Sotto a Test event (evento di test), procedi come segue:
  - a. Seleziona un Template (modello).
  - b. Inserisci un Nome per l'evento di test.
  - c. Nella casella di immissione testo, inserire l'evento di test JSON.
  - d. Sotto Event sharing settings (impostazioni di condivisione degli eventi), scegli Private (privato).
5. Scegli Save changes (Salva modifiche).

Per i Node.js runtime Python e Ruby, puoi anche creare eventi di test nella scheda Codice. Utilizza la sezione EVENTI DI TEST per creare, modificare ed eseguire test.

## Creazione di eventi di test condivisibili

Gli eventi di test condivisibili sono eventi di test che puoi condividere con altri utenti nello stesso account. AWS Puoi modificare gli eventi di test condivisibili di altri utenti e richiamare la funzione con essi.

Lambda salva gli eventi di test condivisibili come schemi in un registro di schemi [Amazon EventBridge \(CloudWatch Events\) denominato](#) `lambda-testevent-schemas` Poiché Lambda utilizza questo registro per archiviare e richiamare gli eventi di test condivisibili creati, ti consigliamo di non modificare questo registro o di creare un registro utilizzando il nome `lambda-testevent-schemas`

Per visualizzare, condividere e modificare eventi di test condivisibili, è necessario disporre delle autorizzazioni per tutte le seguenti operazioni dell'API del registro dello schema [EventBridge \(CloudWatch Events\)](#):

- [schemi. CreateRegistry](#)
- [schemi. CreateSchema](#)
- [schemi. DeleteSchema](#)
- [schemi. DeleteSchemaVersion](#)
- [schemi. DescribeRegistry](#)
- [schemi. DescribeSchema](#)
- [schemi. GetDiscoveredSchema](#)
- [schemi. ListSchemaVersions](#)
- [schemi. UpdateSchema](#)

Ricorda che il salvataggio delle modifiche apportate a un evento di test condivisibile sovrascrive tale evento.

Se non riesci a creare, modificare o visualizzare eventi di test condivisibili, verifica che il tuo account disponga delle autorizzazioni necessarie per queste operazioni. Se disponi delle autorizzazioni richieste ma non riesci ancora ad accedere agli eventi di test condivisibili, verifica eventuali politiche [basate sulle risorse](#) che potrebbero limitare l'accesso al registro (Events). EventBridge CloudWatch

Per creare un evento di test

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione che desideri testare.
3. Seleziona la scheda Test.
4. Sotto a Test event (evento di test), procedi come segue:
  - a. Seleziona un Template (modello).
  - b. Inserisci un Nome per l'evento di test.
  - c. Nella casella di immissione testo, inserire l'evento di test JSON.
  - d. Sotto Event sharing settings (impostazioni di condivisione degli eventi), scegli Shareable (condivisibile).
5. Scegli Save changes (Salva modifiche).



 Usa eventi di test condivisibili con AWS Serverless Application Model.

È possibile utilizzarlo AWS SAM per richiamare eventi di test condivisibili. Vedi [sam remote test-event](#) nella Guida per gli sviluppatori [AWS Serverless Application Model](#)

## Eliminare schemi di eventi di test condivisibili

Quando si eliminano eventi di test condivisibili, Lambda li rimuove dal registro `lambda-testevent-schemas`. Se si rimuove l'ultimo evento di test condivisibile dal Registro di sistema, Lambda elimina il Registro di sistema.

Se elimini la funzione, Lambda non elimina gli schemi di eventi di test condivisibili associati. È necessario pulire queste risorse manualmente dalla console [EventBridge \(CloudWatch Events\)](#).

## Stati funzione Lambda

Lambda include un [campo](#) State nella configurazione della funzione per tutte le funzioni per indicare quando la funzione è pronta per essere richiamata. State fornisce informazioni sullo stato corrente della funzione, incluso se è possibile richiamarla correttamente. Gli stati della funzione non modificano il comportamento delle chiamate della funzione o il modo in cui la funzione esegue il codice.

### Note

Le definizioni dello stato delle funzioni differiscono leggermente per [SnapStart](#) le funzioni. Per ulteriori informazioni, consulta [Lambda SnapStart e stati delle funzioni](#).

In molti casi, una tabella DynamoDB è il modo ideale per mantenere lo stato tra una chiamata e l'altra, poiché fornisce un accesso ai dati a bassa latenza e può essere scalata con il servizio Lambda. Puoi anche archiviare i dati in [Amazon EFS for Lambda](#) se utilizzi questo servizio, che fornisce un accesso a bassa latenza allo storage del file system.

Gli stati delle funzioni includono:

- **Pending** – Dopo che Lambda ha creato la funzione, imposta lo stato su sospeso. Mentre lo stato è in sospeso, Lambda tenta di creare o configurare le risorse per la funzione, ad esempio le risorse VPC o EFS. Lambda non richiama una funzione durante lo stato in sospeso. Qualsiasi chiamata o altra azione API che opera sulla funzione ha esito negativo.
- **Active** – La funzione passa allo stato attivo dopo che Lambda ha completato la configurazione e il provisioning delle risorse. Le funzioni possono essere richiamate correttamente solo quando sono attive.
- **Failed** – Indica che la configurazione o il provisioning delle risorse ha riscontrato un errore. Quando la creazione della funzione fallisce, Lambda imposta lo stato della funzione su Fallita ed è necessario eliminare e ricreare la funzione.
- **Inactive** – Una funzione diventa inattiva quando è stata inattiva abbastanza a lungo perché Lambda recuperi le risorse esterne configurate per essa. Quando si tenta di richiamare una funzione che è inattiva, la chiamata non riesce e Lambda imposta la funzione sullo stato in sospeso fino a quando le risorse della funzione non vengono ricreate. Se Lambda non riesce a ricreare le risorse, la funzione viene reimpostata sullo stato inattivo. Potrebbe essere necessario risolvere eventuali errori e ridistribuire la funzione per ripristinarla allo stato attivo.

Una funzione passa `Inactive` allo stato anche se Lambda perde l'accesso all'artefatto sorgente della funzione. Ciò si applica alle [funzioni di immagine del](#) contenitore in cui l'immagine viene eliminata da Amazon ECR o le autorizzazioni vengono revocate e alle funzioni `.zip` che utilizzano l'archiviazione di codice S3 [autogestita in](#) cui l'oggetto sorgente viene eliminato o le autorizzazioni della policy del bucket vengono revocate. Per ripristinare la funzione, ripristina l'accesso all'artefatto sorgente e aggiorna la funzione.

Se utilizzi flussi di lavoro di SDK-based automazione o chiami direttamente le API di servizio di Lambda, assicurati di controllare lo stato di una funzione prima dell'invocazione per verificare che sia attiva. Puoi farlo con l'azione [GetFunction](#) Lambda API o configurando un cameriere utilizzando l'SDK per Java 2.0. [AWS](#)

```
aws lambda get-function --function-name my-function --query 'Configuration.[State, LastUpdateStatus]'
```

Verrà visualizzato l'output seguente:

```
[
  "Active",
  "Successful"
]
```

Le operazioni seguenti non riescono mentre la creazione della funzione è in attesa:

- [Invoke](#)
- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)
- [PublishVersion](#)

## Stati delle funzioni durante gli aggiornamenti

Lambda ha due operazioni per l'aggiornamento delle funzioni:

- [UpdateFunctionCode](#): aggiorna il pacchetto di distribuzione della funzione
- [UpdateFunctionConfiguration](#): aggiorna la configurazione della funzione

Lambda utilizza l'[LastUpdateStatus](#) attributo per tracciare l'avanzamento di queste operazioni di aggiornamento. Mentre è in corso un aggiornamento (quando "LastUpdateStatus": "InProgress"):

- Lo [stato della funzione](#) rimane `Active`.
- Le chiamate continuano a utilizzare il codice e la configurazione precedenti della funzione fino al completamento dell'aggiornamento.
- Le seguenti operazioni hanno esito negativo:
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)
  - [PublishVersion](#)
  - [TagResource](#)

Quando un aggiornamento fallisce (quando "LastUpdateStatus": "Failed"):

- Lo [stato della funzione](#) rimane `Active`.
- Le chiamate continuano a utilizzare il codice e la configurazione precedenti della funzione.

#### Example GetFunctionConfiguration risposta

L'esempio seguente è il risultato di una [GetFunctionConfiguration](#) richiesta su una funzione in fase di aggiornamento.

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
  "Runtime": "nodejs24.x",
  "VpcConfig": {
    "SubnetIds": [
      "subnet-071f712345678e7c8",
      "subnet-07fd123456788a036",
      "subnet-0804f77612345cacf"
    ],
    "SecurityGroupIds": [
      "sg-085912345678492fb"
    ],
    "VpcId": "vpc-08e1234569e011e83"
  },
}
```

```
"State": "Active",  
"LastUpdateStatus": "InProgress",  
...  
}
```

## Informazioni sul comportamento relativo ai nuovi tentativi in Lambda

Quando invochi una funzione esplicitamente, stabilisci la strategia per la gestione degli errori relativi al codice della funzione. Lambda non riprova automaticamente questi tipi di errori per conto dell'utente. Per riprovare, puoi re-invocare manualmente la funzione, inviare l'evento che presenta l'errore a una coda per il debug o ignorare l'errore. Il codice della funzione può essere eseguito completamente, parzialmente o per niente. Se si riprova, accertare che il codice della funzione sia in grado di gestire lo stesso evento più volte senza provocare transazioni duplicate o altri effetti collaterali indesiderati.

Quando si invoca una funzione in modo indiretto, è necessario conoscere il comportamento del nuovo tentativo dell'invoker e qualsiasi servizio che la richiesta incontra. Questo include i seguenti scenari.

- **Invocazione asincrona** – Lambda tenta due volte gli errori di funzione. Se la funzione non dispone di capacità sufficiente per gestire tutte le richieste in entrata, gli eventi possono attendere in coda per ore o giorni per essere inviati alla funzione. È possibile configurare una coda DLQ sulla funzione per acquisire eventi che non sono stati elaborati. Per ulteriori informazioni, consulta [the section called “Dead-letter code”](#).
- **Mappature evento di origine** – Le mappature evento di origine che leggono dai flussi effettuano nuovi tentativi sull'intero batch di elementi. Errori ripetuti bloccano l'elaborazione degli shard interessati finché l'errore non viene risolto o gli elementi non scadono. Per rilevare shard in stallo, è possibile monitorare il parametro [Età iteratore](#).

Per le mappature origine evento che leggono da una coda, stabilire il tempo trascorso tra i nuovi tentativi e la destinazione degli eventi non riusciti configurando il timeout di visibilità e la policy di reindirizzamento sulla coda di origine. Per ulteriori informazioni, consulta [In che modo Lambda elabora i record provenienti da origini eventi basate su flussi e code](#) e gli argomenti specifici del servizio in [Richiamare Lambda con eventi provenienti da altri AWS services](#).

- **AWS servizi**: i AWS servizi possono richiamare la tua funzione in modo sincrono [o](#) asincrono. Per l'invocazione sincrona, il servizio decide se effettuare un nuovo tentativo. Ad esempio, le operazioni batch Amazon S3 ritentano l'operazione se la funzione Lambda restituisce un codice di risposta `TemporaryFailure`. I servizi che inviano tramite proxy le richieste di un utente o client upstream potrebbero avere una strategia di riprova o potrebbero inoltrare la risposta di errore al richiedente. Ad esempio, API Gateway inoltra sempre di nuovo la risposta di errore al richiedente.

Per l'invocazione asincrona, la logica dei tentativi è la stessa indipendentemente dall'origine di invocazione. Per impostazione predefinita, Lambda ritenta una invocazione asincrona fallita un

massimo di due volte. Per ulteriori informazioni, consulta [In che modo Lambda gestisce gli errori e i nuovi tentativi con una invocazione asincrona](#).

- Altri account e client – Quando si concede l'accesso ad altri account, è possibile usare [policy basate su risorse](#) per limitare i servizi e le risorse che possono configurare per invocare la funzione. Per proteggere la funzione dal sovraccarico, considerare di inserire un livello API davanti alla funzione con [Amazon API Gateway](#). Per informazioni sulla gestione degli errori di API Gateway, consulta [the section called “Errori”](#)

Per aiutarti a gestire gli errori nelle applicazioni Lambda, Lambda si integra con servizi come Amazon e. CloudWatch AWS X-Ray Puoi utilizzare una combinazione di log, parametri, allarmi e tracciamento per rilevare e identificare rapidamente problemi relativi a codice della funzione, API e altre risorse che supportano l'applicazione. Per ulteriori informazioni, consulta [Monitoraggio, debug e risoluzione dei problemi delle funzioni Lambda](#).

# Usa il rilevamento di un ciclo ricorsivo Lambda per prevenire loop infiniti

Quando configuri una funzione Lambda di modo che invii l'output sullo stesso servizio o risorsa che richiama la funzione, è possibile che venga creato un ciclo ricorsivo infinito. Ad esempio, una funzione Lambda potrebbe scrivere un messaggio a una coda di Amazon Simple Queue Service (Amazon SQS), che quindi richiama la stessa funzione. Questa invocazione fa sì che la funzione scriva un altro messaggio nella coda, che a sua volta richiama nuovamente la funzione.

I loop ricorsivi involontari possono comportare la fatturazione di addebiti imprevisti. Account AWS I cicli ricorsivi possono anche far sì che Lambda si [dimensioni](#) e utilizzi tutta la simultaneità disponibile del tuo account. Per contribuire a ridurre l'impatto dei cicli involontari, Lambda è in grado di rilevare determinati tipi di cicli ricorsivi poco dopo che si sono verificati.

Per impostazione predefinita, quando Lambda rileva un ciclo ricorsivo, interrompe l'invocazione della funzione e invia una notifica. Se il tuo progetto utilizza intenzionalmente modelli ricorsivi, puoi modificare la configurazione predefinita di una funzione per consentirne l'invocazione ricorsiva. Per ulteriori informazioni, consulta [the section called “Consentire l'esecuzione di una funzione Lambda in un ciclo ricorsivo”](#).

## Sections

- [Comprendere il rilevamento dei cicli ricorsivi](#)
- [Supportata Servizi AWS e SDK](#)
- [Notifiche dei cicli ricorsivi](#)
- [Risposta alle notifiche di rilevamento dei cicli ricorsivi](#)
- [Consentire l'esecuzione di una funzione Lambda in un ciclo ricorsivo](#)
- [Regioni supportate per il rilevamento dei loop ricorsivi Lambda](#)

## Comprendere il rilevamento dei cicli ricorsivi

Il rilevamento dei cicli ricorsivi in Lambda si basa sul tracciamento gli eventi. Lambda è un servizio di elaborazione basato su eventi che esegue il codice della funzione quando si verificano determinati eventi. Ad esempio, quando un elemento viene aggiunto a una coda Amazon SQS o a un argomento di Amazon Simple Notification Service (Amazon SNS). Lambda trasmette gli eventi alla funzione come oggetti JSON, che contengono informazioni sulla modifica dello stato del sistema. Quando un evento causa l'esecuzione di una funzione, si parla di invocazione.



Per rilevare i cicli ricorsivi, Lambda utilizza le intestazioni di tracciamento [AWS X-Ray](#). Quando dei [Servizi AWS che supportano il rilevamento dei cicli ricorsivi](#) inviano eventi a Lambda, tali eventi vengono automaticamente annotati con metadati. Quando la funzione Lambda scrive uno di questi eventi su un altro supportato Servizio AWS utilizzando una versione [supportata di un SDK, aggiorna questi metadati. AWS](#) I metadati aggiornati includono il numero di volte in cui l'evento ha richiamato la funzione.

#### Note

Non è necessario abilitare l' X-Ray active tracing perché questa funzionalità funzioni. Il rilevamento dei cicli ricorsivi è attivato per impostazione predefinita per tutti i clienti AWS . L'utilizzo della funzionalità è gratuito.

Una catena di richieste è una sequenza di invocazioni Lambda causate dallo stesso evento di attivazione. Ad esempio, immagina che una coda Amazon SQS richiami la tua funzione Lambda. La funzione Lambda invia quindi l'evento elaborato alla stessa coda di Amazon SQS, che richiama nuovamente la funzione. In questo esempio, ogni invocazione della funzione rientra nella stessa catena di richieste.

Se la tua funzione viene richiamata più di 16 volte nella stessa catena di richieste, Lambda interrompe automaticamente la successiva invocazione della funzione in quella catena di richieste e invia una notifica all'utente. Ad esempio, se la funzione è configurata con più trigger, le invocazioni da altri trigger non sono interessate.

#### Note

Quando l'impostazione di `maxReceiveCount` sulla policy di reindirizzamento della coda di origine è superiore a 16, la protezione da ricorsione di Lambda non impedisce ad Amazon SQS di ritentare il messaggio dopo che un loop ricorsivo è stato rilevato e terminato. Quando Lambda rileva un ciclo ricorsivo e annulla le chiamate successive, restituisce `RecursiveInvocationException` allo strumento di mappatura dell'origine degli eventi. Questo incrementa il valore `receiveCount` sul messaggio. Lambda continua a riprovare il messaggio e continua a bloccare le chiamate di funzione, finché Amazon SQS non determina che il limite `maxReceiveCount` è stato superato e invia il messaggio alla coda DLQ configurata.

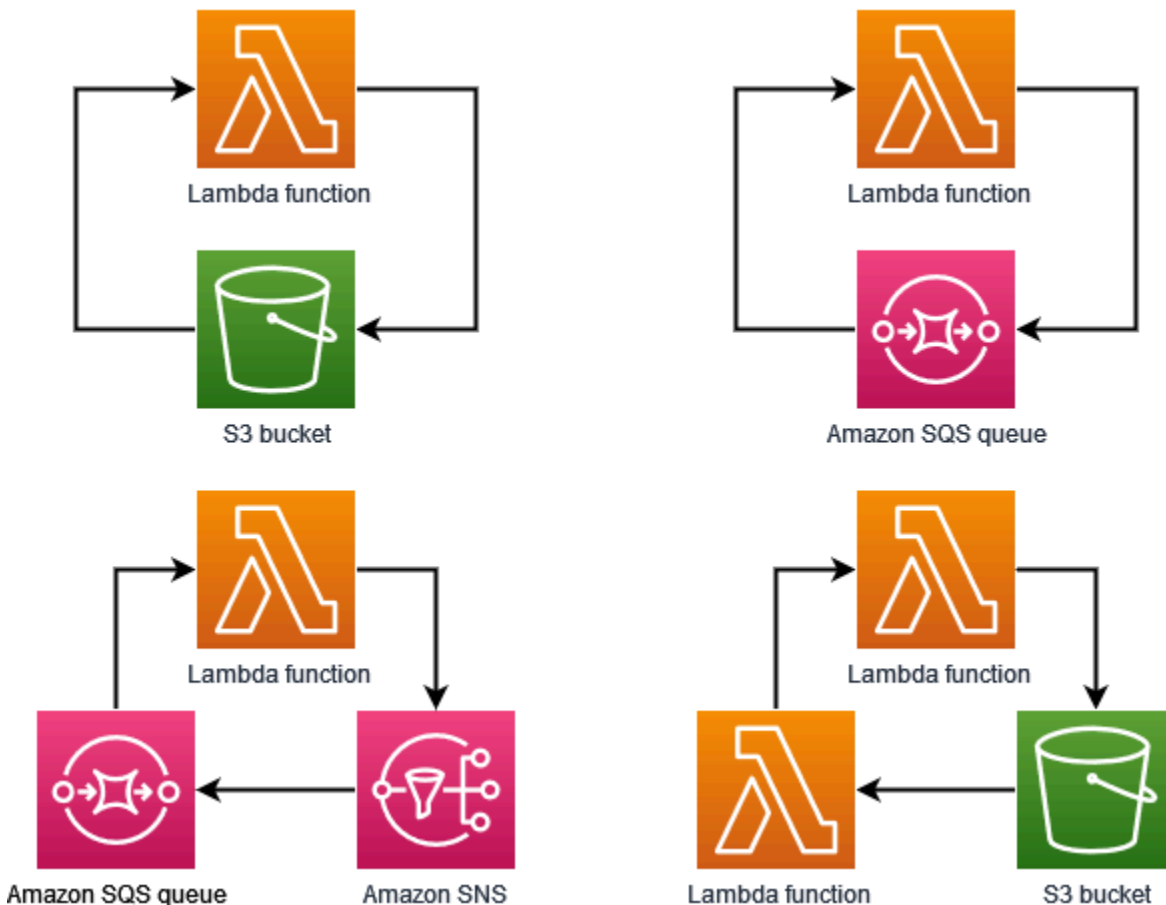
Se hai configurato una [destinazione in caso di errore](#) o una [coda DLQ](#) per la funzione, Lambda invia anche l'evento dall'invocazione interrotta alla destinazione o alla coda DLQ. Quando configuri una coda di destinazione o di lettere morte per la tua funzione, assicurati di non utilizzare un trigger di evento o una mappatura della sorgente di eventi utilizzata anche dalla tua funzione. Se invii eventi alla stessa risorsa che richiama la funzione, potete creare un altro ciclo ricorsivo e anche questo ciclo viene terminato. Se si disattiva il rilevamento del ciclo di ricorsione, questo ciclo non viene terminato.

## Supportata Servizi AWS e SDK

Lambda è in grado di rilevare solo i loop ricorsivi che includono alcuni supportati. Servizi AWS Per rilevare i loop ricorsivi, la funzione deve utilizzare anche uno degli SDK supportati. AWS

### Supportata Servizi AWS

Lambda attualmente rileva i cicli ricorsivi tra le tue funzioni, Amazon SQS, Amazon S3 e Amazon SNS. Lambda rileva anche i loop composti solo da funzioni Lambda, che potrebbero richiamarsi vicendevolmente in modo sincrono o asincrono. I diagrammi seguenti mostrano alcuni esempi di ciclo ricorsivo che Lambda è in grado di rilevare:



Quando un altro Servizio AWS , come Amazon DynamoDB, fa parte del ciclo, Lambda non è attualmente in grado di rilevarlo e interromperlo.

Poiché Lambda attualmente rileva solo i loop ricorsivi che coinvolgono Amazon SQS, Amazon S3 e Amazon SNS, è comunque possibile che i loop che coinvolgono altri possano comportare un utilizzo involontario delle tue funzioni Lambda. Servizi AWS

Per evitare che ti vengano addebitati imprevisti Account AWS, ti consigliamo di configurare gli allarmi Amazon per avvisarti di modelli di utilizzo insoliti. [CloudWatch](#) Ad esempio, puoi configurarlo in modo CloudWatch da ricevere notifiche in caso di picchi di concorrenza o invocazioni di funzioni Lambda. Puoi anche configurare un [avviso di fatturazione](#) che ti segnali quando la spesa nel tuo account supera una soglia da te specificata. In alternativa, puoi utilizzare [AWS Cost Anomaly Detection](#) per ricevere avvisi in merito a schemi di fatturazione insoliti.

### Supportata AWS SDK

Affinché Lambda possa rilevare i cicli ricorsivi, la funzione deve utilizzare un SDK delle versioni seguenti o successive:

| Runtime          | Versione SDK minima richiesta AWS |
|------------------|-----------------------------------|
| Node.js          | 2.1147.0 (SDK versione 2)         |
|                  | 3.105.0 (SDK versione 3)          |
| Python           | 1.24.46 (boto3)                   |
|                  | 1.27.46 (botocore)                |
| Java 8 e Java 11 | 2.17.135                          |
| Java 17          | 2,20,81                           |
| Java 21          | 2,21,24                           |
| .NET             | 3,7293,0                          |
| Ruby             | 3,134,0                           |
| PHP              | 3,232,0                           |

| Runtime | Versione SDK minima richiesta AWS |
|---------|-----------------------------------|
| Go      | V2 SDK 1.57.0                     |

Alcuni runtime Lambda come Python includono una versione dell'SDK. Node.js AWS Se la versione SDK inclusa nel runtime della funzione è inferiore al minimo richiesto, puoi aggiungere una versione supportata dell'SDK al pacchetto di implementazione della funzione. Puoi aggiungere una versione supportata dell'SDK alla funzione anche utilizzando un [livello Lambda](#). Per un elenco degli SDK inclusi in ogni runtime Lambda, consulta la pagina [Runtime Lambda](#).

## Notifiche dei cicli ricorsivi

Quando Lambda interrompe un ciclo ricorsivo, ricevi notifiche tramite [Dashboard Health](#) e tramite e-mail. Puoi anche utilizzare le CloudWatch metriche per monitorare il numero di invocazioni ricorsive che Lambda ha interrotto.

### Dashboard Health notifiche

Quando Lambda interrompe un'invocazione ricorsiva, Dashboard Health visualizza una notifica nella pagina sullo stato del tuo account, in Problemi aperti e recenti. <https://health.aws.amazon.com/health/home#/account/dashboard/open-issues> Tieni presente che possono essere necessarie fino a 3,5 ore dopo che Lambda interrompe un'invocazione ricorsiva prima che questa notifica venga visualizzata. Per ulteriori informazioni sulla visualizzazione degli eventi dell'account in Dashboard Health, consulta [Getting started with your AWS Health Dashboard — Your account health](#) nella Health User Guide. AWS

### Avvisi via e-mail

Quando Lambda interrompe per la prima volta un'invocazione ricorsiva della funzione, ti invia un avviso e-mail. Lambda invia un massimo di un'e-mail ogni 24 ore per ogni funzione del tuo Account AWS. Dopo l'invio di una notifica e-mail da parte di Lambda, non riceverai altre e-mail per la stessa funzione per altre 24 ore, anche se Lambda interrompe ulteriori invocazioni ricorsive della funzione. Tieni presente che possono essere necessarie fino a 3,5 ore dopo che Lambda interrompe un'invocazione ricorsiva prima di ricevere questo avviso via email.

Lambda invia avvisi e-mail in loop ricorsivo al contatto principale dell'account e al contatto operativo alternativo dell'utente Account AWS. Per informazioni sulla visualizzazione o l'aggiornamento degli indirizzi e-mail nel tuo account, consulta la pagina [Aggiornamento delle informazioni di contatto](#) della Guida generale di AWS .

## Metriche Amazon CloudWatch

La [CloudWatch metrica](#) `RecursiveInvocationsDropped` registra il numero di chiamate di funzioni che Lambda ha interrotto perché la tua funzione è stata richiamata più di circa 16 volte in una singola catena di richieste. Lambda emette questo parametro non appena interrompe un'invocazione ricorsiva. Per visualizzare questa metrica, segui le istruzioni per la [visualizzazione delle metriche sulla console e scegli la metrica](#). `CloudWatch RecursiveInvocationsDropped`

## Risposta alle notifiche di rilevamento dei cicli ricorsivi

Quando la funzione viene richiamata più di 16 volte dallo stesso evento di attivazione, Lambda interrompe la successiva invocazione della funzione per quell'evento per interrompere il ciclo ricorsivo. Per evitare il ripetersi di un ciclo ricorsivo interrotto da Lambda, procedi come segue:

- Riduci la [simultaneità](#) disponibile della funzione a zero, il che limita tutte le invocazioni future.
- Rimuovi o disabilita il trigger o la mappatura dell'origine degli eventi che richiama la tua funzione.
- Identifica e correggi i difetti del codice che reindirizzano gli eventi alla AWS risorsa che richiama la tua funzione. Una fonte comune di difetti è l'utilizzo di variabili per definire l'origine e la destinazione degli eventi di una funzione. Verifica di non utilizzare lo stesso valore per entrambe le variabili.

Inoltre, se l'origine degli eventi della funzione Lambda è una coda Amazon SQS, valuta la possibilità di [configurare una coda DLQ](#) nella coda di origine.

### Note

Assicurati di configurare la coda DLQ sulla coda di origine, non sulla funzione Lambda. La coda DLQ che si configura su una funzione viene utilizzata per la [coda di invocazione asincrona](#) della funzione, non per le code di origine eventi.

Se l'origine degli eventi è un argomento di Amazon SNS, valuta la possibilità di aggiungere una [destinazione in caso di errore](#) per la funzione.

Azzeramento della simultaneità disponibile per la funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione .
3. Scegli Limitatore.

4. Nella finestra di dialogo Limita la tua funzione, scegli Conferma.

Rimozione di un trigger o di una mappatura dell'origine degli eventi per la funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione .
3. Scegli la scheda Configurazione, quindi scegli Trigger.
4. In Trigger, seleziona il trigger o la mappatura dell'origine degli eventi che desideri eliminare, quindi scegli Elimina.
5. Nella finestra di dialogo Elimina trigger, scegli Elimina.

Per disattivare una mappatura della sorgente di eventi per la tua funzione (AWS CLI)

1. Per trovare l'UUID per la mappatura della sorgente di eventi che desideri disabilitare, esegui il comando AWS Command Line Interface ()AWS CLI list-event-source-mappings. <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/list-event-source-mappings.html>

```
aws lambda list-event-source-mappings
```

2. Per disabilitare la mappatura della sorgente degli eventi, esegui il seguente comando update-event-source-mapping. AWS CLI <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/update-event-source-mapping.html>

```
aws lambda update-event-source-mapping --function-name MyFunction \  
--uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 --no-enabled
```

## Consentire l'esecuzione di una funzione Lambda in un ciclo ricorsivo

Se il tuo progetto utilizza intenzionalmente un ciclo ricorsivo, puoi modificare la configurazione predefinita di una funzione Lambda per consentirne l'invocazione ricorsiva. È preferibile evitare l'uso di cicli ricorsivi nel codice. Gli errori di implementazione possono portare a richiami ricorsivi che utilizzano tutta la concorrenza disponibile e alla fatturazione di addebiti imprevisti sul tuo account.

Account AWS

### ⚠ Important

Se utilizzi cicli ricorsivi, trattali con attenzione. Implementa i guardrail basati sulle best practice per ridurre al minimo i rischi di errori di implementazione. Per ulteriori informazioni sulle best practice per l'utilizzo di modelli ricorsivi, consulta la pagina [Schemi ricorsivi che causano funzioni Lambda indeterminate](#) in Serverless Land.

Puoi configurare le funzioni per consentire i loop ricorsivi utilizzando la console Lambda, la () e l'API. AWS Command Line Interface AWS CLI [PutFunctionRecursionConfig](#) Puoi anche configurare l'impostazione di rilevamento del ciclo ricorsivo di una funzione in e. AWS SAM CloudFormation

Per impostazione predefinita, Lambda rileva e termina i cicli ricorsivi. A meno che il tuo progetto non utilizzi intenzionalmente un ciclo ricorsivo, ti consigliamo di non modificare la configurazione predefinita delle funzioni.

Nota che quando configuri una funzione per consentire i loop ricorsivi, la metrica non viene [CloudWatch emessa](#). RecursiveInvocationsDropped

## Console

Consentire l'esecuzione di una funzione Lambda in un ciclo ricorsivo (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione per aprire la pagina dei dettagli della funzione.
3. Scegli la scheda Configurazione, quindi scegli Rilevamento di simultaneità e ricorsione.
4. Oltre al rilevamento di cicli ricorsivi, scegli Modifica.
5. Seleziona Consenti cicli ricorsivi.
6. Selezionare Salva.

## AWS CLI

Puoi utilizzare l'[PutFunctionRecursionConfig](#) API per consentire l'invocazione della tua funzione in un ciclo ricorsivo. Specifica Allow per il parametro del ciclo ricorsivo. Ad esempio, puoi chiamare questa API con il comando: put-function-recursion-config AWS CLI

```
aws lambda put-function-recursion-config --function-name yourFunctionName --recursive-loop Allow
```

Puoi riportare la configurazione della funzione all'impostazione predefinita in modo che Lambda termini i cicli ricorsivi quando li rileva. Modifica la configurazione della tua funzione utilizzando la console Lambda o la AWS CLI.

## Console

Per configurare una funzione in modo che i cicli ricorsivi vengano terminati (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione per aprire la pagina dei dettagli della funzione.
3. Scegli la scheda Configurazione, quindi scegli Rilevamento di simultaneità e ricorsione.
4. Oltre al rilevamento di cicli ricorsivi, scegli Modifica.
5. Seleziona Termina i cicli ricorsivi.
6. Selezionare Salva.

## AWS CLI

Puoi utilizzare l'[PutFunctionRecursionConfig](#) API per configurare la tua funzione in modo che Lambda termini i loop ricorsivi quando li rileva. Specifica `Terminate` per il parametro del ciclo ricorsivo. Ad esempio, puoi chiamare questa API con il comando: `put-function-recursion-config` AWS CLI

```
aws lambda put-function-recursion-config --function-name yourFunctionName --recursive-loop Terminate
```

## Regioni supportate per il rilevamento dei loop ricorsivi Lambda

Il rilevamento dei loop ricorsivi Lambda è supportato in tutte le regioni commerciali. <https://docs.aws.amazon.com/general/latest/gr/glos-chap.html#region>



## Creazione e gestione degli URL della funzione Lambda

Un URL della funzione è un endpoint HTTP(S) dedicato alla funzione Lambda. È possibile creare e configurare un URL della funzione tramite la console Lambda o l'API Lambda.

### Tip

Lambda offre due modi per richiamare la tua funzione tramite un endpoint HTTP: gli URL delle funzioni e Amazon API Gateway. In caso di dubbi su quale sia il metodo migliore per il tuo caso d'uso, consulta [the section called “URL delle funzioni e Amazon API Gateway”](#).

Quando si crea un URL della funzione, Lambda genera automaticamente un endpoint URL univoco. Dopo aver creato un URL della funzione, il suo endpoint URL non cambia mai. Gli endpoint URL della funzione hanno il formato seguente:

```
https://<url-id>.lambda-url.<region>.on.aws
```

### Note

Gli URL delle funzioni non sono supportati in tutte le regioni. AWS Per verificare la disponibilità, consulta Servizi [AWS regionali](#) e filtra per Feature Name = Function URLs.

Gli URL della funzione sono abilitati alla rete dual stack e supportano IPv4 e IPv6. Dopo aver configurato l'URL di una funzione per la tua funzione, puoi richiamarla tramite il relativo endpoint HTTP (S) utilizzando un browser web, curl, Postman o qualsiasi client HTTP.

### Note

Puoi accedere all'URL della funzione solo tramite Internet pubblico. Sebbene le funzioni Lambda supportino AWS PrivateLink, gli URL delle funzioni no.

Gli URL della funzione Lambda utilizzano [policy basate sulle risorse](#) per sicurezza e controllo degli accessi. Gli URL della funzione supportano anche le opzioni di configurazione CORS (cross-origin resource sharing).

È possibile applicare gli URL della funzione a qualsiasi alias di funzione o alla versione della funzione non pubblicata di \$LATEST. Non è possibile aggiungere un URL della funzione a nessun'altra versione della funzione.

La sezione seguente mostra come creare e gestire l'URL di una funzione utilizzando la console e il modello Lambda. AWS CLI CloudFormation Per un tutorial dettagliato sull'utilizzo degli URL delle funzioni come endpoint del webhook, consulta. [the section called “Tutorial: Creazione di un endpoint webhook”](#)

## Argomenti

- [Creazione di un URL della funzione \(console\)](#)
- [Creazione dell'URL di una funzione \(AWS CLI\)](#)
- [Aggiungere l'URL di una funzione a un CloudFormation modello](#)
- [Cross-origin condivisione delle risorse \(CORS\)](#)
- [Throttling di URL di funzioni](#)
- [Disattivazione degli URL della funzione](#)
- [Eliminazione degli URL della funzione](#)
- [Controllo dell'accesso agli URL della funzione Lambda](#)
- [Richiamo di URL di funzioni Lambda](#)
- [Monitoraggio degli URL della funzione Lambda](#)
- [Selezionare un metodo per richiamare la funzione Lambda tramite una richiesta HTTP](#)
- [Tutorial: creazione di un endpoint webhook utilizzando l'URL di una funzione Lambda](#)

## Creazione di un URL della funzione (console)

Segui queste fasi per creare un URL della funzione usando la console.

Per creare l'URL di una funzione esistente

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione per la quale desideri creare l'URL della funzione.
3. Scegli la scheda Configurazione, quindi scegli URL della funzione.
4. Scegli Crea URL della funzione.
5. Per Tipo di autenticazione, scegli AWS\_IAM o NONE. Per ulteriori informazioni sull'autenticazione dell'URL della funzione, consulta [Controllo accessi](#).

6. (Opzionale) Seleziona Configura CORS e quindi configura le impostazioni CORS per l'URL della funzione. Per ulteriori informazioni sulla funzionalità CORS, consulta [Cross-origin condivisione delle risorse \(CORS\)](#).
7. Scegli Save (Salva).

In questo modo viene creato un URL di funzione per la versione non pubblicata di \$LATEST della funzione. L'URL della funzione viene visualizzato nella sezione Panoramica della funzione della console.

Per creare l'URL di una funzione per un alias esistente

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione con l'alias per cui desideri creare l'URL della funzione.
3. Seleziona la scheda Aliases (Alias) e quindi scegli il nome dell'alias per cui desideri creare l'URL della funzione.
4. Scegli la scheda Configuration (Configurazione), quindi scegli Function URL (URL della funzione).
5. Scegli Crea URL della funzione.
6. Per Tipo di autenticazione, scegli AWS\_IAM o NONE. Per ulteriori informazioni sull'autenticazione dell'URL della funzione, consulta [Controllo accessi](#).
7. (Opzionale) Seleziona Configura CORS e quindi configura le impostazioni CORS per l'URL della funzione. Per ulteriori informazioni sulla funzionalità CORS, consulta [Cross-origin condivisione delle risorse \(CORS\)](#).
8. Scegli Save (Salva).

In questo modo viene creato un URL della funzione per l'alias della funzione. L'URL della funzione viene visualizzato nella sezione Panoramica della funzione della console per l'alias.

Per creare una nuova funzione con l'URL di una funzione

Creazione di una nuova funzione con un URL della funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli Crea funzione.
3. In Basic information (Informazioni di base) eseguire queste operazioni:

- a. In Function name (Nome funzione), inserisci un nome per la funzione, ad esempio **my-function**.
  - b. Per Runtime, scegli la lingua di runtime che preferisci, ad esempio Node.js 24.
  - c. Per Architecture (Architettura), scegli x86\_64 o arm64.
  - d. Espandi Permissions (Autorizzazioni), quindi scegli se creare un nuovo ruolo di esecuzione o usarne uno esistente.
4. Espandi Advanced settings (Impostazioni avanzate) e quindi seleziona Function URL (URL della funzione).
  5. Per Auth type (Tipo di autenticazione), scegli AWS\_IAM o NONE. Per ulteriori informazioni sull'autenticazione dell'URL della funzione, consulta [Controllo accessi](#).
  6. (Opzionale) Seleziona Configure cross-origin resource sharing (CORS) (Configura CORS). Selezionando questa opzione durante la creazione della funzione, l'URL della funzione consente le richieste da tutte le origini per impostazione predefinita. È possibile modificare le impostazioni CORS per l'URL della funzione dopo aver creato la funzione. Per ulteriori informazioni sulla funzionalità CORS, consulta [Cross-origin condivisione delle risorse \(CORS\)](#).
  7. Scegli Crea funzione.

In questo modo viene creata una nuova funzione con un URL della funzione per la versione non pubblicata di \$LATEST della funzione. L'URL della funzione viene visualizzato nella sezione Function overview (Panoramica della funzione) della console.

## Creazione dell'URL di una funzione (AWS CLI)

Per creare un URL di funzione per una funzione Lambda esistente utilizzando AWS Command Line Interface (AWS CLI), esegui il seguente comando:

```
aws lambda create-function-url-config \  
  --function-name my-function \  
  --qualifier prod \ // optional  
  --auth-type AWS_IAM  
  --cors-config {AllowOrigins="https://example.com"} // optional
```

Un URL di funzione viene aggiunto al qualificatore **prod** per la funzione **my-function**. Per ulteriori informazioni su questi parametri di configurazione, consulta [CreateFunctionUrlConfig](#) nella documentazione di riferimento delle API.

 Note

Per creare l'URL di una funzione utilizzando il AWS CLI, la funzione deve già esistere.

## Aggiungere l'URL di una funzione a un CloudFormation modello

Per aggiungere una `AWS::Lambda::Url` risorsa al CloudFormation modello, utilizzate la seguente sintassi:

### JSON

```
{
  "Type" : "AWS::Lambda::Url",
  "Properties" : {
    "AuthType" : String,
    "Cors" : Cors,
    "Qualifier" : String,
    "TargetFunctionArn" : String
  }
}
```

### YAML

```
Type: AWS::Lambda::Url
Properties:
  AuthType: String
  Cors:
    Cors
  Qualifier: String
  TargetFunctionArn: String
```

### Parameters

- (Obbligatorio)`AuthType`: definisce il tipo di autenticazione per l'URL della funzione. I valori possibili sono `AWS_IAM` o `NONE`. Per limitare l'accesso solo agli utenti autenticati, imposta su `AWS_IAM`. Per ignorare l'autenticazione IAM e consentire a qualsiasi utente di effettuare richieste alla propria funzione, imposta su `NONE`.
- (Opzionale)`Cors`: definisce le [impostazioni CORS](#) per l'URL della funzione. Per aggiungere Cors alla tua `AWS::Lambda::Url` risorsa in CloudFormation, usa la seguente sintassi.

## Example AWS::Lambda:: (JSON) Url.Cors

```
{
  "AllowCredentials" : Boolean,
  "AllowHeaders" : [ String, ... ],
  "AllowMethods" : [ String, ... ],
  "AllowOrigins" : [ String, ... ],
  "ExposeHeaders" : [ String, ... ],
  "MaxAge" : Integer
}
```

## Example AWS::Lambda:: (YAML) Url.Cors

```
AllowCredentials: Boolean
AllowHeaders:
  - String
AllowMethods:
  - String
AllowOrigins:
  - String
ExposeHeaders:
  - String
MaxAge: Integer
```

- (Opzionale) `Qualifier`: il nome dell'alias.
- (Obbligatorio) `TargetFunctionArn`: il nome o l'Amazon Resource Name (ARN) della funzione Lambda. I formati dei nomi validi includono quanto segue:
  - Nome funzione – `my-function`
  - ARN funzione – `arn:aws:lambda:us-west-2:123456789012:function:my-function`
  - ARN parziale – `123456789012:function:my-function`.

## Cross-origin condivisione delle risorse (CORS)

Per definire in che modo le diverse origini possono accedere all'URL della funzione, utilizza [cross-origin resource sharing \(CORS\)](#). Si consiglia di configurare CORS se si intende chiamare l'URL della funzione da un dominio diverso. Lambda supporta le seguenti intestazioni CORS per gli URL della funzione.

| Intestazioni CORS                                | Proprietà di configurazione CORS | Valori di esempio                                                                          |
|--------------------------------------------------|----------------------------------|--------------------------------------------------------------------------------------------|
| <a href="#">Access-Control-Allow-Origin</a>      | AllowOrigins                     | * (consente tutte le origini)<br><br>https://www.example.com<br><br>http://localhost:60905 |
| <a href="#">Access-Control-Allow-Methods</a>     | AllowMethods                     | GET, POST, DELETE, *                                                                       |
| <a href="#">Access-Control-Allow-Headers</a>     | AllowHeaders                     | Date, Keep-Alive , X-Custom-Header                                                         |
| <a href="#">Access-Control-Expose-Headers</a>    | ExposeHeaders                    | Date, Keep-Alive , X-Custom-Header                                                         |
| <a href="#">Access-Control-Allow-Credentials</a> | AllowCredentials                 | TRUE                                                                                       |
| <a href="#">Access-Control-Max-Age</a>           | MaxAge                           | 5 (default), 300                                                                           |

Quando si configura CORS per l'URL di una funzione utilizzando la console Lambda o il AWS CLI, Lambda aggiunge automaticamente le intestazioni CORS a tutte le risposte tramite l'URL della funzione. In alternativa, è possibile aggiungere manualmente le intestazioni CORS alla risposta della funzione. Se ci sono intestazioni in conflitto, il comportamento previsto dipende dal tipo di richiesta:

- Per le richieste di preflight come le richieste OPTIONS, le intestazioni CORS configurate nella funzione URL hanno la precedenza. Lambda restituisce solo queste intestazioni CORS nella risposta.
- Per le richieste non preflight come le richieste GET o POST, Lambda restituisce sia le intestazioni CORS configurate nella funzione URL, sia le intestazioni CORS restituite dalla funzione. Ciò può comportare intestazioni CORS duplicate nella risposta. È possibile che venga visualizzato un errore simile al seguente: The 'Access-Control-Allow-Origin' header contains multiple values '\*, \*', but only one is allowed.

In generale, consigliamo di configurare tutte le impostazioni CORS nella funzione URL anziché inviare manualmente le intestazioni CORS nella risposta della funzione.

## Throttling di URL di funzioni

Il throttling limita la frequenza con cui la funzione elabora le richieste. Ciò è utile in molte situazioni, ad esempio per impedire alla funzione di sovraccaricare le risorse a valle o per gestire un improvviso aumento delle richieste.

È possibile limitare la frequenza delle richieste elaborate dalla funzione Lambda tramite un URL di funzione configurando la simultaneità riservata. La simultaneità riservata limita il numero massimo di richiami simultanei della funzione. La frequenza massima di richieste al secondo (RPS) della funzione equivale a 10 volte la simultaneità riservata configurata. Ad esempio, se si configura la funzione con una simultaneità riservata di 100, l'RPS massimo è 1.000.

Ogni volta che la simultaneità della funzione supera la simultaneità riservata, l'URL della funzione restituisce un codice di stato HTTP 429. Se la funzione riceve una richiesta che supera 10 volte l'RPS massimo in base alla simultaneità riservata configurata, viene ricevuto anche un errore HTTP 429. Per ulteriori informazioni sulla simultaneità riservata, consulta [Configurazione della simultaneità riservata per una funzione](#).

## Disattivazione degli URL della funzione

In caso di emergenza, potresti voler rifiutare tutto il traffico verso l'URL della funzione. Per disattivare l'URL della funzione, imposta la simultaneità riservata su zero. In questo modo vengono limitate tutte le richieste all'URL della funzione, con conseguenti risposte di stato HTTP 429. Per riattivare l'URL della funzione, elimina la configurazione di simultaneità riservata o imposta la configurazione su una quantità maggiore di zero.

## Eliminazione degli URL della funzione

Quando si elimina un URL di funzione, non è possibile ripristinarlo. La creazione di un nuovo URL di funzione determina un indirizzo URL diverso.

### Note

Se elimini l'URL della funzione con il tipo di autenticazione NONE, Lambda non elimina automaticamente la policy basata sulle risorse associata. Se desideri eliminare questa policy, dovrai farlo manualmente.



1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere il nome della funzione.
3. Scegli la scheda Configurazione, quindi scegli URL della funzione.
4. Scegli Elimina.
5. Inserisci la parola delete (elimina) nel campo per confermare l'eliminazione.
6. Scegli Elimina.

#### Note

Quando si elimina una funzione con una funzione URL, Lambda elimina in modo asincrono la funzione URL. Se si crea immediatamente una nuova funzione con lo stesso nome nello stesso account, è possibile che l'URL della funzione originale venga mappato alla nuova funzione anziché eliminato.

## Controllo dell'accesso agli URL della funzione Lambda

#### Note

A partire da ottobre 2025, gli URL delle nuove funzioni richiederanno entrambe `lambda:InvokeFunctionUrl` e `lambda:InvokeFunction` autorizzazioni.

Puoi controllare l'accesso agli URL della funzione Lambda utilizzando il parametro [AuthType](#) combinato con [policy basate sulle risorse](#) collegate alla funzione specifica. La configurazione di questi due componenti determina chi può richiamare o eseguire altre azioni amministrative sull'URL della funzione.

Il parametro `AuthType` determina il modo in cui Lambda autentica o autorizza le richieste all'URL della funzione. Quando configuri l'URL della funzione, è necessario specificare una delle seguenti opzioni `AuthType`:

- **AWS\_IAM**— Lambda utilizza AWS Identity and Access Management (IAM) per autenticare e autorizzare le richieste in base alla politica di identità del responsabile IAM e alla politica basata sulle risorse della funzione. Scegli questa opzione se desideri che solo utenti e ruoli autenticati invochino la tua funzione utilizzando l'URL della funzione.

- **NONE:** Lambda non esegue alcuna autenticazione prima di richiamare la funzione. Tuttavia, la policy basata sulle risorse della funzione è sempre valida e deve concedere l'accesso pubblico prima che l'URL della funzione possa ricevere richieste. Scegli questa opzione per consentire l'accesso pubblico e non autenticato all'URL della funzione.

Per ulteriori informazioni sulla sicurezza, puoi utilizzarla AWS Identity and Access Management Access Analyzer per ottenere un'analisi completa dell'accesso esterno all'URL della tua funzione. IAM Access Analyzer controlla inoltre le autorizzazioni nuove o aggiornate sulle funzioni Lambda per aiutarti a identificare le autorizzazioni che garantiscono l'accesso pubblico e tra account. Puoi utilizzare IAM Access Analyzer gratuitamente. Per le nozioni di base su IAM Access Analyzer, consulta [Utilizzo di AWS IAM Access Analyzer](#).

Questa pagina contiene esempi di policy basate sulle risorse per entrambi i tipi di autenticazione e come crearle utilizzando l'operazione [AddPermission](#) API o la console Lambda. Per informazioni su come richiamare l'URL della funzione dopo aver impostato le autorizzazioni, consulta [Richiamo di URL di funzioni Lambda](#)

## Argomenti

- [Utilizzo del tipo di autenticazione AWS\\_IAM](#)
- [Utilizzo del tipo di autenticazione NONE](#)
- [Governance e controllo degli accessi](#)

## Utilizzo del tipo di autenticazione **AWS\_IAM**

Se scegli il tipo di autenticazione AWS\_IAM, gli utenti che hanno bisogno di richiamare l'URL della funzione Lambda devono avere le autorizzazioni `lambda:InvokeFunctionUrl` e `lambda:InvokeFunction`. A seconda di chi effettua la richiesta di invocazione, potrebbe essere necessario concedere questa autorizzazione utilizzando una [policy basata sulle risorse](#).

Se il principale che effettua la richiesta è lo Account AWS stesso dell'URL della funzione, deve disporre delle `lambda:InvokeFunction` autorizzazioni nella propria politica basata sull'[identità o avere lambda:InvokeFunctionUrl le autorizzazioni concesse nella politica](#) basata sulle risorse della funzione. In altre parole, una policy basata sulle risorse è facoltativa se l'utente ha già autorizzazioni `lambda:InvokeFunctionUrl` e `lambda:InvokeFunction` nella policy basata sull'identità. La valutazione delle politiche segue le regole delineate nella logica di valutazione delle politiche. [https://docs.aws.amazon.com/IAM/latest/UserGuide/reference\\_policies\\_evaluation-logic.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html)

Se il principal che effettua la richiesta si trova in un account diverso, il principal deve avere sia una policy basata sull'identità che fornisce autorizzazioni `lambda:InvokeFunctionUrl` e `lambda:InvokeFunction`, sia autorizzazioni concesse in una policy basata sulle risorse nella funzione che si sta tentando di invocare. La valutazione delle politiche segue le regole delineate in [Determinare se è consentita una richiesta tra più account](#).

La seguente politica basata sulle risorse consente al `example` ruolo in di richiamare l'URL della funzione associato Account AWS 444455556666 alla funzione. `my-function` Il tasto [lambda:InvokedViaFunctionUrl](#) context limita l'azione alle chiamate URL della `lambda:InvokeFunction` funzione. Ciò significa che il principale deve utilizzare l'URL della funzione per richiamare la funzione. Se non includi `lambda:InvokedViaFunctionUrl`, il principale può richiamare la tua funzione tramite altri metodi di richiamo, oltre all'URL della funzione.

Example— politica basata sulle risorse Cross-account

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::444455556666:role/example"
      },
      "Action": "lambda:InvokeFunctionUrl",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
      "Condition": {
        "StringEquals": {
          "lambda:FunctionUrlAuthType": "AWS_IAM"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::444455556666:role/example"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
      "Condition": {
```

```
    "Bool": {
      "lambda:InvokedViaFunctionUrl": "true"
    }
  }
}
```

È possibile creare questa politica basata sulle risorse tramite la console utilizzando i seguenti passaggi:

Per concedere le autorizzazioni di richiamo URL a un altro account (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione per la quale desideri concedere autorizzazioni di richiamo URL.
3. Quindi, seleziona la scheda Configuration (Configurazione) e poi Permissions (Autorizzazioni).
4. In Resource-based policy, scegli Aggiungi autorizzazioni.
5. Scegli Function URL (URL funzione).
6. Per Auth type (Tipo di autenticazione), scegli AWS\_IAM.
7. Inserisci un ID di dichiarazione per la tua dichiarazione sulla polizza.
8. Per Principal, inserisci l'ID dell'account o l'Amazon Resource Name (ARN) dell'utente o del ruolo a cui desideri concedere le autorizzazioni. Ad esempio: **444455556666**.
9. Scegli Save (Salva).

In alternativa, puoi creare questa policy utilizzando i seguenti comandi [add-permission](#) AWS Command Line Interface (AWS CLI). Quando si utilizza il AWS CLI, è necessario aggiungere le `lambda:InvokeFunction` istruzioni `lambda:InvokeFunctionUrl` and separatamente. Ad esempio:

```
aws lambda add-permission --function-name my-function \  
  --statement-id UrlPolicyInvokeURL \  
  --action lambda:InvokeFunctionUrl \  
  --principal 444455556666 \  
  --function-url-auth-type AWS_IAM
```

```
aws lambda add-permission --function-name my-function \  
  --statement-id UrlPolicyInvokeFunction \  
  --action Lambda:InvokeFunction \  
  --principal 444455556666 \  
  --invoked-via-function-url
```

## Utilizzo del tipo di **autenticazione** NONE

### Important

Quando il tipo di autenticazione dell'URL della funzione è NONE e hai una [policy basata sulle risorse](#) che garantisce l'accesso pubblico, qualsiasi utente non autenticato con l'URL della funzione può invocare la funzione.

In alcuni casi, potresti volere che l'URL della tua funzione sia pubblico. Ad esempio, potrebbe essere preferibile inviare le richieste effettuate direttamente da un browser Web. Per consentire l'accesso pubblico all'URL della funzione, scegli il tipo di autenticazione NONE.

Se scegli il tipo di autenticazione NONE, Lambda non utilizza IAM per autenticare le richieste all'URL della funzione. Tuttavia, la funzione deve disporre di una politica basata sulle risorse che consenta `lambda:InvokeFunctionUrl` e `lambda:InvokeFunction`. Quando crei un URL di funzione con tipo di autenticazione NONE utilizzando la console o AWS Serverless Application Model (AWS SAM), Lambda crea automaticamente la policy basata sulle risorse per te. Se stai utilizzando direttamente la AWS CLI, AWS CloudFormation, o l'API Lambda, devi aggiungere tu stesso la policy. ???

Ti consigliamo di includere la chiave `lambda: InvokedViaFunctionUrl` context nelle tue policy basate sulle risorse quando usi il tipo di autenticazione. NONE Questa chiave di contesto garantisce che la funzione possa essere richiamata solo tramite l'URL della funzione e non tramite altri metodi di richiamo.

Nota quanto segue su questa politica:

- Tutte le entità possono chiamare `lambda:InvokeFunctionUrl` e `lambda:InvokeFunction`. Ciò significa che chiunque abbia l'URL della tua funzione può richiamare la tua funzione.
- Il valore della chiave della `lambda:FunctionUrlAuthType` condizione è NONE. Ciò significa che la dichiarazione della politica consente l'accesso solo se lo è anche NONE il tipo di autenticazione dell'URL della funzione.

- La `lambda:InvokeFunctionUrl` condizione garantisce che la funzione possa essere richiamata solo tramite l'URL della funzione e non tramite altri metodi di richiamo.

Example— Politica predefinita basata sulle risorse per il tipo di autenticazione NONE

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "FunctionURLAllowPublicAccess",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "lambda:InvokeFunctionUrl",
      "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
      "Condition": {
        "StringEquals": {
          "lambda:FunctionUrlAuthType": "NONE"
        }
      }
    },
    {
      "Sid": "FunctionURLInvokeAllowPublicAccess",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
      "Condition": {
        "Bool": {
          "lambda:InvokedViaFunctionUrl": "true"
        }
      }
    }
  ]
}
```

Crea la politica basata sulle risorse utilizzando AWS CLI

A meno che non utilizzi la console o non AWS SAM crei un URL di funzione con tipo di autenticazione `NONE`, devi aggiungere tu stesso la policy basata sulle risorse. Usa i comandi seguenti per creare istruzioni con le autorizzazioni `lambda:InvokeFunctionUrl` e `lambda:InvokeFunction`. Ogni istruzione deve essere aggiunta in un comando separato.

```
aws lambda add-permission \  
  --function-name UrlTestFunction \  
  --statement-id UrlPolicyInvokeURL \  
  --action lambda:InvokeFunctionUrl \  
  --principal * \  
  --function-url-auth-type NONE
```

```
aws lambda add-permission \  
  --function-name UrlTestFunction \  
  --statement-id UrlPolicyInvokeFunction \  
  --action lambda:InvokeFunction \  
  --principal * \  
  --invoked-via-function-url
```

#### Note

Se elimini l'URL della funzione con il tipo di autenticazione `NONE`, Lambda non elimina automaticamente la policy basata sulle risorse associata. Se desideri eliminare questa policy, dovrai farlo manualmente.

Se la policy basata sulle risorse di una funzione non concede `lambda:invokeFunctionUrl` `lambda:InvokeFunction` autorizzazioni, gli utenti ricevono un codice di errore 403 Forbidden quando tentano di richiamare l'URL della funzione. Ciò si verifica anche se l'URL della funzione utilizza il tipo di autenticazione. `NONE`

## Governance e controllo degli accessi

Oltre alle autorizzazioni di richiamo degli URL della funzione, puoi anche controllare l'accesso alle azioni utilizzate per configurare gli URL della funzione. Lambda supporta le seguenti azioni delle policy IAM per gli URL della funzione:

- `lambda:InvokeFunctionUrl`: richiamo di una funzione Lambda utilizzando l'URL della funzione.

- `lambda:CreateFunctionUrlConfig`: creazione di un URL della funzione e impostazione del relativo `AuthType`.
- `lambda:UpdateFunctionUrlConfig`: aggiornamento della configurazione dell'URL della funzione e del relativo `AuthType`.
- `lambda:GetFunctionUrlConfig`: visualizzazione dei dettagli dell'URL della funzione.
- `lambda:ListFunctionUrlConfigs`: elenco delle configurazioni dell'URL della funzione.
- `lambda>DeleteFunctionUrlConfig`: eliminazione dell'URL della funzione.

Per consentire o negare l'accesso all'URL della funzione ad altre AWS entità, includi queste azioni nelle policy IAM. Ad esempio, la seguente policy concede il `example` ruolo nelle Account AWS 444455556666 autorizzazioni per aggiornare l'URL della funzione per la funzione nell'account. **my-function** 123456789012

Example Policy dell'URL della funzione tra account

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::444455556666:role/example"
      },
      "Action": "lambda:UpdateFunctionUrlConfig",
      "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-  
function"
    }
  ]
}
```

Chiavi di condizione

Per un controllo granulare dell'accesso agli URL delle funzioni, utilizza le chiavi di contesto delle condizioni. Lambda supporta i seguenti tasti di contesto per gli URL delle funzioni:



- `lambda:FunctionUrlAuthType`: definisce un valore enum che descrive il tipo di autenticazione utilizzato dall'URL della funzione. Il valore può essere `AWS_IAM` o `NONE`.
- `lambda:InvokedViaFunctionUrl`— Limita l'`lambda:InvokeFunction` alle chiamate effettuate tramite l'URL della funzione. Ciò garantisce che la funzione possa essere richiamata solo utilizzando l'URL della funzione e non tramite altri metodi di richiamo. Per esempi di politiche basate sulle risorse che utilizzano la chiave di `lambda:InvokedViaFunctionUrl` contesto, consulta gli esempi in e. [Utilizzo del tipo di autenticazione AWS\\_IAM](#) [Utilizzo del tipo di autenticazione NONE](#)

Puoi utilizzare queste chiavi di contesto nelle politiche associate alla tua funzione. Ad esempio, è preferibile limitare chi può apportare modifiche alla configurazione degli URL della funzione. Per negare tutte le richieste `UpdateFunctionUrlConfig` a qualsiasi funzione con tipo di autenticazione URL `NONE`, è possibile definire la seguente policy:

Example Policy dell'URL della funzione con rifiuto esplicito

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": [
        "lambda:UpdateFunctionUrlConfig"
      ],
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:*",
      "Condition": {
        "StringEquals": {
          "lambda:FunctionUrlAuthType": "NONE"
        }
      }
    }
  ]
}
```

Per concedere il `example` ruolo nelle Account AWS 444455556666 autorizzazioni da effettuare `CreateFunctionUrlConfig` e nelle `UpdateFunctionUrlConfig` richieste sulle funzioni con tipo di autenticazione `URLAWS_IAM`, puoi definire la seguente politica:

Example Policy dell'URL della funzione con permesso esplicito

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::444455556666:role/example"
      },
      "Action": [
        "lambda:CreateFunctionUrlConfig",
        "lambda:UpdateFunctionUrlConfig"
      ],
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:*",
      "Condition": {
        "StringEquals": {
          "lambda:FunctionUrlAuthType": "AWS_IAM"
        }
      }
    }
  ]
}
```

È inoltre possibile utilizzare questa chiave di condizione in una [policy di controllo dei servizi](#) (SCP). Utilizzare SCP per gestire le autorizzazioni di un'intera organizzazione in AWS Organizations. Ad esempio, per negare agli utenti di creare o aggiornare gli URL della funzione che utilizzano qualcosa di diverso dal tipo di autenticazione `AWS_IAM`, utilizzare i seguenti criteri di controllo dei servizi:

Example SCP dell'URL della funzione con rifiuto esplicito

JSON

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Deny",
    "Action": [
      "lambda:CreateFunctionUrlConfig",
      "lambda:UpdateFunctionUrlConfig"
    ],
    "Resource": "arn:aws:lambda*:123456789012:function:*",
    "Condition": {
      "StringNotEquals": {
        "lambda:FunctionUrlAuthType": "AWS_IAM"
      }
    }
  }
]
}
```

## Richiamo di URL di funzioni Lambda

Un URL della funzione è un endpoint HTTP(S) dedicato alla funzione Lambda. È possibile creare e configurare un URL della funzione tramite la console Lambda o l'API Lambda.

### Tip

Lambda offre due modi per richiamare la tua funzione tramite un endpoint HTTP: gli URL delle funzioni e Amazon API Gateway. In caso di dubbi su quale sia il metodo migliore per il tuo caso d'uso, consulta [the section called “URL delle funzioni e Amazon API Gateway”](#).

Quando si crea un URL della funzione, Lambda genera automaticamente un endpoint URL univoco. Dopo aver creato un URL della funzione, il suo endpoint URL non cambia mai. Gli endpoint URL della funzione hanno il formato seguente:

```
https://<url-id>.lambda-url.<region>.on.aws
```

### Note

Gli URL delle funzioni non sono supportati in tutte le regioni. AWS Per verificare la disponibilità, consulta Servizi [AWS regionali](#) e filtra per Feature Name = Function URLs.

Gli URL della funzione sono abilitati alla rete dual stack e supportano IPv4 e IPv6. Dopo aver configurato l'URL della funzione, puoi richiamarla tramite il relativo endpoint HTTP (S) utilizzando un browser web, curl, Postman o qualsiasi client HTTP. Per invocare un URL della funzione, è necessario disporre di autorizzazioni `lambda:InvokeFunctionUrl` e `lambda:InvokeFunction`. Per ulteriori informazioni, consulta [Controllo accessi](#).

### Argomenti

- [Nozioni di base sul richiamo di URL di funzioni](#)
- [Payload di richieste e risposte](#)

## Nozioni di base sul richiamo di URL di funzioni

Se l'URL della funzione utilizza il tipo di autenticazione `AWS_IAM`, è necessario firmare ogni richiesta HTTP utilizzando [AWS Signature Version 4 \(SigV4\)](#). Strumenti come [Postman](#) offrono modalità integrate per firmare le richieste con Sigv4.

Se non utilizzi uno strumento per firmare le richieste HTTP all'URL della funzione, è necessario firmare manualmente ogni richiesta utilizzando Sigv4. Quando l'URL della funzione riceve una richiesta, Lambda calcola anche la firma Sigv4. Lambda elabora la richiesta solo se le firme corrispondono. Per istruzioni su come firmare manualmente le richieste con Sigv4, consulta [Signing requests with Signature Version 4 nella AWS Guida](#). Riferimenti generali di Amazon Web Services

Se l'URL della funzione utilizza il tipo di autenticazione `NONE`, non è necessario firmare le richieste utilizzando Sigv4. Puoi richiamare la funzione utilizzando un browser Web, curl, Postman o un client HTTP.

Per verificare semplici richieste GET alla funzione, usa un browser Web. Ad esempio, se l'URL della funzione è `https://abcdefg.lambda-url.us-east-1.on.aws` e richiede un parametro stringa `message`, l'URL della richiesta potrebbe essere simile al seguente:

```
https://abcdefg.lambda-url.us-east-1.on.aws/?message=HelloWorld
```

Per verificare altre richieste HTTP, ad esempio una richiesta POST, puoi utilizzare uno strumento come curl. Ad esempio, se desideri includere alcuni dati JSON in una richiesta POST all'URL della funzione, potresti utilizzare il comando curl seguente:

```
curl -v 'https://abcdefg.lambda-url.us-east-1.on.aws/?message=HelloWorld' \  
-H 'content-type: application/json' \  
-d '{ "example": "test" }'
```

## Payload di richieste e risposte

Quando un client chiama l'URL della funzione, Lambda mappa la richiesta a un oggetto evento prima di passarlo alla funzione. La risposta della funzione viene quindi mappata a una risposta HTTP che Lambda invia al client tramite l'URL della funzione.

I formati degli eventi di richiesta e risposta seguono lo stesso schema del [tipo di formato del payload di Gateway Amazon API versione 2.0](#).

## Formato del payload di richiesta

Un payload di richiesta ha la seguente struttura:

```
{
  "version": "2.0",
  "routeKey": "$default",
  "rawPath": "/my/path",
  "rawQueryString": "parameter1=value1&parameter1=value2&parameter2=value",
  "cookies": [
    "cookie1",
    "cookie2"
  ],
  "headers": {
    "header1": "value1",
    "header2": "value1,value2"
  },
  "queryStringParameters": {
    "parameter1": "value1,value2",
    "parameter2": "value"
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "<urlid>",
    "authentication": null,
    "authorizer": {
      "iam": {
        "accessKey": "AKIA...",
        "accountId": "111122223333",
        "callerId": "AIDA...",
        "cognitoIdentity": null,
        "principalOrgId": null,
        "userArn": "arn:aws:iam::111122223333:user/example-user",
        "userId": "AIDA..."
      }
    },
    "domainName": "<url-id>.lambda-url.us-west-2.on.aws",
    "domainPrefix": "<url-id>",
    "http": {
      "method": "POST",
      "path": "/my/path",
      "protocol": "HTTP/1.1",
      "sourceIp": "123.123.123.123",
      "userAgent": "agent"
    }
  }
}
```

```

    },
    "requestId": "id",
    "routeKey": "$default",
    "stage": "$default",
    "time": "12/Mar/2020:19:03:58 +0000",
    "timeEpoch": 1583348638390
  },
  "body": "Hello from client!",
  "pathParameters": null,
  "isBase64Encoded": false,
  "stageVariables": null
}

```

| Parametro      | Description                                                                                                                                                                      | Esempio                                |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------|
| version        | Il tipo di formato del payload per questo evento. Gli URL della funzione Lambda attualmente supportano il <a href="#">tipo di formato del payload versione 2.0</a> .             | 2.0                                    |
| routeKey       | Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su \$default come segnaposto.                                                                          | \$default                              |
| rawPath        | Percorso della richiesta. Ad esempio, se l'URL della richiesta è https://{url-id}.lambda-url.{region}.on.aws/example/test/demo, il valore raw del percorso è /example/test/demo. | /example/test/demo                     |
| rawQueryString | La stringa raw contenente i parametri della stringa di query della richiesta. I caratteri                                                                                        | "?parameter1=value1&parameter2=value2" |

| Parametro                             | Description                                                                                                                                                                                                                                                                                      | Esempio                                                 |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
|                                       | supportati includono a-z, A-Z, 0-9, ., _, -, %, &, = e +.                                                                                                                                                                                                                                        |                                                         |
| <code>cookies</code>                  | Un array contenente tutti i cookie inviati come parte della richiesta.                                                                                                                                                                                                                           | <code>["Cookie_1=Value_1", "Cookie_2=Value_2"]</code>   |
| <code>headers</code>                  | L'elenco delle intestazioni della richiesta, presentate come coppie chiave-valore.                                                                                                                                                                                                               | <code>{"header1": "value1", "header2": "value2"}</code> |
| <code>queryStringParameters</code>    | I parametri di query per la richiesta. Ad esempio, se l'URL della richiesta è <code>https://{url-id}.lambda-url.{region}.on.aws/example?name=Jane</code> , il valore <code>queryStringParameters</code> è un oggetto JSON con una chiave di <code>name</code> e un valore di <code>Jane</code> . | <code>{"name": "Jane"}</code>                           |
| <code>requestContext</code>           | Un oggetto contenente informazioni aggiuntive sulla richiesta, ad esempio <code>requestId</code> , l'ora della richiesta e l'identità del chiamante se autorizzato tramite AWS Identity and Access Management (IAM).                                                                             |                                                         |
| <code>requestContext.accountId</code> | L'Account AWS ID del proprietario della funzione.                                                                                                                                                                                                                                                | <code>"123456789012"</code>                             |
| <code>requestContext.apiId</code>     | L'ID dell'URL della funzione.                                                                                                                                                                                                                                                                    | <code>"33anwqw8fj"</code>                               |



| Parametro                                                  | Description                                                                                                                                                                                            | Esempio                                       |
|------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| <code>requestContext.authentication</code>                 | Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su <code>null</code> .                                                                                                       | <code>null</code>                             |
| <code>requestContext.authorizer</code>                     | Un oggetto contenente informazioni sull'identità del chiamante, se l'URL della funzione utilizza il tipo di autenticazione <code>AWS_IAM</code> . Altrimenti, Lambda lo imposta su <code>null</code> . |                                               |
| <code>requestContext.authorizer.iam.accessKey</code>       | La chiave di accesso dell'identità del chiamante.                                                                                                                                                      | "AKIAIOSFODNN7EXAMPLE"                        |
| <code>requestContext.authorizer.iam.accountId</code>       | L' Account AWS ID dell'identità del chiamante.                                                                                                                                                         | "111122223333"                                |
| <code>requestContext.authorizer.iam.callerId</code>        | L'ID (ID utente) del chiamante.                                                                                                                                                                        | "AIDACKCEVSQ6C2EXAMPLE"                       |
| <code>requestContext.authorizer.iam.cognitoIdentity</code> | Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su <code>null</code> o lo esclude dal JSON.                                                                                  | <code>null</code>                             |
| <code>requestContext.authorizer.iam.principalOrgId</code>  | L'ID dell'organizzazione principale associato all'identità del chiamante.                                                                                                                              | "AIDACKCEVSQORGEXAMPLE"                       |
| <code>requestContext.authorizer.iam.userArn</code>         | L'Amazon Resource Name (ARN) utente dell'identità del chiamante.                                                                                                                                       | "arn:aws:iam::111122223333:user/example-user" |

| Parametro                                         | Description                                                                                                                                                                                              | Esempio                                |
|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------|
| <code>requestContext.authorizer.iam.userId</code> | L'ID utente dell'identità del chiamante.                                                                                                                                                                 | "AIDACOSF0DNN7EXAMPLE2"                |
| <code>requestContext.domainName</code>            | Il nome di dominio dell'URL della funzione.                                                                                                                                                              | "<url-id>.lambda-url.us-west-2.on.aws" |
| <code>requestContext.domainPrefix</code>          | Il prefisso di dominio dell'URL della funzione.                                                                                                                                                          | "<url-id>"                             |
| <code>requestContext.http</code>                  | Un oggetto contenente i dettagli sulla richiesta HTTP.                                                                                                                                                   |                                        |
| <code>requestContext.http.method</code>           | Il metodo HTTP utilizzato nella richiesta. I valori validi includono GET, POST, PUT, HEAD, OPTIONS, PATCH e DELETE.                                                                                      | GET                                    |
| <code>requestContext.http.path</code>             | Percorso della richiesta. Ad esempio, se l'URL della richiesta è <code>https://{url-id}.lambda-url.{region}.on.aws/example/test/demo</code> , il valore del percorso è <code>/example/test/demo</code> . | <code>/example/test/demo</code>        |
| <code>requestContext.http.protocol</code>         | Il protocollo della richiesta.                                                                                                                                                                           | HTTP/1.1                               |
| <code>requestContext.http.sourceIp</code>         | L'indirizzo IP di origine della connessione TCP immediata da cui proviene la richiesta.                                                                                                                  | 123.123.123.123                        |

| Parametro                                  | Description                                                                                                                              | Esempio                                                                     |
|--------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|
| <code>requestContext.http.userAgent</code> | Il valore dell'intestazione della User-Agent richiesta.                                                                                  | Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) Gecko/20100101 Firefox/42.0 |
| <code>requestContext.requestId</code>      | L'ID della richiesta di richiamo. È possibile utilizzare questo ID per tenere traccia dei registri dei richiami correlati alla funzione. | e1506fd5-9e7b-434f-bd42-4f8fa224b599                                        |
| <code>requestContext.routeKey</code>       | Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su <code>\$default</code> come segnaposto.                     | <code>\$default</code>                                                      |
| <code>requestContext.stage</code>          | Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su <code>\$default</code> come segnaposto.                     | <code>\$default</code>                                                      |
| <code>requestContext.time</code>           | Il timestamp della richiesta.                                                                                                            | "07/Sep/2021:22:50:22 +0000"                                                |
| <code>requestContext.timeEpoch</code>      | Il timestamp della richiesta, in formato temporale Unix epoch.                                                                           | "1631055022677"                                                             |
| <code>body</code>                          | Il corpo della richiesta. Se il tipo di contenuto della richiesta è binario, il corpo è con codifica base64.                             | {"key1": "value1", "key2": "value2"}                                        |
| <code>pathParameters</code>                | Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su <code>null</code> o lo esclude dal JSON.                    | <code>null</code>                                                           |

| Parametro       | Description                                                                                              | Esempio |
|-----------------|----------------------------------------------------------------------------------------------------------|---------|
| isBase64Encoded | TRUE se il corpo è un payload binario e con codifica base64. FALSE in caso contrario.                    | FALSE   |
| stageVariables  | Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su null o lo esclude dal JSON. | null    |

## Formato del payload di risposta

Quando la funzione restituisce una risposta, Lambda analizza la risposta e la converte in una risposta HTTP. I payload di risposta della funzione hanno il formato seguente:

```
{
  "statusCode": 201,
  "headers": {
    "Content-Type": "application/json",
    "My-Custom-Header": "Custom Value"
  },
  "body": "{ \"message\": \"Hello, world!\" }",
  "cookies": [
    "Cookie_1=Value1; Expires=21 Oct 2021 07:48 GMT",
    "Cookie_2=Value2; Max-Age=78000"
  ],
  "isBase64Encoded": false
}
```

Lambda deduce il formato di risposta per l'utente. Se la funzione Lambda restituisce un JSON valido e non restituisce un `statusCode`, Lambda presuppone quanto segue:

- `statusCode` è 200.

### Note

I valori validi `statusCode` sono compresi tra 100 e 599.

- `content-type` è `application/json`.

- `body` è la risposta della funzione.
- `isBase64Encoded` è `false`.

Gli esempi seguenti mostrano come l'output della funzione Lambda viene mappato al payload della risposta e come il payload della risposta viene mappato alla risposta HTTP finale. Quando il client richiama l'URL della funzione, viene visualizzata la risposta HTTP.

#### Esempio di output per una risposta stringa

| Risultato della funzione Lambda | Output di risposta interpretata                                                                                                                      | Risposta HTTP (cosa vede il client)                                                                                          |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| "Hello, world!"                 | <pre>{   "statusCode": 200,   "body": "Hello, world!",   "headers": {     "content-type": "application/json"   },   "isBase64Encoded": false }</pre> | <pre>HTTP/2 200 date: Wed, 08 Sep 2021 18:02:24 GMT content-type: application/json content-length: 15  "Hello, world!"</pre> |

#### Esempio di output per una risposta JSON

| Risultato della funzione Lambda             | Output di risposta interpretata                                                                                                               | Risposta HTTP (cosa vede il client)                                                                            |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <pre>{   "message": "Hello, world!" }</pre> | <pre>{   "statusCode": 200,   "body": {     "message": "Hello, world!"   },   "headers": {     "content-type": "application/json"   } }</pre> | <pre>HTTP/2 200 date: Wed, 08 Sep 2021 18:02:24 GMT content-type: application/json content-length: 34  {</pre> |

| Risultato della funzione Lambda | Output di risposta interpretata            | Risposta HTTP (cosa vede il client)           |
|---------------------------------|--------------------------------------------|-----------------------------------------------|
|                                 | <pre> }, "isBase64Encoded": false } </pre> | <pre> {   "message": "Hello, world!" } </pre> |

### Esempio di output per una risposta personalizzata

| Risultato della funzione Lambda                                                                                                                                                                                                      | Output di risposta interpretata                                                                                                                                                                                                      | Risposta HTTP (cosa vede il client)                                                                                                                                             |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> {   "statusCode": 201,   "headers": {     "Content-Type": "application/json",     "My-Custom-Header": "Custom Value"   },   "body": JSON.stri ngify({     "message": "Hello, world!"   }),   "isBase64Encoded": false } </pre> | <pre> {   "statusCode": 201,   "headers": {     "Content-Type": "application/json",     "My-Custom-Header": "Custom Value"   },   "body": JSON.stri ngify({     "message": "Hello, world!"   }),   "isBase64Encoded": false } </pre> | <pre> HTTP/2 201 date: Wed, 08 Sep 2021 18:02:24 GMT content-type: applicati on/json content-length: 27 my-custom-header: Custom Value  {   "message": "Hello, world!" } </pre> |

### Cookie

Per restituire i cookie della funzione, non aggiungere manualmente intestazioni set-cookie. Al contrario, includi i cookie nell'oggetto payload di risposta. Lambda li interpreta automaticamente e li aggiunge come intestazioni set-cookie nella risposta HTTP, come nell'esempio seguente.

## Risultato della funzione Lambda

```
{
  "statusCode": 201,
  "headers": {
    "Content-Type": "application/
json",
    "My-Custom-Header": "Custom
Value"
  },
  "body": JSON.stringify({
    "message": "Hello, world!"
  }),
  "cookies": [
    "Cookie_1=Value1; Expires=21
Oct 2021 07:48 GMT",
    "Cookie_2=Value2; Max-Age=7
8000"
  ],
  "isBase64Encoded": false
}
```

## Risposta HTTP (cosa vede il client)

```
HTTP/2 201
date: Wed, 08 Sep 2021 18:02:24 GMT
content-type: application/json
content-length: 27
my-custom-header: Custom Value
set-cookie: Cookie_1=Value2;
  Expires=21 Oct 2021 07:48 GMT
set-cookie: Cookie_2=Value2; Max-
Age=78000

{
  "message": "Hello, world!"
}
```

## Monitoraggio degli URL della funzione Lambda

Puoi utilizzare AWS CloudTrail e Amazon CloudWatch per monitorare gli URL della funzione.

### Argomenti

- [Monitoraggio degli URL della funzione con CloudTrail](#)
- [Parametri CloudWatch per gli URL della funzione](#)

### Monitoraggio degli URL della funzione con CloudTrail

Per gli URL della funzione, Lambda supporta automaticamente la registrazione delle seguenti operazioni API come eventi nei file di log di CloudTrail:

- [CreateFunctionUrlConfig](#)
- [UpdateFunctionUrlConfig](#)
- [DeleteFunctionUrlConfig](#)
- [GetFunctionUrlConfig](#)
- [ListFunctionUrlConfigs](#)

Ogni voce di registro contiene informazioni sull'identità del chiamante, sul momento in cui è stata effettuata la richiesta e altri dettagli. Puoi vedere tutti gli eventi degli ultimi 90 giorni visualizzando la cronologia eventi di CloudTrail. Per conservare i record degli ultimi 90 giorni, puoi creare un percorso.

Per impostazione predefinita, CloudTrail non registra le richieste `InvokeFunctionUrl`, che sono considerate eventi di dati. È comunque possibile attivare la registrazione degli eventi di dati in CloudTrail. Per ulteriori informazioni, consulta [Registrazione di eventi di dati per i percorsi](#) nella Guida per l'utente di AWS CloudTrail.

### Parametri CloudWatch per gli URL della funzione

Lambda invia parametri aggregati relativi alle richieste degli URL della funzione a CloudWatch. Con questi parametri, puoi monitorare gli URL della funzione, creare dashboard e configurare allarmi nella console di CloudWatch.

Gli URL della funzione supportano i seguenti parametri di richiamo. È consigliabile visualizzare questi parametri con le statistiche di Sum.



- `UrlRequestCount`: il numero di richieste inviate all'URL della funzione.
- `Url4xxCount`: il numero di richieste che hanno restituito un codice di stato HTTP 4XX. I codici della serie 4XX indicano errori lato client, ad esempio richieste non valide.
- `Url5xxCount`: il numero di richieste che hanno restituito un codice di stato HTTP 5XX. I codici della serie 5XX indicano errori lato server, ad esempio errori di funzione e timeout.

Gli URL della funzione supportano anche il seguente parametro di prestazioni. È consigliabile visualizzare questo parametro con le statistiche di Average o Max.

- `UrlRequestLatency`: il periodo di tempo che intercorre tra il momento in cui l'URL della funzione riceve una richiesta e il momento in cui l'URL della funzione restituisce una risposta.

Ciascuno di questi parametri di richiamo e prestazioni supporta le seguenti dimensioni:

- `FunctionName`: visualizza i parametri aggregati per gli URL della funzione assegnati a una versione `$LATEST` non pubblicata di una funzione o a uno degli alias della funzione. Ad esempio, `hello-world-function`.
- `Resource`: visualizza i parametri relativi a un URL della funzione specifico. Questo URL è definito dal nome di una funzione, insieme alla versione `$LATEST` non pubblicata della funzione o a uno degli alias della funzione. Ad esempio, `hello-world-function:$LATEST`.
- `ExecutedVersion`: visualizza i parametri per un URL di funzione specifico, in base alla versione eseguita. Puoi utilizzare questa dimensione principalmente per tenere traccia dell'URL della funzione assegnato alla versione `$LATEST` non pubblicata.

## Selezionare un metodo per richiamare la funzione Lambda tramite una richiesta HTTP

Numerosi casi d'uso comuni per Lambda implicano l'invocazione della funzione tramite una richiesta HTTP. Ad esempio, potrebbe essere preferibile che un'applicazione Web invochi la funzione tramite una richiesta del browser. Le funzioni Lambda possono essere utilizzate anche per creare REST API complete, gestire le interazioni degli utenti da app mobili, elaborare dati da servizi esterni tramite chiamate HTTP o creare webhook personalizzati.

Le sezioni seguenti spiegano quali sono le tue scelte per richiamare Lambda tramite HTTP e forniscono informazioni per aiutarti a prendere la decisione giusta per il tuo caso d'uso particolare.

Quali sono le tue scelte quando selezioni un metodo di invocazione HTTP?

Lambda offre due metodi principali per richiamare una funzione utilizzando una richiesta HTTP: [URL di funzione](#) e [API Gateway](#). Le differenze tra queste due opzioni sono le seguenti:

- Gli URL delle funzioni Lambda forniscono un endpoint HTTP semplice e diretto per una funzione Lambda. Sono ottimizzati per semplicità ed economicità e forniscono il percorso più veloce per esporre una funzione Lambda tramite HTTP.
- API Gateway è un servizio più avanzato per la creazione di API complete. API Gateway è ottimizzato per la creazione e la gestione di API di produzione su larga scala e fornisce strumenti completi per la sicurezza, il monitoraggio e la gestione del traffico.

Suggerimenti se conosci già le tue esigenze

Se hai già le idee chiare sulle tue esigenze, ecco i nostri suggerimenti di base:

Consigliamo gli URL di [funzione](#) per applicazioni semplici o prototipi in cui sono necessari solo metodi e request/response gestione di autenticazione di base e dove si desidera ridurre al minimo i costi e la complessità.

[API Gateway](#) è la scelta migliore per le applicazioni di produzione su larga scala o per i casi in cui sono necessarie funzionalità più avanzate come il [supporto per](#) OpenAPI Description, una scelta di opzioni di autenticazione, nomi di dominio personalizzati o una request/response gestione avanzata che include limitazione, memorizzazione nella cache e trasformazione. request/response

Cosa considerare quando si seleziona un metodo per richiamare la funzione Lambda

Quando si sceglie tra le funzioni URL e API Gateway, è necessario considerare i seguenti fattori:

- Le tue esigenze di autenticazione, ad esempio se richiedi OAuth o Amazon Cognito per autenticare gli utenti
- I tuoi requisiti di scalabilità e la complessità dell'API che desideri implementare
- Se hai bisogno di funzionalità avanzate come la convalida e la formattazione delle richieste request/response
- I tuoi requisiti di monitoraggio
- I tuoi obiettivi di costo

Comprendendo questi fattori, è possibile selezionare l'opzione che meglio bilancia i requisiti di sicurezza, complessità e costi.

Le informazioni seguente riepilogano le differenze principali tra le due opzioni.

### Autenticazione

- Gli URL delle funzioni forniscono opzioni di autenticazione di base tramite AWS Identity and Access Management (IAM). Puoi configurare gli endpoint in modo che siano pubblici (senza autenticazione) o che richiedano l'autenticazione IAM. Con l'autenticazione IAM, puoi utilizzare AWS credenziali standard o ruoli IAM per controllare l'accesso. Sebbene sia semplice da configurare, questo approccio offre opzioni limitate rispetto ad altri metodi di autenticazione.
- API Gateway fornisce l'accesso a una gamma più completa di opzioni di autenticazione. Oltre all'autenticazione IAM, puoi utilizzare autorizzatori [Lambda](#) (logica di autenticazione personalizzata), pool di [utenti](#) Amazon Cognito e flussi. OAuth2.0 Questa flessibilità consente di implementare schemi di autenticazione complessi, inclusi provider di autenticazione di terze parti, autenticazione basata su token e autenticazione a più fattori.

### Request/response gestione

- Le funzioni URL forniscono la gestione di base delle richieste e delle risposte HTTP. Supportano i metodi HTTP standard e includono il supporto integrato per CORS (cross-origin resource sharing). Sebbene siano in grado di gestire i payload JSON e i parametri di query in modo naturale, non offrono funzionalità di trasformazione o convalida delle richieste. La gestione delle risposte è altrettanto semplice: il client riceve la risposta dalla funzione Lambda esattamente come la restituisce Lambda.
- API Gateway offre sofisticate funzionalità di gestione delle richieste e delle risposte. È possibile definire validatori di richieste, trasformare richieste e risposte utilizzando modelli di mappatura, impostare le request/response intestazioni e implementare la memorizzazione nella cache delle risposte. API Gateway supporta anche payload binari e nomi di dominio personalizzati e può modificare le risposte prima che raggiungano il client. È possibile impostare modelli per la request/response convalida e la trasformazione utilizzando JSON Schema.

### Dimensionamento

- Le funzioni URL si adattano direttamente ai limiti di simultaneità della funzione Lambda e gestiscono i picchi di traffico scalando la funzione fino al limite di simultaneità massimo configurato.

Una volta raggiunto tale limite, Lambda risponde alle richieste aggiuntive con risposte HTTP 429. Non esiste un meccanismo di accodamento integrato, pertanto la gestione della scalabilità dipende interamente dalla configurazione della funzione Lambda. Per impostazione predefinita, le funzioni Lambda hanno un limite di 1.000 esecuzioni simultanee per Regione AWS.

- API Gateway offre funzionalità di scalabilità aggiuntive oltre alla scalabilità propria di Lambda. Include controlli integrati per l'accodamento e la limitazione delle richieste che consentono di gestire i picchi di traffico con maggiore efficienza. Per impostazione predefinita, API Gateway è in grado di gestire fino a 10.000 richieste al secondo per regione, con una capacità di espansione di 5.000 richieste al secondo. Fornisce inoltre strumenti per limitare le richieste a diversi livelli (API, fase o metodo) per proteggere il backend.

## Monitoraggio

- Gli URL delle funzioni offrono un monitoraggio di base tramite le CloudWatch metriche di Amazon, tra cui il numero di richieste, la latenza e i tassi di errore. Puoi accedere a parametri e log Lambda standard, che mostrano le richieste non elaborate che arrivano alla tua funzione. Sebbene ciò fornisca una visibilità operativa essenziale, i parametri si concentrano principalmente sull'esecuzione delle funzioni.
- API Gateway offre funzionalità di monitoraggio complete, tra cui parametri dettagliati, opzioni di registrazione e tracciamento. Puoi monitorare le chiamate API, la latenza, i tassi di errore e i tassi di cache tramite hit/miss CloudWatch API Gateway si integra anche con il AWS X-Ray tracciamento distribuito e fornisce formati di registrazione personalizzabili.

## Costo

- Le funzioni URL seguono il modello di prezzo standard di Lambda: paghi solo per le invocazioni di funzione e il tempo di calcolo. Non sono previsti costi aggiuntivi per l'endpoint URL stesso. Ciò lo rende una scelta conveniente per API semplici o applicazioni a basso traffico se non sono necessarie le funzionalità aggiuntive di API Gateway.
- API Gateway offre un [piano gratuito](#) che include un milione di chiamate API ricevute per le REST API e un milione di chiamate API ricevute per le API HTTP. Successivamente, API Gateway addebita i costi per le chiamate API, il trasferimento dei dati e la memorizzazione nella cache (se abilitata). Consulta la [pagina dei prezzi](#) di API Gateway per conoscere i costi relativi al tuo caso d'uso.

## Altre funzionalità

- Le funzioni URL sono progettati per la semplicità e l'integrazione diretta con Lambda. Supportano sia endpoint HTTP che HTTPS, offrono supporto CORS integrato e forniscono endpoint dual stack (IPv4 e IPv6). Sebbene non dispongano di funzionalità avanzate, eccellono negli scenari in cui è necessario un modo rapido e diretto per esporre le funzioni Lambda tramite HTTP.
- API Gateway include numerose funzionalità aggiuntive come il controllo delle versioni delle API, la gestione delle fasi, le chiavi API per i piani di utilizzo, la documentazione delle API tramite, le API Swagger/OpenAPI, WebSocket le API private all'interno di un VPC e l'integrazione WAF per una maggiore sicurezza. Supporta inoltre implementazioni canarie, integrazioni simulate per i test e l'integrazione con altre applicazioni oltre a Lambda. Servizi AWS

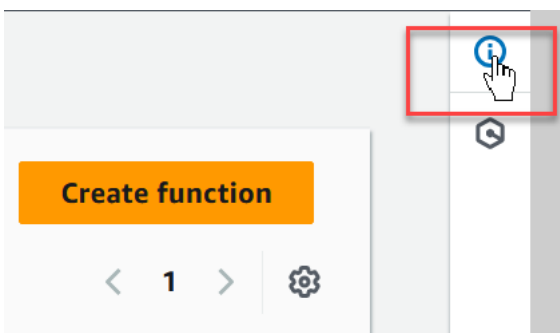
Selezionare un metodo per richiamare la funzione Lambda

Ora che hai letto i criteri per la selezione tra le funzioni Lambda URL e l'API Gateway e le principali differenze tra questi, puoi selezionare l'opzione più adatta alle tue esigenze e utilizzare le seguenti risorse per iniziare a utilizzarla.

## Function URLs

Iniziare a usare le funzioni URL con le seguenti risorse

- Segui il tutorial [Creazione di una funzione Lambda con la funzione URL](#)
  - Scopri di più sulle funzioni URL nel capitolo [the section called “URL delle funzioni”](#) di questa guida
  - Prova il tutorial guidato dalla console Creare una semplice app Web effettuando le seguenti operazioni:
1. Apri la pagina [Funzioni](#) della console Lambda.
  2. Apri il pannello di aiuto scegliendo l'icona nell'angolo in alto a destra della schermata.



3. Seleziona Tutorial.
4. In Crea una semplice app Web, scegli Avvia tutorial.

## API Gateway

Iniziare a usare Lambda e API Gateway con le seguenti risorse

- Segui il tutorial [Utilizzo di Lambda con API Gateway](#) per creare una REST API integrata con una funzione Lambda di backend.
- Scopri di più sui diversi tipi di API offerti da API Gateway nelle seguenti sezioni della Guida per gli sviluppatori di Gateway Amazon API:
  - [REST API di API Gateway](#)
  - [API HTTP di API Gateway](#)
  - [WebSocket API Gateway](#)
- Prova uno o più esempi nella sezione [Tutorial e workshop](#) della Guida per gli sviluppatori di Gateway Amazon API.

## Tutorial: creazione di un endpoint webhook utilizzando l'URL di una funzione Lambda

In questo tutorial, crei l'URL di una funzione Lambda per implementare un endpoint webhook. Un webhook è una comunicazione leggera basata sugli eventi che invia automaticamente i dati tra le applicazioni tramite HTTP. È possibile utilizzare un webhook per ricevere aggiornamenti immediati sugli eventi che si verificano in un altro sistema, ad esempio quando un nuovo cliente si registra su un sito Web, viene elaborato un pagamento o viene caricato un file.

Con Lambda, i webhook possono essere implementati utilizzando la funzione Lambda o l'API Gateway. Le funzioni sono un'ottima scelta per webhook semplici che non richiedono funzionalità come l'autorizzazione avanzata o la convalida delle richieste.

### Tip

Se non sei sicuro di quale sia la soluzione migliore per il tuo caso d'uso particolare, consulta [the section called “URL delle funzioni e Amazon API Gateway”](#)

## Prerequisiti

Per completare questo tutorial, devi avere Python (versione 3.8 o successiva) o Node.js (versione 18 o successiva) installato sul tuo computer locale.

Per testare l'endpoint utilizzando una richiesta HTTP, il tutorial utilizza [curl](#), uno strumento a riga di comando che è possibile utilizzare per trasferire dati utilizzando vari protocolli di rete. Fai riferimento alla [documentazione di curl](#) per scoprire come installare lo strumento se non lo possiedi già.

## Creazione della funzione Lambda

Per prima cosa crea la funzione Lambda che viene eseguita quando una richiesta HTTP viene inviata al tuo endpoint webhook. In questo esempio, l'applicazione di invio invia un aggiornamento ogni volta che viene inviato un pagamento e indica nel corpo della richiesta HTTP se il pagamento è andato a buon fine. La funzione Lambda analizza la richiesta e interviene in base allo stato del pagamento. In questo esempio, il codice stampa solo l'ID dell'ordine per il pagamento, ma in un'applicazione reale è possibile aggiungere l'ordine a un database o inviare una notifica.

La funzione implementa anche il metodo di autenticazione più comune utilizzato per i webhook, l'autenticazione dei messaggi basata su hash (HMAC). Con questo metodo, sia l'applicazione di invio che quella di ricezione condividono una chiave segreta. L'applicazione di invio utilizza un

algoritmo di hashing per generare una firma univoca utilizzando questa chiave insieme al contenuto del messaggio e include la firma nella richiesta del webhook come intestazione HTTP. L'applicazione ricevente ripete quindi questo passaggio, generando la firma utilizzando la chiave segreta e confronta il valore risultante con la firma inviata nell'intestazione della richiesta. Se il risultato corrisponde, la richiesta viene considerata legittima.

Crea la funzione utilizzando la console Lambda con il runtime Python o Node.js.

## Python

### Creazione della funzione Lambda

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Crea una funzione di base «Hello world» effettuando le seguenti operazioni:
  - a. Scegli Crea funzione.
  - b. Scegli Crea da zero.
  - c. Nel campo Function name (Nome funzione), immettere **myLambdaWebhook**.
  - d. Per Runtime, seleziona python3.14.
  - e. Scegli Crea funzione.
3. Nel riquadro Codice sorgente, sostituisci il codice esistente copiando e incollando quanto segue:

```
import json
import hmac
import hashlib
import os

def lambda_handler(event, context):

    # Get the webhook secret from environment variables
    webhook_secret = os.environ['WEBHOOK_SECRET']

    # Verify the webhook signature
    if not verify_signature(event, webhook_secret):
        return {
            'statusCode': 401,
            'body': json.dumps({'error': 'Invalid signature'})
        }
```



```
try:
    # Parse the webhook payload
    payload = json.loads(event['body'])

    # Handle different event types
    event_type = payload.get('type')

    if event_type == 'payment.success':
        # Handle successful payment
        order_id = payload.get('orderId')
        print(f"Processing successful payment for order {order_id}")

        # Add your business logic here
        # For example, update database, send notifications, etc.

    elif event_type == 'payment.failed':
        # Handle failed payment
        order_id = payload.get('orderId')
        print(f"Processing failed payment for order {order_id}")

        # Add your business logic here

    else:
        print(f"Received unhandled event type: {event_type}")

    # Return success response
    return {
        'statusCode': 200,
        'body': json.dumps({'received': True})
    }

except json.JSONDecodeError:
    return {
        'statusCode': 400,
        'body': json.dumps({'error': 'Invalid JSON payload'})
    }

except Exception as e:
    print(f"Error processing webhook: {e}")
    return {
        'statusCode': 500,
        'body': json.dumps({'error': 'Internal server error'})
    }

def verify_signature(event, webhook_secret):
```

```

"""
Verify the webhook signature using HMAC
"""
try:
    # Get the signature from headers
    signature = event['headers'].get('x-webhook-signature')

    if not signature:
        print("Error: Missing webhook signature in headers")
        return False

    # Get the raw body (return an empty string if the body key doesn't
    exist)
    body = event.get('body', '')

    # Create HMAC using the secret key
    expected_signature = hmac.new(
        webhook_secret.encode('utf-8'),
        body.encode('utf-8'),
        hashlib.sha256
    ).hexdigest()

    # Compare the expected signature with the received signature to
    authenticate the message
    is_valid = hmac.compare_digest(signature, expected_signature)
    if not is_valid:
        print(f"Error: Invalid signature. Received: {signature}, Expected:
        {expected_signature}")
        return False

    return True
except Exception as e:
    print(f"Error verifying signature: {e}")
    return False

```

4. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione.

## Node.js

### Creazione della funzione Lambda

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Crea una funzione di base «Hello world» effettuando le seguenti operazioni:

- a. Scegli Crea funzione.
  - b. Scegli Crea da zero.
  - c. Nel campo Function name (Nome funzione), immettere **myLambdaWebhook**.
  - d. Per Runtime, seleziona nodejs24.x.
  - e. Scegli Crea funzione.
3. Nel riquadro Codice sorgente, sostituisci il codice esistente copiando e incollando quanto segue:

```
import crypto from 'crypto';

export const handler = async (event, context) => {
  // Get the webhook secret from environment variables
  const webhookSecret = process.env.WEBHOOK_SECRET;

  // Verify the webhook signature
  if (!verifySignature(event, webhookSecret)) {
    return {
      statusCode: 401,
      body: JSON.stringify({ error: 'Invalid signature' })
    };
  }

  try {
    // Parse the webhook payload
    const payload = JSON.parse(event.body);

    // Handle different event types
    const eventType = payload.type;

    switch (eventType) {
      case 'payment.success': {
        // Handle successful payment
        const orderId = payload.orderId;
        console.log(`Processing successful payment for order
${orderId}`);

        // Add your business logic here
        // For example, update database, send notifications, etc.
        break;
      }
    }
  }
}
```

```
        case 'payment.failed': {
            // Handle failed payment
            const orderId = payload.orderId;
            console.log(`Processing failed payment for order ${orderId}`);

            // Add your business logic here
            break;
        }

        default:
            console.log(`Received unhandled event type: ${eventType}`);
    }

    // Return success response
    return {
        statusCode: 200,
        body: JSON.stringify({ received: true })
    };

} catch (error) {
    if (error instanceof SyntaxError) {
        // Handle JSON parsing errors
        return {
            statusCode: 400,
            body: JSON.stringify({ error: 'Invalid JSON payload' })
        };
    }

    // Handle all other errors
    console.error('Error processing webhook:', error);
    return {
        statusCode: 500,
        body: JSON.stringify({ error: 'Internal server error' })
    };
}

};

// Verify the webhook signature using HMAC

const verifySignature = (event, webhookSecret) => {
    try {
        // Get the signature from headers
        const signature = event.headers['x-webhook-signature'];
```

```
    if (!signature) {
        console.log('No signature found in headers:', event.headers);
        return false;
    }

    // Get the raw body (return an empty string if the body key doesn't
    exist)
    const body = event.body || '';

    // Create HMAC using the secret key
    const hmac = crypto.createHmac('sha256', webhookSecret);
    const expectedSignature = hmac.update(body).digest('hex');

    // Compare expected and received signatures
    const isValid = signature === expectedSignature;
    if (!isValid) {
        console.log(`Invalid signature. Received: ${signature}, Expected:
        ${expectedSignature}`);
        return false;
    }

    return true;
} catch (error) {
    console.error('Error during signature verification:', error);
    return false;
}
};
```

4. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione.

## Crea la chiave segreta

Per autenticare la richiesta webhook, la funzione Lambda utilizza una chiave segreta che condivide con l'applicazione chiamante. In questo esempio, la chiave viene memorizzata in una variabile di ambiente. In un'applicazione di produzione, non includere informazioni sensibili come le password nel codice funzione. Invece, [crea un AWS Secrets Manager segreto](#) e poi [usa l'estensione Lambda AWS Parameters and Secrets](#) per recuperare le credenziali nella funzione Lambda.

## Crea e archivia la chiave segreta del webhook

1. Genera una stringa lunga e casuale utilizzando un generatore di numeri casuali crittograficamente sicuro. Puoi usare i seguenti frammenti di codice in Python o Node.js per generare e stampare un segreto di 32 caratteri o usare il tuo metodo preferito.

### Python

Example codice per generare un segreto

```
import secrets
webhook_secret = secrets.token_urlsafe(32)
print(webhook_secret)
```

### Node.js

Example codice per generare un segreto (formato del modulo ES)

```
import crypto from 'crypto';
let webhookSecret = crypto.randomBytes(32).toString('base64');
console.log(webhookSecret)
```

2. Memorizza la stringa generata come variabile di ambiente per la tua funzione effettuando le seguenti operazioni:
  - a. Nella scheda Configurazione della funzione, seleziona Variabili di ambiente.
  - b. Scegli Modifica.
  - c. Scegli Add environment variable (Aggiungi variabile d'ambiente).
  - d. Per Chiave **WEBHOOK\_SECRET**, inserisci, quindi per Valore, inserisci il segreto generato nel passaggio precedente.
  - e. Scegli Save (Salva).

Dovrai utilizzare nuovamente questo segreto più avanti nel tutorial per testare la tua funzione, quindi prendine nota ora.

## Crea l'endpoint URL della funzione

Crea un endpoint per il tuo webhook utilizzando l'URL di una funzione Lambda. Poiché utilizzi il tipo di autenticazione di **NONE** per creare un endpoint con accesso pubblico, chiunque disponga dell'URL

può richiamare la tua funzione. Per ulteriori informazioni sul controllo dell'accesso alla funzione, consulta. URLs [the section called “Controllo accessi”](#) Se hai bisogno di opzioni di autenticazione più avanzate per il tuo webhook, prendi in considerazione l'utilizzo di API Gateway.

### Crea l'endpoint URL della funzione

1. Nella scheda Configurazione della funzione, selezionate Function URL.
2. Scegli Crea URL della funzione.
3. Per il tipo di autenticazione, seleziona NESSUNA.
4. Scegli Save (Salva).

L'endpoint per l'URL della funzione appena creato viene visualizzato nel riquadro URL della funzione. Copia l'endpoint per utilizzarlo più avanti nel tutorial.

### Prova la funzione nella console

Prima di utilizzare una richiesta HTTP per richiamare la funzione utilizzando l'endpoint URL, testala nella console per confermare che il codice funzioni come previsto.

Per verificare la funzione nella console, devi prima calcolare una firma webhook utilizzando il segreto generato in precedenza nel tutorial con il seguente payload JSON di test:

```
{
  "type": "payment.success",
  "orderId": "1234",
  "amount": "99.99"
}
```

Usa uno dei seguenti esempi di codice Python o Node.js per calcolare la firma del webhook usando il tuo segreto.

### Python

#### Calcola la firma del webhook

1. Salva il codice seguente come file denominato `calculate_signature.py`. Sostituisci il webhook secret nel codice con il tuo valore.

```
import secrets
import hmac
```

```
import json
import hashlib

webhook_secret = "arlbSDCP86n_1H90s0fL_Qb2NAHBIBQ0yGI0X4Zay4M"

body = json.dumps({"type": "payment.success", "orderId": "1234", "amount":
    "99.99"})

signature = hmac.new(
    webhook_secret.encode('utf-8'),
    body.encode('utf-8'),
    hashlib.sha256
).hexdigest()

print(signature)
```

2. Calcola la firma eseguendo il comando seguente dalla stessa directory in cui hai salvato il codice. Copia la firma generata dal codice.

```
python calculate_signature.py
```

## Node.js

### Calcola la firma del webhook

1. Salva il codice seguente come file denominato `calculate_signature.mjs`. Sostituisci il webhook secret nel codice con il tuo valore.

```
import crypto from 'crypto';

const webhookSecret = "arlbSDCP86n_1H90s0fL_Qb2NAHBIBQ0yGI0X4Zay4M"
const body = "{\"type\": \"payment.success\", \"orderId\": \"1234\", \"amount\": \"99.99\"}";

let hmac = crypto.createHmac('sha256', webhookSecret);
let signature = hmac.update(body).digest('hex');

console.log(signature);
```

2. Calcola la firma eseguendo il comando seguente dalla stessa directory in cui hai salvato il codice. Copia la firma generata dal codice.



```
node calculate_signature.mjs
```

Ora puoi testare il codice della funzione utilizzando una richiesta HTTP di prova nella console.

Prova la funzione nella console

1. Seleziona la scheda Codice relativa alla tua funzione.
2. Nella sezione EVENTI DI TEST, scegli Crea nuovo evento di test
3. Per Event name (Nome evento), immettere **myEvent**.
4. Sostituisci il JSON esistente copiando e incollando quanto segue nel riquadro Event JSON. Sostituisci la firma del webhook con il valore calcolato nel passaggio precedente.

```
{
  "headers": {
    "Content-Type": "application/json",
    "x-webhook-signature":
    "2d672e7a0423fab740fbc040e801d1241f2df32d2ffd8989617a599486553e2a"
  },
  "body": "{\"type\": \"payment.success\", \"orderId\": \"1234\", \"amount\": \"99.99\"}"
}
```

5. Scegli Save (Salva).
6. Scegli Richiama .

Verrà visualizzato un output simile al seguente:

Python

```
Status: Succeeded
Test Event Name: myEvent

Response:
{
  "statusCode": 200,
  "body": "{\"received\": true}"
}

Function Logs:
```

```
START RequestId: 50cc0788-d70e-453a-9a22-ceaa210e8ac6 Version: $LATEST
Processing successful payment for order 1234
END RequestId: 50cc0788-d70e-453a-9a22-ceaa210e8ac6
REPORT RequestId: 50cc0788-d70e-453a-9a22-ceaa210e8ac6 Duration: 1.55 ms Billed
Duration: 2 ms Memory Size: 128 MB Max Memory Used: 36 MB Init Duration: 136.32
ms
```

## Node.js

```
Status: Succeeded
Test Event Name: myEvent

Response:
{
  "statusCode": 200,
  "body": "{\"received\":true}"
}

Function Logs:
START RequestId: e54fe6c7-1df9-4f05-a4c4-0f71cacd64f4 Version: $LATEST
2025-01-10T18:05:42.062Z e54fe6c7-1df9-4f05-a4c4-0f71cacd64f4 INFO Processing
successful payment for order 1234
END RequestId: e54fe6c7-1df9-4f05-a4c4-0f71cacd64f4
REPORT RequestId: e54fe6c7-1df9-4f05-a4c4-0f71cacd64f4 Duration: 60.10 ms Billed
Duration: 61 ms Memory Size: 128 MB Max Memory Used: 72 MB Init Duration:
174.46 ms

Request ID: e54fe6c7-1df9-4f05-a4c4-0f71cacd64f4
```

Prova la funzione utilizzando una richiesta HTTP

Usa lo strumento da riga di comando curl per testare il tuo endpoint webhook.

Prova la funzione usando le richieste HTTP

1. In un terminale o in un programma shell, esegui il seguente comando curl. Sostituisci l'URL con il valore dell'endpoint URL della tua funzione e sostituisci la firma del webhook con la firma calcolata utilizzando la tua chiave segreta.

```
curl -X POST https://ryqgmbx5xjzxahif6frvzikpre0bpvpf.lambda-url.us-west-2.on.aws/
\
```

```
-H "Content-Type: application/json" \  
-H "x-webhook-  
signature: d5f52b76ffba65ff60ea73da67bdf1fc5825d4db56b5d3ffa0b64b7cb85ef48b" \  
-d '{"type": "payment.success", "orderId": "1234", "amount": "99.99"}'
```

Verrà visualizzato l'output seguente:

```
{"received": true}
```

2. Esamina CloudWatch i log della tua funzione per confermare che abbia analizzato correttamente il payload effettuando le seguenti operazioni:
  - a. Apri la pagina del [gruppo Logs](#) nella CloudWatch console Amazon.
  - b. Seleziona il gruppo di log della tua funzione (/aws/lambda/myLambdaWebhook).
  - c. Seleziona il flusso di log più recente.

Dovresti vedere un output simile al seguente nei log della tua funzione:

Python

```
Processing successful payment for order 1234
```

Node.js

```
2025-01-10T18:05:42.062Z e54fe6c7-1df9-4f05-a4c4-0f71cacd64f4 INFO  
Processing successful payment for order 1234
```

3. Verifica che il codice rilevi una firma non valida eseguendo il seguente comando curl. Sostituisci l'URL con il tuo endpoint URL della funzione.

```
curl -X POST https://ryqgmbx5xjzxahif6frvzikpre0bpvpf.lambda-url.us-west-2.on.aws/  
\  
-H "Content-Type: application/json" \  
-H "x-webhook-signature: abcdefg" \  
-d '{"type": "payment.success", "orderId": "1234", "amount": "99.99"}'
```

Verrà visualizzato l'output seguente:

```
{"error": "Invalid signature"}
```

## Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili a tuo carico. Account AWS

Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **confirm** nel campo di immissione del testo, quindi scegli Elimina.

Quando hai creato la funzione Lambda nella console, Lambda ha anche creato un [ruolo di esecuzione](#) per la tua funzione.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina [Ruoli](#) della console IAM.
2. Seleziona il ruolo di esecuzione creato da Lambda. Il ruolo ha il formato myLambdaWebhook-role-`<random string>` del nome.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Elimina.

## Informazioni sulla scalabilità della funzione Lambda

La concorrenza è il numero di richieste in volo che la tua AWS Lambda funzione sta gestendo contemporaneamente. Per ogni richiesta simultanea, Lambda fornisce un'istanza separata del tuo ambiente di esecuzione. Man mano che le funzioni ricevono più richieste, Lambda gestisce automaticamente il dimensionamento del numero di ambienti di esecuzione fino al raggiungimento del limite di simultaneità dell'account. Per impostazione predefinita, Lambda fornisce all'account un limite totale di simultaneità pari a 1.000 esecuzioni simultanee per tutte le funzioni in una Regione AWS. Per soddisfare le esigenze specifiche dell'account, è possibile [richiedere un aumento della quota](#) e configurare i controlli di simultaneità a livello di funzione in modo che le funzioni critiche non subiscano limitazioni.

In questo argomento sono descritti i concetti di simultaneità e la scalabilità delle funzioni in Lambda. Alla fine di questo argomento, sarà possibile capire come calcolare la simultaneità, visualizzare le

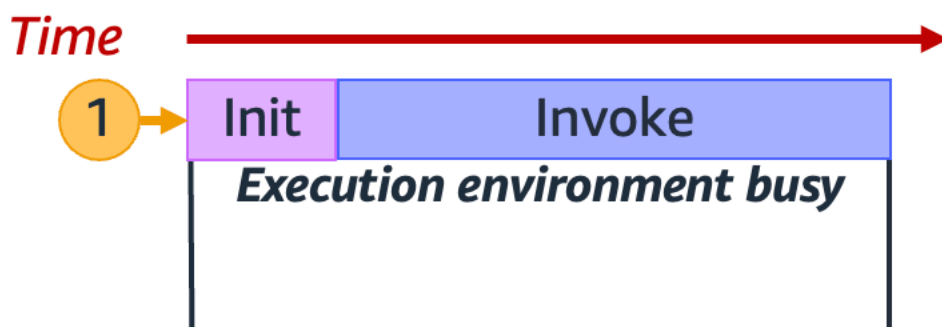
due principali opzioni di controllo della simultaneità (riservata e fornita), stimare le impostazioni di controllo della simultaneità appropriate e visualizzare i parametri per un'ulteriore ottimizzazione.

## Sections

- [Comprendere e visualizzare la simultaneità](#)
- [Calcolo della simultaneità per una funzione](#)
- [Informazioni sulla simultaneità riservata e simultaneità fornita](#)
- [Informazioni sulla simultaneità e le richieste al secondo](#)
- [Quote di simultaneità](#)
- [Configurazione della simultaneità riservata per una funzione](#)
- [Configurazione della simultaneità fornita per una funzione](#)
- [Comportamento del dimensionamento Lambda](#)
- [Monitoraggio della simultaneità](#)

## Comprendere e visualizzare la simultaneità

Lambda richiama la funzione in un [ambiente di esecuzione](#) sicuro e isolato. Per gestire una richiesta, Lambda deve prima inizializzare un ambiente di esecuzione (la [fase Init](#)) e poi utilizzare tale ambiente per richiamare la funzione (la [fase Invoke](#)):

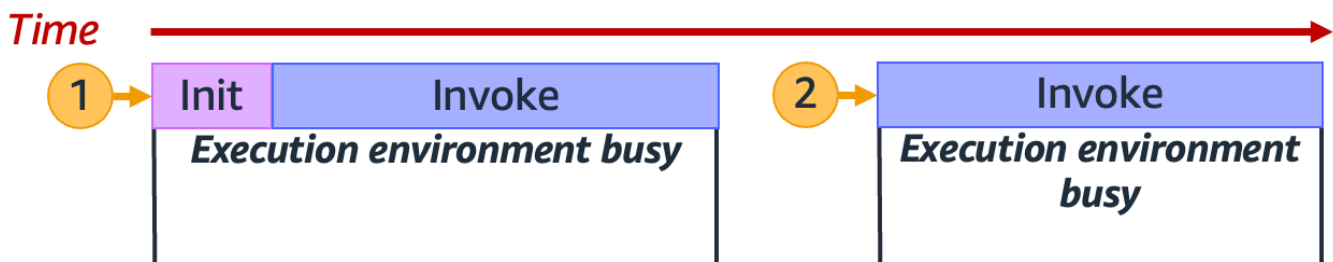


### Note

Le durate effettive delle fasi Init e Invoke possono variare in base a molti fattori, come il runtime scelto e il codice della funzione Lambda. Il diagramma precedente non intende rappresentare le proporzioni esatte delle durate delle fasi Init e Invoke.

Il rettangolo nel diagramma rappresenta un singolo ambiente di esecuzione. Quando la funzione riceve la sua primissima richiesta (rappresentata dal cerchio giallo con etichetta 1), Lambda crea un nuovo ambiente di esecuzione ed esegue il codice all'esterno del gestore principale durante la fase Init. Quindi, esegue il codice del gestore principale della funzione durante la fase Invoke. Durante l'intero processo, questo ambiente di esecuzione è occupato e non può elaborare altre richieste.

Quando Lambda termina l'elaborazione della prima richiesta, questo ambiente di esecuzione potrà elaborare richieste aggiuntive per la stessa funzione. Per le richieste successive, Lambda non deve reinizializzare l'ambiente.

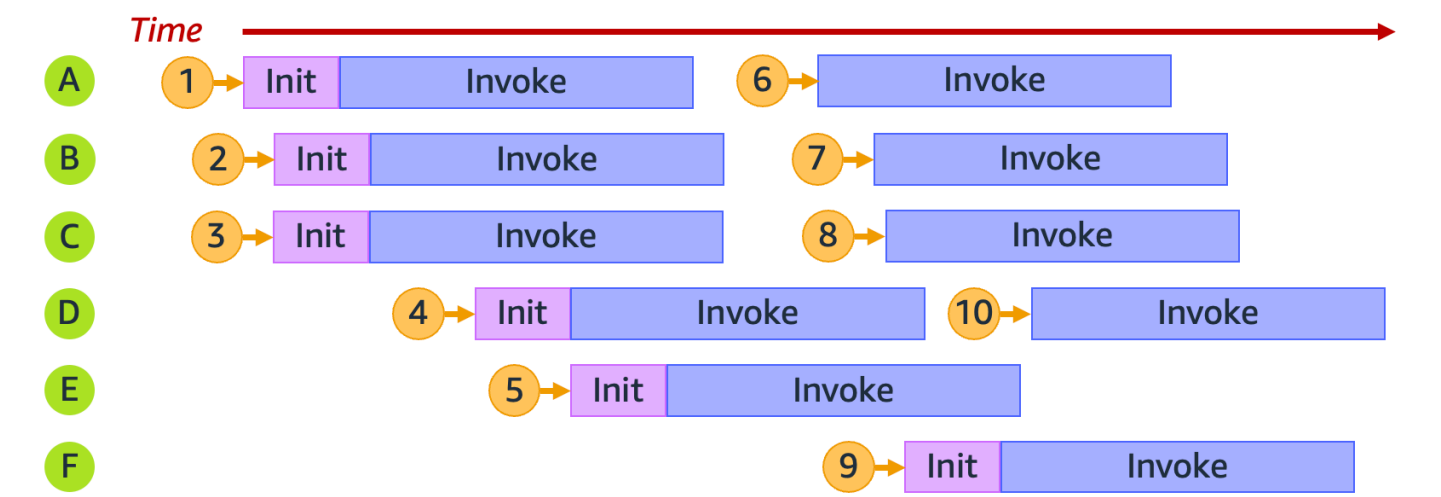


Nel diagramma precedente, Lambda riutilizza l'ambiente di esecuzione per gestire la seconda richiesta (rappresentata dal cerchio giallo con etichetta 2).

Finora ci siamo concentrati su una sola istanza dell'ambiente di esecuzione (ossia una simultaneità pari a 1). In pratica, Lambda potrebbe dover eseguire il provisioning di più istanze dell'ambiente di esecuzione in parallelo per gestire tutte le richieste in arrivo. Quando la funzione riceve una nuova richiesta, può succedere una delle due cose:

- Se è disponibile un'istanza dell'ambiente di esecuzione pre-inizializzata, Lambda la utilizza per elaborare la richiesta.
- Altrimenti, Lambda crea una nuova istanza dell'ambiente di esecuzione.

Ad esempio, vediamo cosa accade quando la funzione riceve 10 richieste:



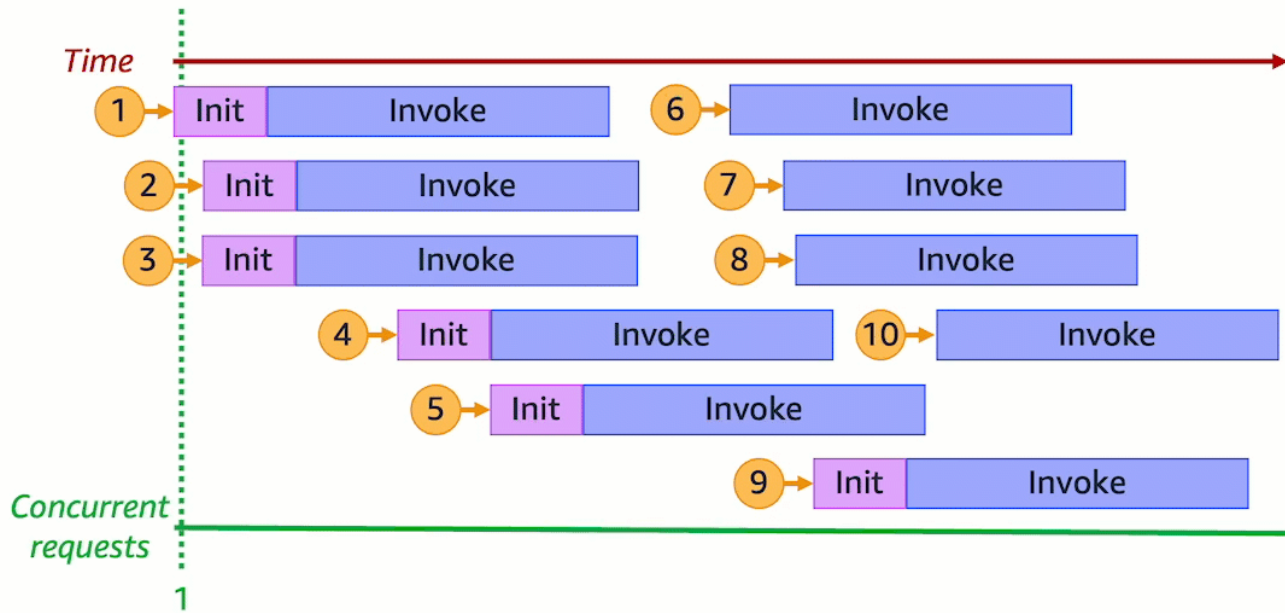
Nel diagramma precedente, ogni piano orizzontale rappresenta una singola istanza dell'ambiente di esecuzione (etichettata da A a F). Ecco come Lambda gestisce ogni richiesta:

| Richiesta | Comportamento di Lambda      | Ragionamento                                                                            |
|-----------|------------------------------|-----------------------------------------------------------------------------------------|
| 1         | Fornisce il nuovo ambiente A | Questa è la prima richiesta; nessuna istanza dell'ambiente di esecuzione è disponibile. |
| 2         | Fornisce il nuovo ambiente B | L'istanza A dell'ambiente di esecuzione esistente è occupata.                           |
| 3         | Fornisce il nuovo ambiente C | Le istanze A e B dell'ambiente di esecuzione esistenti sono entrambe occupate.          |
| 4         | Fornisce il nuovo ambiente D | Le istanze A, B e C dell'ambiente di esecuzione esistenti sono tutte occupate.          |
| 5         | Fornisce il nuovo ambiente E | Le istanze A, B, C e D dell'ambiente di esecuzione esistenti sono tutte occupate.       |

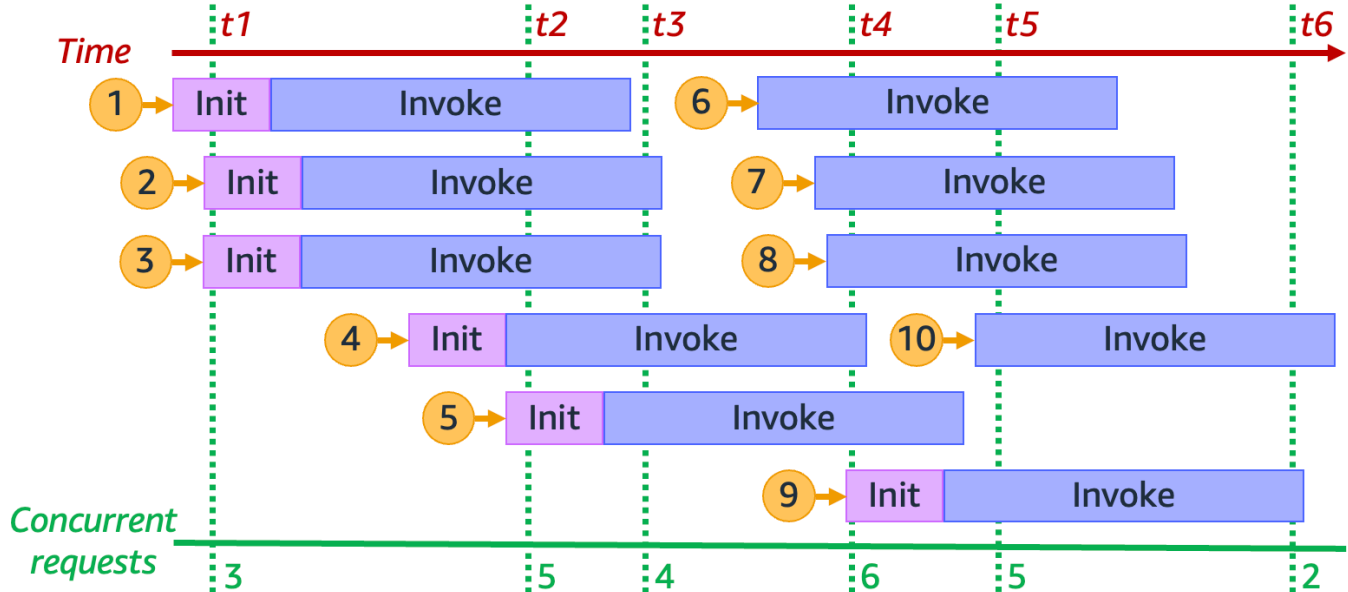
| Richiesta | Comportamento di Lambda      | Ragionamento                                                                                                |
|-----------|------------------------------|-------------------------------------------------------------------------------------------------------------|
| 6         | Riutilizza l'ambiente A      | L'istanza A dell'ambiente di esecuzione ha terminato l'elaborazione della richiesta 1 ed è ora disponibile. |
| 7         | Riutilizza l'ambiente B      | L'istanza B dell'ambiente di esecuzione ha terminato l'elaborazione della richiesta 2 ed è ora disponibile. |
| 8         | Riutilizza l'ambiente C      | L'istanza C dell'ambiente di esecuzione ha terminato l'elaborazione della richiesta 3 ed è ora disponibile. |
| 9         | Fornisce il nuovo ambiente F | Le istanze A, B, C, D e E dell'ambiente di esecuzione esistenti sono tutte occupate.                        |
| 10        | Riutilizza l'ambiente D      | L'istanza D dell'ambiente di esecuzione ha terminato l'elaborazione della richiesta 4 ed è ora disponibile. |

Man mano che la funzione riceve più richieste simultanee, Lambda aumenta il numero di istanze dell'ambiente di esecuzione in risposta. L'animazione seguente tiene traccia del numero di richieste simultanee nel tempo:





Osservando l'animazione precedente in sei momenti distinti nel tempo, otteniamo il seguente diagramma:



Nel diagramma precedente, possiamo tracciare una linea verticale in qualsiasi momento e contare il numero di ambienti che intersecano questa linea. Questo ci dà il numero di richieste simultanee in quel momento. Ad esempio, al momento  $t1$ , ci sono tre ambienti attivi che gestiscono tre richieste

simultanee. Il numero massimo di richieste simultanee in questa simulazione si verifica nel momento  $t_4$ , quando ci sono sei ambienti attivi che gestiscono sei richieste simultanee.

Per riassumere, la simultaneità della funzione è il numero di richieste simultanee che sono gestite nello stesso momento. In risposta a un aumento della simultaneità della funzione, Lambda fornisce più istanze dell'ambiente di esecuzione per soddisfare la domanda di richieste.

## Calcolo della simultaneità per una funzione

In generale, la simultaneità di un sistema è la capacità di elaborare più di un'attività contemporaneamente. In Lambda, la simultaneità è il numero di richieste in corso che la funzione può gestire nello stesso momento. Un modo rapido e pratico per misurare la simultaneità di una funzione Lambda consiste nell'utilizzare la seguente formula:

$$\text{Concurrency} = (\text{average requests per second}) * (\text{average request duration in seconds})$$

La simultaneità è diversa dalle richieste al secondo. Ad esempio, supponiamo che la funzione riceva in media 100 richieste al secondo. Se la durata media della richiesta è di 1 secondo, anche la simultaneità è 100:

$$\text{Concurrency} = (100 \text{ requests/second}) * (1 \text{ second/request}) = 100$$

Tuttavia, se la durata media della richiesta è di 500 ms, la simultaneità è 50:

$$\text{Concurrency} = (100 \text{ requests/second}) * (0.5 \text{ second/request}) = 50$$

Cosa significa in pratica una simultaneità pari a 50? Se la durata media della richiesta è di 500 ms, si può immaginare che un'istanza della funzione sia in grado di gestire 2 richieste al secondo. Quindi, sono necessarie 50 istanze della funzione per gestire un carico di 100 richieste al secondo. Una simultaneità pari a 50 significa che Lambda deve fornire 50 istanze dell'ambiente di esecuzione per gestire in modo efficiente questo carico di lavoro senza alcuna limitazione. Ecco come esprimerlo in forma di equazione:

$$\text{Concurrency} = (100 \text{ requests/second}) / (2 \text{ requests/second}) = 50$$

Se la funzione riceve il doppio del numero di richieste (200 richieste al secondo), ma richiede solo la metà del tempo per elaborare ciascuna richiesta (250 ms), la simultaneità è ancora 50:

$$\text{Concurrency} = (200 \text{ requests/second}) * (0.25 \text{ second/request}) = 50$$

Verifica se hai capito come funziona la simultaneità

Supponiamo di avere una funzione che richiede, in media, 200 ms per essere eseguita. Durante i picchi di carico, si osservano 5.000 richieste al secondo. Qual è la simultaneità della tua funzione durante i picchi di carico?

Risposta

La durata media della funzione è di 200 ms o 0,2 secondi. Utilizzando la formula della simultaneità, a partire da questi numeri si ottiene una simultaneità pari a 1.000:

$$\text{Concurrency} = (5,000 \text{ requests/second}) * (0.2 \text{ seconds/request}) = 1,000$$

In alternativa, una durata media della funzione di 200 ms significa che la funzione può elaborare 5 richieste al secondo. Per gestire il carico di lavoro di 5.000 richieste al secondo, sono necessarie 1.000 istanze dell'ambiente di esecuzione. Pertanto, la simultaneità è 1.000:

$$\text{Concurrency} = (5,000 \text{ requests/second}) / (5 \text{ requests/second}) = 1,000$$

## Informazioni sulla simultaneità riservata e simultaneità fornita

Per impostazione predefinita, il tuo account ha un limite di simultaneità pari a 1.000 esecuzioni simultanee per tutte le funzioni in una regione. Le funzioni condividono questo pool di simultaneità 1.000 su richiesta. Le tue funzioni subiscono una limitazione (ovvero iniziano a eliminare le richieste) se esaurisci la concorrenza disponibile.

Alcune delle tue funzioni potrebbero essere più critiche di altre. Di conseguenza, potresti voler configurare le impostazioni di simultaneità in modo da garantire che le funzioni critiche ottengano la simultaneità di cui hanno bisogno. Lambda fornisce due tipi di controlli della simultaneità: la simultaneità riservata e la simultaneità fornita.

- Utilizza la concorrenza riservata per impostare il numero massimo e minimo di istanze simultanee per riservare una parte della concorrenza del tuo account per una funzione. Ciò è utile se non si desidera che altre funzioni occupino tutta la simultaneità non riservata disponibile. Quando una funzione ha la simultaneità riservata, nessun'altra funzione può utilizzare tale simultaneità.
- Usa la simultaneità fornita per pre-inizializzare una serie di istanze di ambiente per una funzione. Ciò è utile per ridurre le latenze di avviamento a freddo.

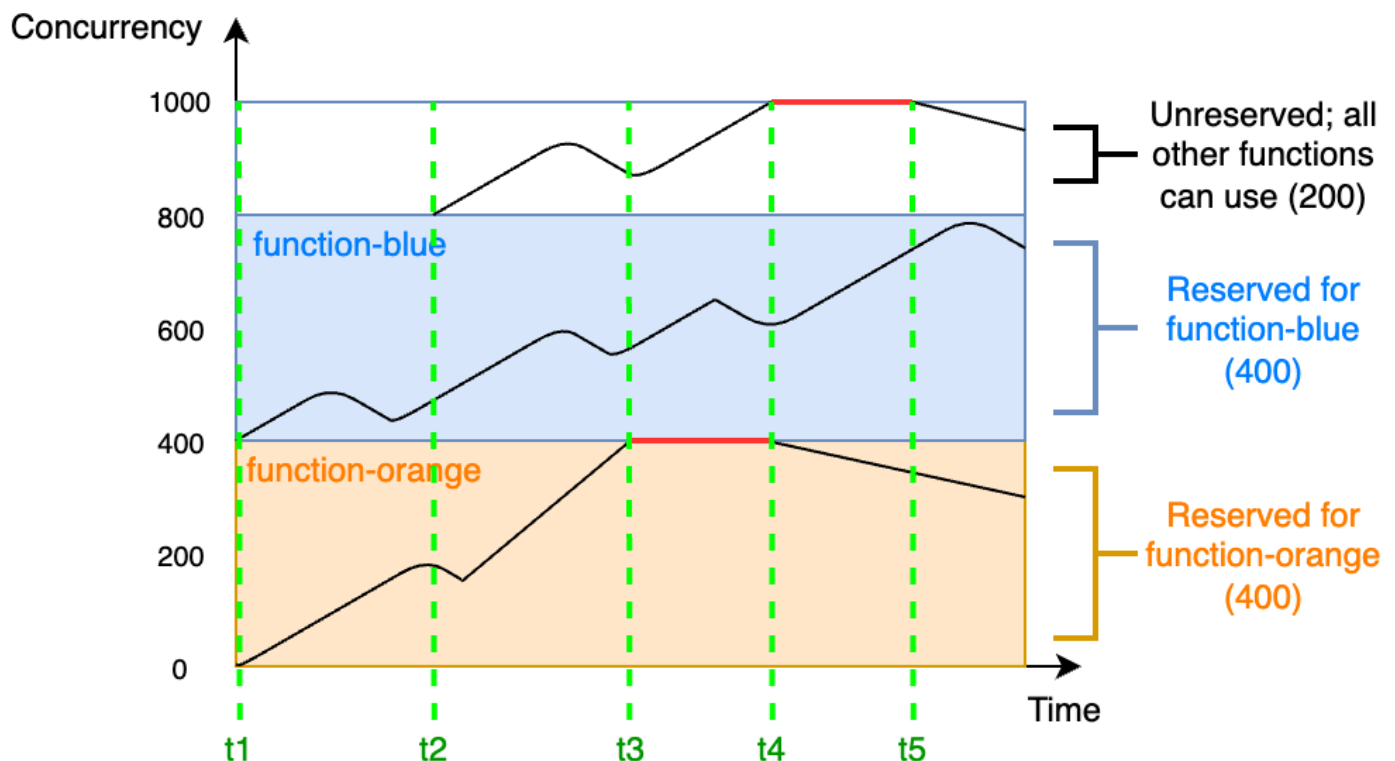
## Simultaneità riservata

Se vuoi garantire che una certa quantità di simultaneità sia disponibile per la tua funzione in qualsiasi momento, usa la simultaneità riservata.

La concorrenza riservata imposta il numero massimo e minimo di istanze simultanee che desideri allocare alla tua funzione. Quando una funzione ha la simultaneità riservata dedicata, nessun'altra funzione potrà utilizzare tale simultaneità. In altre parole, l'impostazione della simultaneità riservata può influire sul pool di simultaneità disponibile per altre funzioni. Le funzioni che non dispongono di simultaneità riservata condividono il pool rimanente di simultaneità non riservata.

La configurazione della simultaneità riservata viene conteggiata ai fini del limite complessivo di simultaneità dell'account. Non è previsto alcun addebito per la configurazione della simultaneità riservata per una funzione.

Per comprendere meglio la simultaneità riservata, considera il seguente diagramma:



In questo diagramma, il limite di simultaneità dell'account per tutte le funzioni in questa regione è il limite predefinito di 1.000. Supponiamo di avere due funzioni critiche function-blue e function-orange che si aspettino di ottenere regolarmente volumi di chiamate elevati. Decidi di assegnare 400

unità di simultaneità riservata a `function-blue` e 400 unità di simultaneità riservata a `function-orange`. In questo esempio, tutte le altre funzioni del tuo account dovranno condividere le restanti 200 unità di simultaneità non riservata.

Il diagramma presenta cinque punti di interesse:

- Su  $t_1$ , sia `function-orange` che `function-blue` iniziano a ricevere richieste. Ogni funzione inizia a utilizzare la parte allocata di unità di simultaneità riservata.
- Al momento  $t_2$ , `function-orange` e `function-blue` stanno ricevendo costantemente più richieste. Allo stesso tempo, vengono implementate altre funzioni Lambda, che iniziano a ricevere richieste. La simultaneità riservata non viene assegnata a queste altre funzioni. Iniziano a utilizzare le restanti 200 unità di simultaneità non riservata.
- Su  $t_3$ , `function-orange` raggiunge la simultaneità massima di 400. Sebbene sia presente una simultaneità inutilizzata altrove nel tuo account, `function-orange` non può accedervi. La linea rossa indica che `function-orange` è in corso una limitazione e Lambda potrebbe interrompere le richieste.
- Su  $t_4$ , `function-orange` inizia a ricevere meno richieste e non è più limitato. Tuttavia, le altre funzioni registrano un picco di traffico e iniziano a rallentare. Sebbene sia presente una simultaneità inutilizzata altrove nel tuo account, queste funzioni non possono accedervi. La linea rossa indica che le altre funzioni sono soggette a limitazioni.
- Su  $t_5$ , le altre funzioni iniziano a ricevere meno richieste e non sono più limitate.

Da questo esempio, nota che riservare la simultaneità ha i seguenti effetti:

- La funzione può essere dimensionata indipendentemente dalle altre funzioni nel tuo account. Tutte le funzioni dell'account nella stessa regione che non dispongono di simultaneità riservata condividono il pool di simultaneità non riservata. Senza simultaneità riservata, altre funzioni possono utilizzare potenzialmente tutta la simultaneità disponibile. Ciò impedisce alle funzioni critiche di aumentare quando necessario.
- La tua funzione non può essere aumentata orizzontalmente senza controllo. La concorrenza riservata limita la concorrenza massima e minima della funzione. Ciò significa che la funzione non può utilizzare la simultaneità riservata ad altre funzioni o la simultaneità dal pool non riservato. Inoltre, la concorrenza riservata funge sia da limite inferiore che superiore: riserva la capacità specificata esclusivamente per la funzione, impedendone al contempo il ridimensionamento oltre tale limite. È possibile riservare la simultaneità per evitare che la funzione utilizzi tutta la simultaneità disponibile nell'account oppure sovraccarichi le risorse in downstream.

- Potresti non essere in grado di utilizzare tutta la concorrenza disponibile del tuo account. La simultaneità di prenotazione viene conteggiata ai fini del limite di simultaneità dell'account, ma ciò significa anche che altre funzioni non possono utilizzare quella parte di simultaneità riservata. Se la tua funzione non utilizza tutta la simultaneità riservata, stai effettivamente sprecando quella simultaneità. Questo non è un problema a meno che altre funzioni del tuo account non possano trarre vantaggio dallo spreco di simultaneità.

Per gestire le impostazioni di simultaneità riservata per le tue funzioni, consulta la pagina [Configurazione della simultaneità riservata per una funzione](#).

## Simultaneità con provisioning

La simultaneità riservata viene utilizzata per definire il numero massimo di ambienti di esecuzione riservati a una funzione Lambda. Tuttavia, nessuno di questi ambienti è pre-inizializzato. Di conseguenza, le chiamate alle funzioni potrebbero richiedere più tempo perché Lambda deve prima inizializzare il nuovo ambiente prima di poterlo utilizzare per richiamare la funzione. Quando Lambda deve inizializzare un nuovo ambiente per eseguire una chiamata, si parla di avvio a freddo. [???](#) Per mitigare gli avviamenti a freddo, è possibile utilizzare la simultaneità fornita.

La simultaneità assegnata è il numero di ambienti di esecuzione pre-inizializzati che si desidera allocare alla funzione. Se si imposta la simultaneità fornita su una funzione, Lambda inizia quel numero di ambienti di esecuzione in modo che siano preparati a rispondere immediatamente alle richieste della funzione.

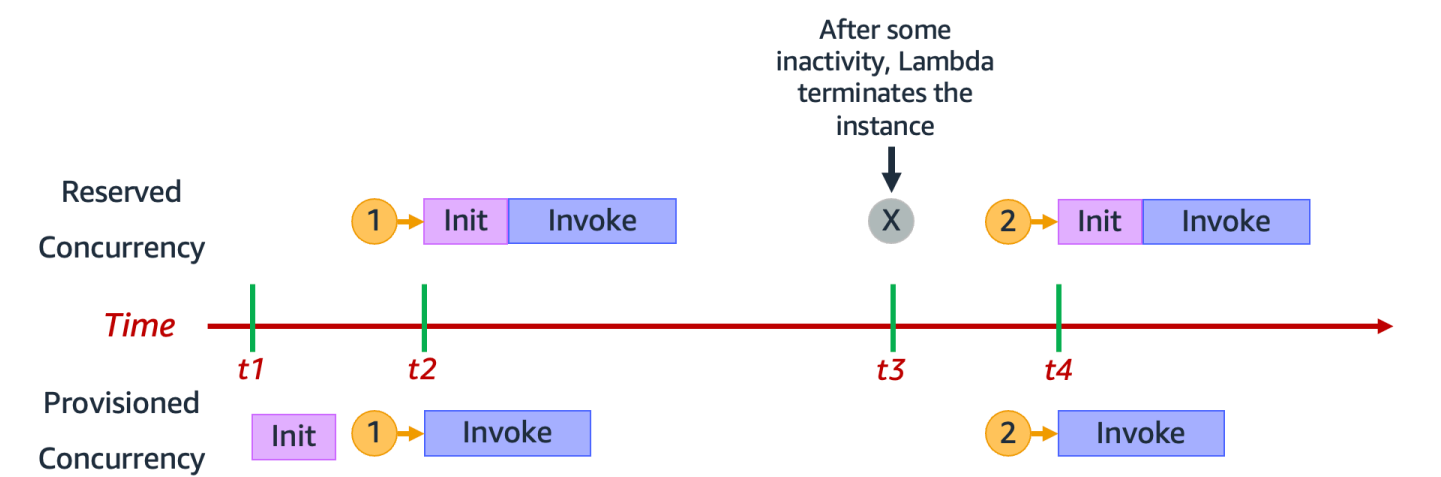
### Note

La configurazione della simultaneità fornita comporta addebiti sul tuo account. Se lavori con i runtime Java 11 o Java 17, puoi anche utilizzare Lambda SnapStart per mitigare i problemi di avvio a freddo senza costi aggiuntivi. SnapStart utilizza istantanee memorizzate nella cache dell'ambiente di esecuzione per migliorare significativamente le prestazioni di avvio. Non è possibile utilizzare entrambe le funzioni SnapStart e la concorrenza predisposta per la stessa versione della funzione. Per ulteriori informazioni su SnapStart funzionalità, limitazioni e regioni supportate, vedere. [Miglioramento delle prestazioni di avvio con Lambda SnapStart](#)

Quando si utilizza la simultaneità fornita, Lambda riavvia comunque gli ambienti di esecuzione in background. Ad esempio, ciò può verificarsi [dopo un errore di richiamo](#). Tuttavia, in qualsiasi momento, Lambda garantisce sempre che il numero di ambienti pre-inizializzati sia uguale al valore

dell'impostazione di simultaneità fornita dalla funzione. È importante sottolineare che, anche se si utilizza la concorrenza su provisioning, è comunque possibile riscontrare un ritardo nell'avvio a freddo se Lambda deve reimpostare l'ambiente di esecuzione.

Al contrario, quando si utilizza la concorrenza riservata, Lambda potrebbe chiudere completamente un ambiente dopo un periodo di inattività. Il diagramma seguente illustra ciò confrontando il ciclo di vita di un singolo ambiente di esecuzione quando si configura la funzione utilizzando la simultaneità riservata rispetto alla simultaneità fornita.

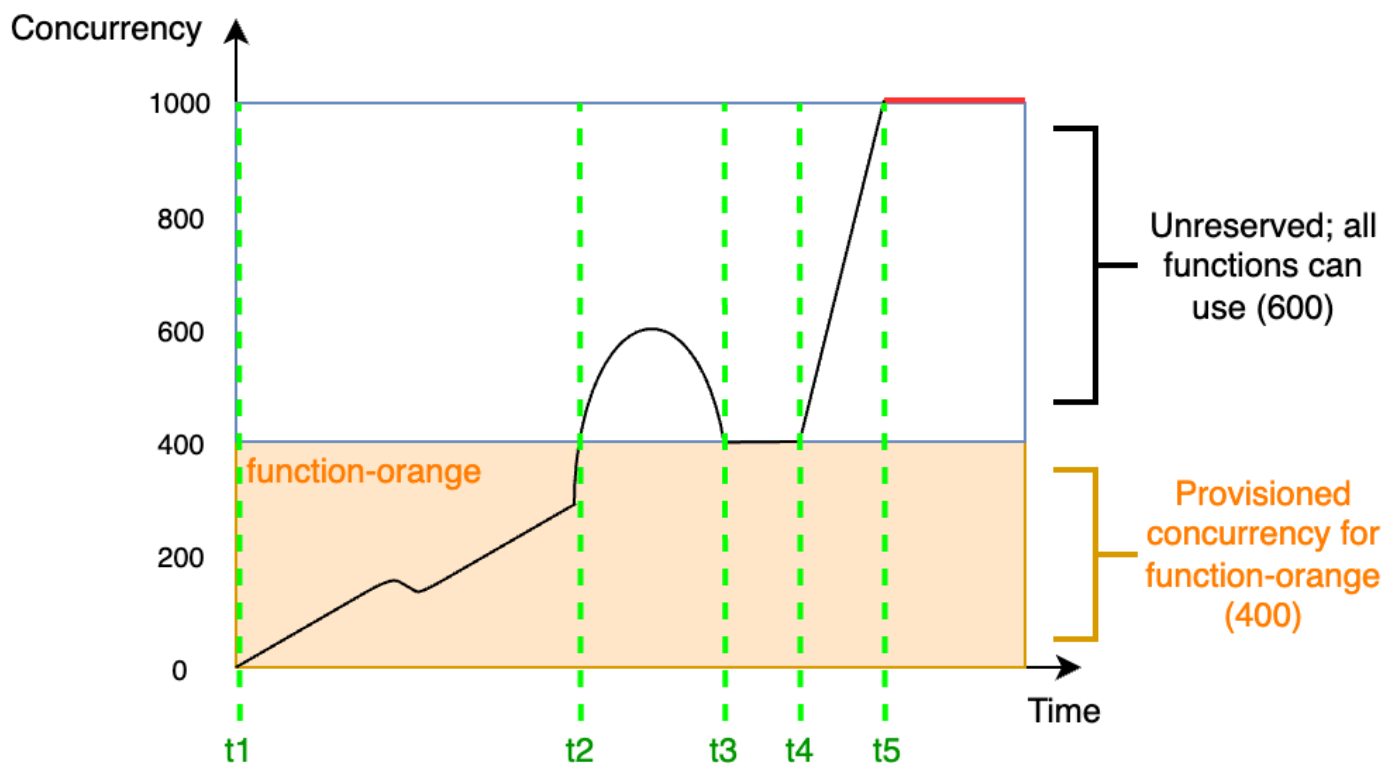


Il diagramma presenta 4 punti di interesse:

| Orario | Simultaneità riservata                                                                          | Simultaneità con provisioning                                                     |
|--------|-------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| t1     | Non succede niente.                                                                             | Lambda pre-inizializza un'istanza dell'ambiente di esecuzione.                    |
| t2     | Arriva la richiesta 1. Lambda deve inizializzare una nuova istanza dell'ambiente di esecuzione. | Arriva la richiesta 1. Lambda utilizza l'istanza pre-inizializzata dell'ambiente. |
| t3     | Dopo una certa inattività, Lambda termina l'istanza dell'ambiente attivo.                       | Non succede niente.                                                               |

| Orario | Simultaneità riservata                                                                          | Simultaneità con provisioning                                                     |
|--------|-------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| t4     | Arriva la richiesta 2. Lambda deve inizializzare una nuova istanza dell'ambiente di esecuzione. | Arriva la richiesta 2. Lambda utilizza l'istanza pre-inizializzata dell'ambiente. |

Per comprendere meglio la simultaneità fornita, considera il seguente diagramma:



In questo diagramma, hai un limite di simultaneità dell'account pari a 1.000. Decidi di assegnare 400 unità di simultaneità fornita a `function-orange`. Tutte le funzioni del tuo account, inclusa `function-orange`, possono utilizzare le restanti 600 unità di simultaneità non riservata.

Il diagramma presenta cinque punti di interesse:

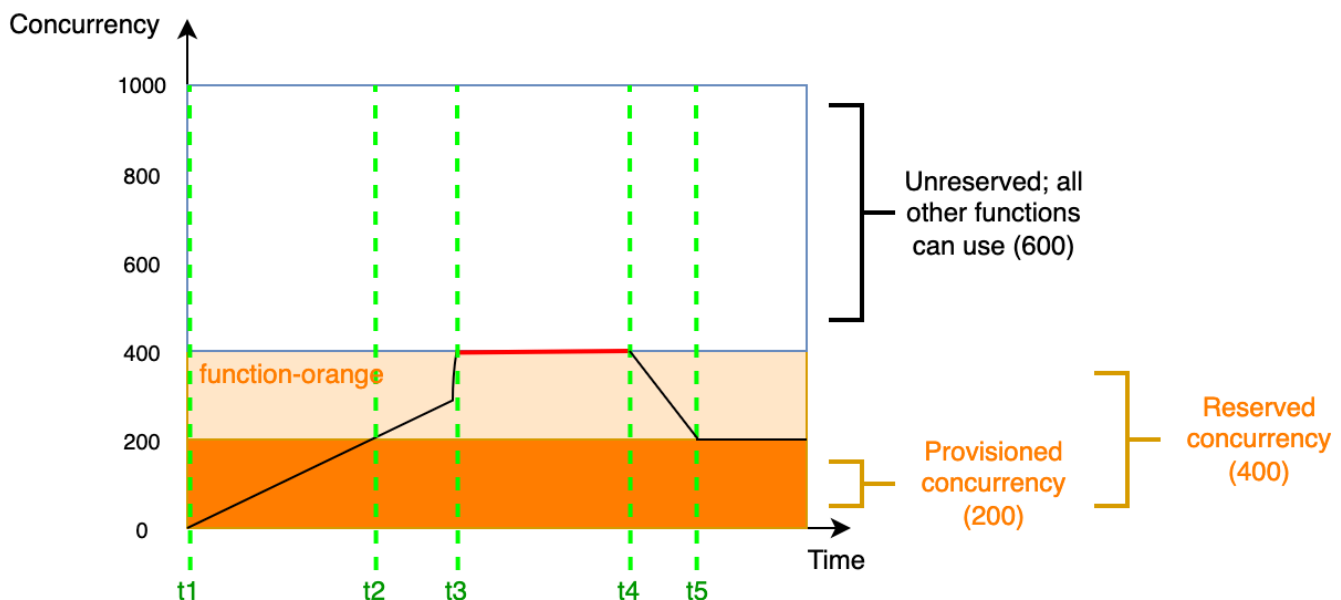
- Su t1, `function-orange` inizia a ricevere richieste. Poiché Lambda ha pre-inizializzato 400 istanze dell'ambiente di esecuzione, `function-orange` è pronta per la chiamata immediata.
- Su t2, `function-orange` raggiunge le 400 richieste simultanee. Di conseguenza, `function-orange` esaurisce la simultaneità fornita. Tuttavia, poiché è ancora disponibile la simultaneità non



riservata, Lambda può utilizzarla per gestire richieste aggiuntive a `function-orange` (senza limitazioni). Lambda deve creare nuove istanze per soddisfare queste richieste e la funzione potrebbe subire latenze di avvio a freddo.

- Su  $t_3$ , `function-orange` torna a 400 richieste simultanee dopo un breve picco di traffico. Lambda è nuovamente in grado di gestire tutte le richieste senza latenze di avvio a freddo.
- Su  $t_4$ , le funzioni del tuo account subiscono un'impennata di traffico. Questa impennata può provenire da `function-orange` o da qualsiasi altra funzione del tuo account. Per gestire queste richieste, Lambda utilizza la simultaneità non riservata.
- Su  $t_5$ , le funzioni del tuo account raggiungono il limite massimo di simultaneità pari a 1.000 e subiscono una limitazione.

L'esempio precedente considerava solo la simultaneità assegnata. In pratica, su una funzione è possibile impostare sia la simultaneità fornita che la simultaneità riservata. Ciò sarebbe possibile se si avesse una funzione che gestisce un carico costante di chiamate nei giorni feriali, ma registra regolarmente picchi di traffico durante i fine settimana. In questo caso, è possibile utilizzare la simultaneità fornita per impostare una quantità di base di ambienti per gestire le richieste durante i giorni feriali e utilizzare la simultaneità riservata per gestire i picchi del fine settimana. Considera il seguente diagramma:



In questo diagramma, supponiamo di configurare 200 unità di simultaneità fornita e 400 unità di simultaneità riservata per `function-orange`. Poiché è stata configurata la simultaneità riservata, `function-orange` non può utilizzare nessuna delle 600 unità di simultaneità non riservata.

Il diagramma presenta cinque punti di interesse:

- Su `t1`, `function-orange` inizia a ricevere richieste. Poiché Lambda ha pre-inizializzato 200 istanze dell'ambiente di esecuzione, `function-orange` è pronta per la chiamata immediata.
- Att2, `function-orange` utilizza tutta la concorrenza assegnata. `function-orange` può continuare a servire le richieste utilizzando la concorrenza riservata, ma queste richieste potrebbero presentare latenze di avvio a freddo.
- Su `t3`, `function-orange` raggiunge le 400 richieste simultanee. Di conseguenza, `function-orange` utilizza tutta la sua simultaneità riservata. Poiché `function-orange` non può utilizzare la simultaneità non riservata, le richieste iniziano a rallentare.
- Su `t4`, `function-orange` inizia a ricevere meno richieste e non è più limitato.
- Al momento `t5`, `function-orange` scende a 200 richieste simultanee, quindi tutte le richieste possono nuovamente utilizzare la simultaneità assegnata (ossia nessuna latenza di avvio a freddo).

Sia la simultaneità riservata che la simultaneità fornita vengono conteggiate ai fini del limite di simultaneità dell'account e delle [quote regionali](#). In altre parole, l'impostazione della simultaneità riservata e fornita può influire sul pool di simultaneità disponibile per altre funzioni. La configurazione della concorrenza su cui è stato effettuato il provisioning comporta costi a carico dell'utente. Account AWS

#### Note

Se la quantità di simultaneità assegnata nelle versioni e negli alias di una funzione si somma alla simultaneità riservata della funzione, tutte le chiamate vengono eseguite sulla simultaneità assegnata. Questa configurazione ha anche l'effetto di limitare il throttling della funzione (\$LATEST), che ne impedisce l'esecuzione. Non è possibile allocare più simultaneità fornita rispetto alla simultaneità riservata per una funzione.

Per gestire le impostazioni di simultaneità riservata delle tue funzioni, consulta la pagina [Configurazione della simultaneità fornita per una funzione](#). Per automatizzare il dimensionamento

della simultaneità assegnata in base a una pianificazione o all'utilizzo dell'applicazione, consulta la pagina [Utilizzo di Application Auto Scaling per automatizzare la gestione della simultaneità fornita](#).

## Come Lambda alloca la simultaneità fornita

La simultaneità fornita non è disponibile online immediatamente dopo la configurazione. Lambda avvia l'allocazione della simultaneità fornita dopo uno o due minuti di preparazione. Per ogni funzione, Lambda può fornire fino a 6.000 ambienti di esecuzione al minuto, indipendentemente da Regione AWS. È esattamente la stessa della [velocità di scalabilità simultanea](#) per le funzioni.

Quando invii una richiesta di allocazione della simultaneità fornita, non puoi accedere a nessuno di questi ambienti finché Lambda non completa l'allocazione. Ad esempio, se richiedi 5.000 simultaneità fornita, nessuna delle tue richieste potrà utilizzare la simultaneità fornita fino a che Lambda completa l'allocazione dei 5.000 ambienti di esecuzione.

## Simultaneità riservata e simultaneità fornita di Lambda.

Di seguito è riportata una tabella che riassume e mette a confronto la simultaneità riservata e la simultaneità assegnata.

| Topic                             | Simultaneità riservata                                                                           | Simultaneità con provisioning                                                                   |
|-----------------------------------|--------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| Definizione                       | Numero massimo di istanze dell'ambiente di esecuzione per la funzione.                           | Imposta il numero massimo di istanze dell'ambiente di esecuzione pre-fornito per la funzione.   |
| Comportamento del provisioning    | Lambda fornisce nuove istanze su richiesta.                                                      | Lambda fornisce le istanze in anticipo, ossia prima che la funzione inizi a ricevere richieste. |
| Comportamento dell'avvio a freddo | È possibile una latenza di avvio a freddo, poiché Lambda deve creare nuove istanze su richiesta. | La latenza di avvio a freddo non è possibile, poiché Lambda non deve creare istanze on demand.  |
| Comportamento della limitazione   | Funzione limitata quando viene raggiunto il limite di simultaneità riservata.                    | Se la simultaneità riservata non è impostata: quando viene raggiunto il limite di simultaneità  |

| Topic                                      | Simultaneità riservata                                                          | Simultaneità con provisioning                                                                                                                                                                                                                                                                 |
|--------------------------------------------|---------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                            |                                                                                 | <p>ità fornita la funzione utilizza la simultaneità non riservata.</p> <p>Se è impostata la simultaneità riservata: quando viene raggiunto il limite di simultaneità riservata la funzione viene limitata.</p>                                                                                |
| Comportamento predefinito se non impostato | La funzione utilizza la simultaneità non riservata disponibile nel tuo account. | <p>Lambda non fornisce alcuna istanza in anticipo. Invece, se la simultaneità riservata non è impostata: la funzione utilizza la simultaneità non riservata disponibile nel tuo account.</p> <p>Se è impostata la simultaneità riservata: la funzione utilizza la simultaneità riservata.</p> |
| Prezzi                                     | Nessun costo aggiuntivo.                                                        | Incorre in costi aggiuntivi.                                                                                                                                                                                                                                                                  |

## Informazioni sulla simultaneità e le richieste al secondo

Come indicato nella sezione precedente, la simultaneità è diversa dalle richieste al secondo. Questa distinzione è particolarmente rilevante quando si lavora con funzioni con una durata media della richiesta inferiore a 100 ms.

In tutte le funzioni del tuo account, Lambda applica un limite di richieste al secondo pari a 10 volte la simultaneità dell'account. Ad esempio, poiché il limite di simultaneità predefinito dell'account è 1.000, le funzioni dell'account possono gestire un massimo di 10.000 richieste al secondo.

Ad esempio, consideriamo una funzione con una durata media della richiesta di 50 ms. A 20.000 richieste al secondo, ecco la simultaneità di questa funzione:

$$\text{Concurrency} = (20,000 \text{ requests/second}) * (0.05 \text{ second/request}) = 1,000$$

In base a questo risultato, potresti aspettarti che il limite di simultaneità dell'account pari a 1.000 sia sufficiente per gestire questo carico. Tuttavia, a causa del limite di 10.000 richieste al secondo, la funzione può gestire solo 10.000 richieste al secondo sulle 20.000 richieste totali. Questa funzione subisce una limitazione.

Pertanto, quando si configurano le impostazioni di simultaneità per le funzioni, è necessario considerare sia la simultaneità sia le richieste al secondo. In questo caso, devi richiedere un aumento del limite di simultaneità dell'account a 2.000, poiché ciò aumenterebbe il limite totale di richieste al secondo a 20.000.

#### Note

In base a questo limite di richieste al secondo, non è corretto affermare che ogni ambiente di esecuzione Lambda può gestire solo un massimo di 10 richieste al secondo. Invece di osservare il carico su ogni singolo ambiente di esecuzione, Lambda considera solo la simultaneità complessiva e le richieste complessive al secondo nel calcolo delle quote.

Metti alla prova la tua comprensione della simultaneità (funzioni inferiori a 100 ms)

Supponiamo di avere una funzione che richiede, in media, 20 ms per essere eseguita. Durante i picchi di carico, si registrano 30.000 richieste al secondo. Qual è la simultaneità della tua funzione durante i picchi di carico?

Risposta

La durata media della funzione è di 20 ms o 0,02 secondi. Utilizzando la formula della simultaneità, a partire da questi numeri si ottiene una simultaneità pari a 600:

$$\text{Concurrency} = (30,000 \text{ requests/second}) * (0.02 \text{ seconds/request}) = 600$$

Per impostazione predefinita, il limite di simultaneità dell'account pari a 1.000 sembra sufficiente per gestire questo carico. Tuttavia, il limite di 10.000 richieste al secondo non è sufficiente per gestire le 30.000 richieste in entrata al secondo. Per soddisfare appieno le 30.000 richieste, devi richiedere un aumento del limite di simultaneità dell'account a 3.000 o più.

Il limite di richieste al secondo si applica a tutte le quote in Lambda che prevedono la simultaneità. In altre parole, si applica alle funzioni on demand sincrone, alle funzioni che utilizzano la simultaneità fornita e al [comportamento di scalabilità della simultaneità](#). Ad esempio, ecco alcuni scenari in cui è necessario considerare attentamente sia i limiti di simultaneità che quelli di richieste al secondo:

- Una funzione che utilizza la simultaneità on demand può registrare un aumento improvviso di 500 richieste simultanee ogni 10 secondi o di 5.000 richieste al secondo ogni 10 secondi, a seconda dell'evento che si verifica per primo.
- Si supponga di disporre di una funzione con un'allocazione di simultaneità fornita pari a 10. Questa funzione si ripercuote sulla simultaneità on demand dopo 10 operazioni simultanee o 100 richieste al secondo, a seconda dell'evento che si verifica per primo.

## Quote di simultaneità

Lambda imposta le quote per la quantità totale di simultaneità che è possibile utilizzare in tutte le funzioni di una regione. Queste quote esistono su due livelli:

- A livello di account: per impostazione predefinita, le funzioni possono avere fino a 1.000 unità di simultaneità. Per aumentare questo limite, consulta [Richiesta di aumento delle quote](#) nella Guida per l'utente di Service Quotas.
- A livello di funzione, per impostazione predefinita, è possibile riservare fino a 900 unità di simultaneità in tutte le regioni. Indipendentemente dal limite totale di simultaneità dell'account, Lambda riserva sempre 100 unità di simultaneità per le funzioni che non la riservano esplicitamente. Ad esempio, se hai aumentato il limite di simultaneità del tuo account a 2.000, puoi riservare fino a 1.900 unità di simultaneità a livello di funzione.
- Sia a livello di account che a livello di funzione, Lambda impone anche un limite di richieste al secondo pari a 10 volte la quota di simultaneità corrispondente. Ad esempio, ciò si applica alla simultaneità a livello di account, alle funzioni che utilizzano la simultaneità on demand, alle funzioni che utilizzano la simultaneità fornita e al [comportamento di scalabilità della simultaneità](#). Per ulteriori informazioni, consulta [the section called “Informazioni sulla simultaneità e le richieste al secondo”](#).

Per verificare la quota di concorrenza a livello di account corrente, usa AWS Command Line Interface (AWS CLI) per eseguire il seguente comando:

```
aws lambda get-account-settings
```

L'output restituito dovrebbe essere simile al seguente:

```
{
  "AccountLimit": {
```

```
    "TotalCodeSize": 80530636800,  
    "CodeSizeUnzipped": 262144000,  
    "CodeSizeZipped": 52428800,  
    "ConcurrentExecutions": 1000,  
    "UnreservedConcurrentExecutions": 900  
  },  
  "AccountUsage": {  
    "TotalCodeSize": 410759889,  
    "FunctionCount": 8  
  }  
}
```

`ConcurrentExecutions` è la quota di simultaneità totale a livello di account.

`UnreservedConcurrentExecutions` è la quantità di simultaneità riservata che puoi ancora destinare alle tue funzioni.

Man mano che la funzione riceve altre richieste, Lambda aumenta automaticamente il numero di ambienti di esecuzione per gestire le richieste fino al raggiungimento della quota di simultaneità dell'account. Tuttavia, per proteggersi dall'eccessivo aumento in risposta a improvvisi picchi di traffico, Lambda limita la velocità di dimensionamento delle funzioni. Questa velocità di scalabilità della simultaneità è la velocità massima alla quale le funzioni del tuo account possono scalare in risposta all'aumento delle richieste. Si tratta della velocità con cui Lambda può creare nuovi ambienti di esecuzione. Il tasso di scalabilità della simultaneità è diverso dal limite di simultaneità a livello di account, che è la quantità totale di simultaneità disponibile per le funzioni.

In ciascuna Regione AWS e per ogni funzione, il tasso di scalabilità della concorrenza è di 1.000 istanze dell'ambiente di esecuzione ogni 10 secondi (o 10.000 richieste al secondo ogni 10 secondi). In altre parole, ogni 10 secondi, Lambda può allocare al massimo 1.000 istanze aggiuntive dell'ambiente di esecuzione o gestire 10.000 richieste aggiuntive al secondo, per ciascuna delle tue funzioni.

Di solito, non è necessario preoccuparsi di questa limitazione. La velocità di dimensionamento di Lambda è sufficiente per la maggior parte dei casi d'uso.

È importante sottolineare che la velocità di scalabilità della simultaneità è un limite a livello di funzione. Ciò significa che ogni funzione del tuo account può dimensionarsi indipendentemente dalle altre funzioni.

Per ulteriori informazioni sul comportamento di dimensionamento, consulta la pagina [Comportamento del dimensionamento Lambda](#).

## Configurazione della simultaneità riservata per una funzione

In Lambda, la [simultaneità](#) è il numero di richieste in transito che la funzione sta gestendo attualmente. Sono disponibili due tipi di controlli di simultaneità:

- **Simultaneità riservata:** rappresenta il numero massimo e minimo di istanze simultanee allocate alla funzione. Quando una funzione ha la simultaneità riservata, nessun'altra funzione può utilizzare tale simultaneità. La simultaneità riservata è utile per garantire che le funzioni più critiche abbiano sempre una simultaneità sufficiente per gestire le richieste in arrivo. Inoltre, la simultaneità riservata può essere utilizzata per limitare la simultaneità ed evitare un sovraccarico di risorse a valle, come le connessioni al database. La simultaneità riservata funge sia da limite inferiore che da limite superiore: riserva la capacità specificata esclusivamente per la funzione, impedendone al contempo l'aumento oltre tale limite. La configurazione della simultaneità riservata per una funzione non comporta alcun addebito ulteriore.
- **Simultaneità fornita:** il numero di ambienti di esecuzione pre-inizializzati che desideri allocare alla funzione. Questi ambienti di esecuzione sono pronti a rispondere immediatamente alle richieste di funzioni in arrivo. La simultaneità fornita è utile per ridurre le latenze di avvio a freddo per le funzioni ed è progettata per rendere disponibili funzioni con tempi di risposta a due cifre in millisecondi. In genere, i carichi di lavoro interattivi traggono il massimo vantaggio da questa funzionalità. Si tratta di applicazioni in cui gli utenti avviano richieste, come le applicazioni Web e per dispositivi mobili, e sono le più sensibili alla latenza. I carichi di lavoro asincroni, come le pipeline di elaborazione dei dati, sono spesso meno sensibili alla latenza e quindi di solito non richiedono la simultaneità fornita. La configurazione della concorrenza predisposta comporta costi aggiuntivi per il tuo Account AWS

In questo argomento viene descritta in dettaglio la modalità gestire e configurare la simultaneità riservata. Per una panoramica concettuale di questi due tipi di controlli della simultaneità, consulta la sezione [Simultaneità riservata e simultaneità fornita](#). Per informazioni sulla configurazione della simultaneità fornita, consulta la sezione [the section called “Configurazione della simultaneità fornita”](#).

### Note

Le funzioni Lambda collegate a uno strumento di mappatura dell'origine degli eventi Amazon MQ hanno una simultaneità massima predefinita. Per Apache Active MQ, il numero massimo di istanze simultanee è 5. Per Rabbit MQ, il numero massimo di istanze simultanee è 1. L'impostazione della simultaneità sottoposta a provisioning o riservata per la funzione non



modifica questi limiti. Per richiedere un aumento della simultaneità massima predefinita quando si utilizza Amazon MQ, contatta Supporto.

## Sections

- [Configurazione della simultaneità riservata](#)
- [Stima accurata della simultaneità riservata richiesta per una funzione](#)

## Configurazione della simultaneità riservata

È possibile configurare le simultaneità fornita per una funzione utilizzando la console Lambda o l'API Lambda.

### Riserva della simultaneità per una funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione per la quale desideri prenotare la simultaneità.
3. Scegliere Configuration (Configurazione) e quindi scegliere Concurrency (Simultaneità).
4. In Concurrency (Concorrenza), scegliere Edit (Modifica).
5. Scegliere Reserve concurrency (Impegna concorrenza). Inserire l'importo della concorrenza da riservare per la funzione.
6. Scegli Save (Salva).

Puoi prenotare fino al valore di Simultaneità dell'account non riservata meno 100. Le restanti 100 unità di simultaneità sono destinate a funzioni che non utilizzano la simultaneità riservata. Ad esempio, se l'account ha un limite di simultaneità di 1.000, non puoi utilizzare tutte le 1.000 unità di simultaneità per una singola funzione.

## Edit concurrency

### Concurrency

Unreserved account concurrency: 0

- ☐ Use unreserved account concurrency
- ☒ Reserve concurrency

 The unreserved account concurrency can't go below 100.

Cancel

Save

La riserva della simultaneità per una funzione può influire sul pool di simultaneità disponibile per altre funzioni. Ad esempio, se riservi 100 unità di simultaneità per `function-a`, le altre funzioni del tuo account devono condividere le 900 unità di simultaneità rimanenti, anche se `function-a` non utilizza tutte le 100 unità di simultaneità riservata.

Per limitare intenzionalmente una funzione, imposta la simultaneità riservata su 0. In questo modo, viene interrotta la capacità della funzione di elaborare ulteriori eventi fino a quando il limite non viene rimosso.

Per configurare la simultaneità riservata con l'API Lambda, utilizza le operazioni dell'API seguenti.

- [PutFunctionConcurrency](#)
- [GetFunctionConcurrency](#)
- [DeleteFunctionConcurrency](#)

Ad esempio, per configurare la concorrenza riservata con AWS Command Line Interface (CLI), utilizzate il comando `put-function-concurrency`. Il comando seguente riserva 100 unità di simultaneità per una funzione denominata `my-function`:

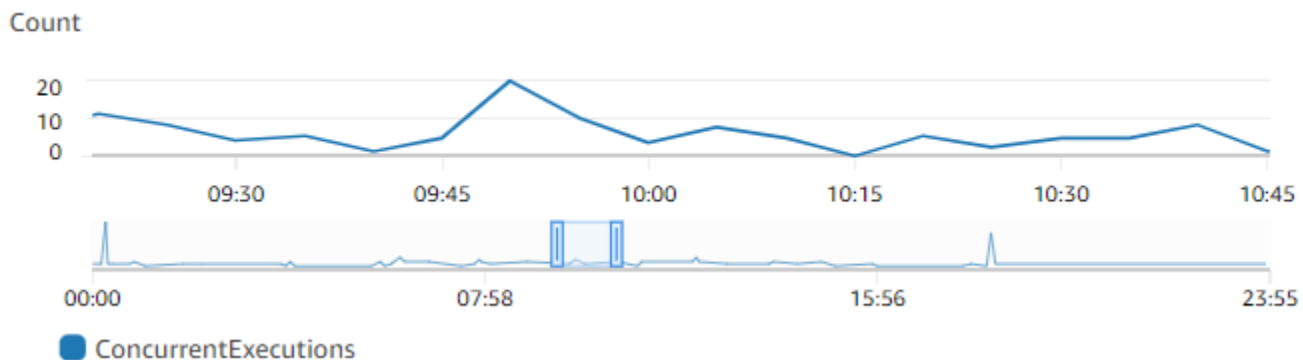
```
aws lambda put-function-concurrency --function-name my-function \  
--reserved-concurrent-executions 100
```

L'output restituito dovrebbe essere simile al seguente:

```
{
  "ReservedConcurrentExecutions": 100
}
```

## Stima accurata della simultaneità riservata richiesta per una funzione

Se la tua funzione sta attualmente servendo traffico, puoi facilmente visualizzarne le metriche di concorrenza utilizzando le metriche. [CloudWatch](#). In particolare, il parametro `ConcurrentExecutions` mostra il numero di chiamate simultanee per ciascuna funzione del tuo account.



Il grafico precedente mostra che questa funzione gestisce in media da 5 a 10 richieste simultanee in ogni momento e in un giorno tipico raggiunge un massimo di 20 richieste. Supponiamo che nel tuo account siano presenti molte altre funzioni. Se questa funzione è fondamentale per la tua applicazione e non vuoi che alcuna richiesta venga tralasciata, puoi utilizzare un numero uguale a o maggiore di 20 come impostazione della simultaneità riservata.

In alternativa, ricorda che è possibile [calcolare la simultaneità](#) anche utilizzando la seguente formula:

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

Moltiplicando le richieste medie al secondo per la durata media delle richieste in secondi, si ottiene una stima approssimativa della quantità di simultaneità che è necessario riservare. Puoi stimare le richieste medie al secondo utilizzando il parametro `Invocation` e la durata media delle richieste in secondi utilizzando il parametro `Duration`. Per ulteriori dettagli, consulta [Utilizzo delle CloudWatch metriche con Lambda](#).

Devi anche essere esperto dei vincoli di throughput a monte e a valle. Sebbene le funzioni Lambda siano scalabili senza problemi in base al carico, le dipendenze upstream e downstream potrebbero non avere le stesse capacità di throughput. Se devi limitare il valore massimo di scalabilità della tua funzione, puoi configurare la simultaneità riservata sulla funzione.

## Configurazione della simultaneità fornita per una funzione

In Lambda, la [simultaneità](#) è il numero di richieste in transito che la funzione sta gestendo attualmente. Sono disponibili due tipi di controlli di simultaneità:

- **Simultaneità riservata:** rappresenta il numero massimo e minimo di istanze simultanee allocate alla funzione. Quando una funzione ha la simultaneità riservata, nessun'altra funzione può utilizzare tale simultaneità. La simultaneità riservata è utile per garantire che le funzioni più critiche abbiano sempre una simultaneità sufficiente per gestire le richieste in arrivo. Inoltre, la simultaneità riservata può essere utilizzata per limitare la simultaneità ed evitare un sovraccarico di risorse a valle, come le connessioni al database. La simultaneità riservata funge sia da limite inferiore che da limite superiore: riserva la capacità specificata esclusivamente per la funzione, impedendone al contempo l'aumento oltre tale limite. La configurazione della simultaneità riservata per una funzione non comporta alcun addebito ulteriore.
- **Simultaneità fornita:** il numero di ambienti di esecuzione pre-inizializzati che desideri allocare alla funzione. Questi ambienti di esecuzione sono pronti a rispondere immediatamente alle richieste di funzioni in arrivo. La simultaneità fornita è utile per ridurre le latenze di avvio a freddo per le funzioni ed è progettata per rendere disponibili funzioni con tempi di risposta a due cifre in millisecondi. In genere, i carichi di lavoro interattivi traggono il massimo vantaggio da questa funzionalità. Si tratta di applicazioni in cui gli utenti avviano richieste, come le applicazioni Web e per dispositivi mobili, e sono le più sensibili alla latenza. I carichi di lavoro asincroni, come le pipeline di elaborazione dei dati, sono spesso meno sensibili alla latenza e quindi di solito non richiedono la simultaneità fornita. La configurazione della concorrenza predisposta comporta costi aggiuntivi per il tuo Account AWS

In questo argomento viene descritta in dettaglio la modalità gestire e configurare la simultaneità fornita. Per una panoramica concettuale di questi due tipi di controlli della simultaneità, consulta la sezione [Simultaneità riservata e simultaneità fornita](#). Per ulteriori informazioni sulla configurazione della simultaneità riservata, consulta la sezione [the section called “Configurazione della simultaneità riservata”](#).

### Note

Le funzioni Lambda collegate a uno strumento di mappatura dell'origine degli eventi Amazon MQ hanno una simultaneità massima predefinita. Per Apache Active MQ, il numero massimo di istanze simultanee è 5. Per Rabbit MQ, il numero massimo di istanze simultanee è 1. L'impostazione della simultaneità sottoposta a provisioning o riservata per la funzione non

modifica questi limiti. Per richiedere un aumento della simultaneità massima predefinita quando si utilizza Amazon MQ, contatta Supporto.

## Sections

- [Configurazione della simultaneità fornita](#)
- [Stima accurata della simultaneità fornita richiesta per una funzione](#)
- [Ottimizzazione del codice della funzione quando si utilizza la simultaneità fornita](#)
- [Utilizzo di variabili di ambiente per visualizzare e controllare il comportamento della simultaneità fornita](#)
- [Informazioni sul comportamento di registrazione e fatturazione con la simultaneità fornita](#)
- [Utilizzo di Application Auto Scaling per automatizzare la gestione della simultaneità fornita](#)

## Configurazione della simultaneità fornita

È possibile configurare le impostazioni di simultaneità fornita per una funzione utilizzando la console Lambda o l'API Lambda.

### Allocazione della simultaneità fornita per una funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione per la quale desideri allocare la simultaneità fornita.
3. Scegliere Configuration (Configurazione) e quindi scegliere Concurrency (Simultaneità).
4. In Provisioned concurrency configurations (Configurazioni di simultaneità fornita), scegliere Add configuration (Aggiungi configurazione).
5. Scegli il tipo di qualificatore e l'alias o la versione.

#### Note

Non è possibile utilizzare la simultaneità fornita con la versione \$LATEST di alcuna funzione.

Se la funzione dispone di un'origine eventi, assicurati che tale origine punti all'alias corretto o alla versione corretta della funzione. In caso contrario, la funzione non utilizzerà gli ambienti di simultaneità fornita.

6. Inserisci un numero in Simultaneità fornita.

## 7. Scegli Save (Salva).

Puoi configurare fino al valore massimo di Simultaneità dell'account non riservata meno 100. Le restanti 100 unità di simultaneità sono destinate a funzioni che non utilizzano la simultaneità riservata. Ad esempio, se il tuo account ha un limite di simultaneità di 1.000 e non hai assegnato simultaneità riservata o fornita ad alcuna delle tue altre funzioni, puoi configurare un massimo di 900 unità di simultaneità fornita per una singola funzione.

**Provisioned concurrency**

To enable your function to scale without fluctuations in latency, use provisioned concurrency. You can use Application Auto Scaling to automatically adjust provisioned concurrency to maintain a configured target utilization. Provisioned concurrency runs continually and has separate pricing for concurrency and execution duration. [Learn more](#)

**\$0.00 per month in addition to pricing for duration and requests.** [Pricing](#)

 The maximum allowed provisioned concurrency is 900, based on the unreserved concurrency available (1000) minus the minimum unreserved account concurrency (100).

900 available

 Please correct the errors above.

La configurazione della simultaneità fornita per una funzione influisce sul pool di simultaneità disponibile per altre funzioni. Ad esempio, se configuri 100 unità di simultaneità fornita per `function-a`, le altre funzioni nell'account devono condividere le 900 unità di simultaneità rimanenti. Ciò vale anche se `function-a` non utilizza tutte le 100 unità.

Per la stessa funzione è possibile allocare sia la simultaneità riservata che la simultaneità fornita. In questi casi, la concorrenza fornita non può superare la concorrenza riservata.

Questa limitazione si estende alle versioni della funzione. La simultaneità fornita massima che si può assegnare a una versione specifica della funzione corrisponde alla simultaneità riservata della funzione meno la simultaneità fornita su altre versioni della funzione.

Per configurare la simultaneità fornita con l'API Lambda, utilizza le operazioni dell'API seguenti.

- [PutProvisionedConcurrencyConfig](#)
- [GetProvisionedConcurrencyConfig](#)
- [ListProvisionedConcurrencyConfigs](#)
- [DeleteProvisionedConcurrencyConfig](#)

Ad esempio, per configurare la concorrenza predisposta con AWS Command Line Interface (CLI), usa il comando `put-provisioned-concurrency-config`. Il comando seguente alloca 100 unità di simultaneità fornita per l'alias BLUE di una funzione denominata `my-function`:

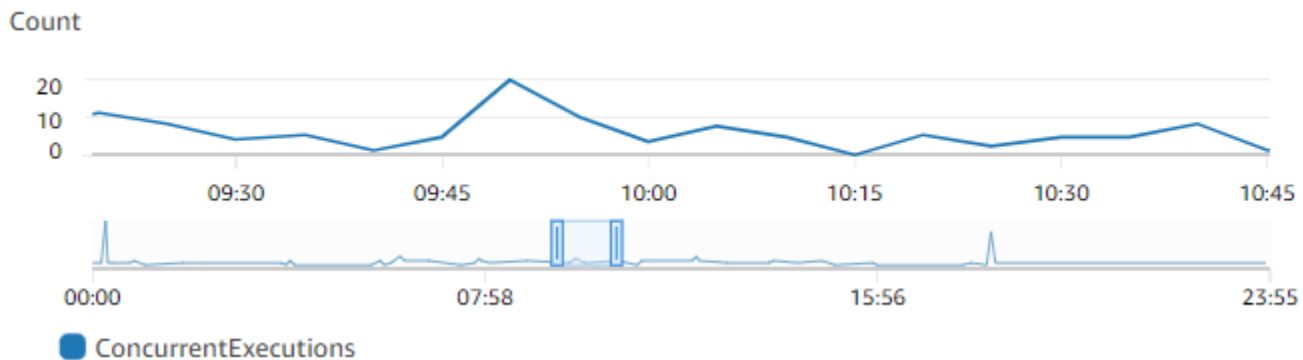
```
aws lambda put-provisioned-concurrency-config --function-name my-function \
  --qualifier BLUE \
  --provisioned-concurrent-executions 100
```

L'output restituito dovrebbe essere simile al seguente:

```
{
  "Requested ProvisionedConcurrentExecutions": 100,
  "Allocated ProvisionedConcurrentExecutions": 0,
  "Status": "IN_PROGRESS",
  "LastModified": "2023-01-21T11:30:00+0000"
}
```

## Stima accurata della simultaneità fornita richiesta per una funzione

È possibile visualizzare le metriche di concorrenza di qualsiasi funzione attiva utilizzando le metriche. [CloudWatch](#). Nello specifico, il parametro `ConcurrentExecutions` mostra il numero di invocazioni simultanee per le funzioni nell'account.



Il grafico precedente mostra che questa funzione gestisce in media da 5 a 10 richieste simultanee in qualsiasi momento dato, e ha un picco di 20 richieste. Supponiamo che nel tuo account siano presenti molte altre funzioni. Se questa funzione è fondamentale per la tua applicazione e necessiti di una risposta a bassa latenza per ogni invocazione, configura almeno 20 unità di simultaneità fornita.

Ricorda che puoi anche [calcolare la simultaneità](#) utilizzando la seguente formula:



$$\text{Concurrency} = (\text{average requests per second}) * (\text{average request duration in seconds})$$

Per stimare la quantità di simultaneità necessaria, moltiplica le richieste medie al secondo per la durata media delle richieste in secondi. Puoi stimare le richieste medie al secondo utilizzando il parametro `Invocation` e la durata media delle richieste in secondi utilizzando il parametro `Duration`.

Quando si configura la simultaneità fornita, Lambda suggerisce di aggiungere un buffer del 10% oltre alla quantità di simultaneità che generalmente occorre alla funzione. Ad esempio, se la funzione di solito raggiunge il picco di 200 richieste simultanee, imposta la simultaneità fornita su 220 (200 richieste simultanee + 10% = 220 unità di simultaneità fornita).

## Ottimizzazione del codice della funzione quando si utilizza la simultaneità fornita

Se utilizzi la simultaneità fornita, prendi in considerazione la possibilità di ristrutturare il codice della funzione per ottimizzarlo per una bassa latenza. Per le funzioni eseguite con simultaneità fornita, Lambda esegue qualsiasi codice di inizializzazione, come il caricamento di librerie e la creazione di istanze di client, in fase di allocazione. Pertanto, è consigliabile trasferire quanti più processi di inizializzazione di istanze all'esterno del gestore della funzione principale per evitare di influire sulla latenza durante le invocazioni effettive della funzione. Al contrario, l'inizializzazione di librerie o la creazione di istanze di client all'interno del codice principale dell'handler implica che la funzione deve ripetere l'esecuzione ogni volta che viene richiamata, a prescindere dal fatto che tu stia utilizzando la simultaneità fornita o meno.

Per le chiamate su richiesta, Lambda potrebbe dover eseguire nuovamente il codice di inizializzazione ogni volta che la funzione subisce un avvio a freddo. Per tali funzioni, puoi scegliere di posticipare l'inizializzazione di una funzionalità specifica fino a quando la funzione non ne avrà bisogno. Ad esempio, considera il seguente flusso di controllo per un gestore Lambda:

```
def handler(event, context):
    ...
    if ( some_condition ):
        // Initialize CLIENT_A to perform a task
    else:
        // Do nothing
```

Nell'esempio precedente, invece di inizializzare `CLIENT_A` all'esterno del gestore principale, lo sviluppatore lo ha inizializzato all'interno dell'istruzione `if`. In tal modo, Lambda esegue il codice

solo se la condizione `some_condition` è soddisfatta. Se esegui l'inizializzazione di `CLIENT_A` all'esterno del gestore principale, Lambda esegue quel codice a ogni avvio a freddo. Ciò può aumentare la latenza complessiva.

Puoi misurare gli avviamenti a freddo man mano che Lambda aumenta la scalabilità aggiungendo il X-Ray monitoraggio alla tua funzione. Una funzione che utilizza la concorrenza predisposta non presenta un comportamento di avvio a freddo poiché l'ambiente di esecuzione viene preparato prima dell'invocazione. Tuttavia, la concorrenza con provisioning deve essere applicata a una versione o alias <https://docs.aws.amazon.com/lambda/latest/dg/configuration-versions.html> specifico di una funzione, non alla versione `$LATEST`. Nei casi in cui continui a riscontrare un comportamento di avvio a freddo, assicurati di richiamare la versione dell'alias con la concorrenza assegnata configurata.

## Utilizzo di variabili di ambiente per visualizzare e controllare il comportamento della simultaneità fornita

È possibile che la funzione utilizzi tutta la simultaneità fornita. Per gestire il traffico in eccesso, Lambda utilizza istanze on demand. Per determinare il tipo di inizializzazione che Lambda ha utilizzato per un ambiente specifico, controlla il valore della variabile di ambiente `AWS_LAMBDA_INITIALIZATION_TYPE`.

Questa variabile ammette due valori possibili: `provisioned-concurrency` o `on-demand`. Il valore di `AWS_LAMBDA_INITIALIZATION_TYPE` è immutabile e rimane costante per tutta la durata dell'ambiente. Per verificare il valore di una variabile di ambiente nel codice della funzione, consulta [Recupero delle variabili di ambiente Lambda](#).

Se utilizzi il runtime `.NET 8`, puoi configurare la variabile di `AWS_LAMBDA_DOTNET_PREJIT` ambiente per migliorare la latenza delle funzioni, anche se non utilizzano la concorrenza assegnata. Il runtime `.NET` impiega la compilazione e l'inizializzazione lenta per ciascuna libreria a cui il codice effettua la chiamata per la prima volta. Di conseguenza, la prima chiamata di una funzione Lambda potrebbe richiedere più tempo rispetto a quelle successive. Per ovviare a questo problema, puoi scegliere tra tre valori per `AWS_LAMBDA_DOTNET_PREJIT`:

- **ProvisionedConcurrency**: Lambda esegue la compilazione JIT anticipata per tutti gli ambienti utilizzando la simultaneità fornita. Si tratta del valore di default.
- **Always**: Lambda esegue la compilazione JIT anticipata per ogni ambiente, anche se la funzione non utilizza la simultaneità fornita.
- **Never**: Lambda disattiva la compilazione JIT anticipata per tutti gli ambienti.

## Informazioni sul comportamento di registrazione e fatturazione con la simultaneità fornita

Per gli ambienti con simultaneità fornita, il codice di inizializzazione della funzione viene eseguito durante l'allocazione e periodicamente mentre Lambda ricicla le istanze attive dell'ambiente. Lambda ti addebita l'inizializzazione anche se l'istanza dell'ambiente non elabora mai una richiesta. La simultaneità fornita viene eseguita continuamente e prevede una fatturazione separata rispetto ai costi di inizializzazione e invocazione. Per maggiori dettagli, consulta [Prezzi di AWS Lambda](#).

Quando si configura una funzione Lambda con una concorrenza preimpostata, Lambda preinizializza l'ambiente di esecuzione in modo che sia disponibile prima delle richieste di richiamo. Lambda registra il campo [Init Duration della funzione in un evento di log ??? Platform-InitReport in formato](#) di registrazione JSON ogni volta che l'ambiente viene inizializzato. Per visualizzare questo evento di registro, configura almeno il livello di registro JSON. [??? INFO](#) Puoi anche utilizzare l'API di [telemetria](#) per utilizzare gli eventi della piattaforma in cui viene riportato il campo Init Duration.

## Utilizzo di Application Auto Scaling per automatizzare la gestione della simultaneità fornita

Puoi utilizzare Application Auto Scaling per gestire la simultaneità fornita in base a una pianificazione o all'utilizzo. Se la funzione riceve modelli di traffico prevedibili, utilizza il dimensionamento pianificato. Se desideri che la funzione mantenga una percentuale di utilizzo specifica, utilizza una policy di dimensionamento con monitoraggio degli obiettivi.

### Note

Se utilizzi Application Auto Scaling per gestire la simultaneità fornita dalla funzione, assicurati di [configurare prima un valore di simultaneità iniziale](#). Se alla funzione non è assegnato un valore di concorrenza iniziale, Application Auto Scaling potrebbe non gestire correttamente il ridimensionamento della funzione.

## Dimensionamento pianificato

Con Application Auto Scaling, puoi creare una pianificazione personalizzata in base alle variazioni di carico prevedibili. Per ulteriori informazioni ed esempi, consultate [Scheduled Scaling for Application Auto Scaling](#) nella Guida per l'utente di Application Auto Scaling e [Scheduling AWS Lambda Provisioned Concurrency per i picchi di utilizzo ricorrenti nel Compute Blog](#). AWS

## Monitoraggio degli obiettivi

Con il tracciamento degli obiettivi, Application Auto Scaling crea e gestisce una serie di allarmi in base alla definizione della politica di CloudWatch ridimensionamento. Quando questi allarmi si attivano, Application Auto Scaling regola automaticamente la quantità di ambienti allocati utilizzando la simultaneità fornita. Utilizza il monitoraggio degli obiettivi per le applicazioni che non presentano modelli di traffico prevedibili.

Per dimensionare la simultaneità fornita utilizzando il tracciamento degli obiettivi, utilizza le operazioni `RegisterScalableTarget` e `PutScalingPolicy` dell'API di Application Auto Scaling. Ad esempio, se utilizzi AWS Command Line Interface (CLI), segui questi passaggi:

1. Registrare l'alias di una funzione come target di dimensionamento. Nell'esempio seguente viene registrato l'alias BLUE di una funzione denominata `my-function`:

```
aws application-autoscaling register-scalable-target --service-namespace lambda \
--resource-id function:my-function:BLUE --min-capacity 1 --max-capacity 100 \
--scalable-dimension lambda:function:ProvisionedConcurrency
```

2. Applicare una policy di dimensionamento alla destinazione. Nell'esempio seguente viene configurato Application Auto Scaling per regolare la configurazione della simultaneità fornita per un alias per mantenerne l'utilizzo intorno al 70% (ma è possibile applicare qualsiasi valore compreso tra il 10% e il 90%).

```
aws application-autoscaling put-scaling-policy \
--service-namespace lambda \
--scalable-dimension lambda:function:ProvisionedConcurrency \
--resource-id function:my-function:BLUE \
--policy-name my-policy \
--policy-type TargetTrackingScaling \
--target-tracking-scaling-policy-configuration '{ "TargetValue":
0.7, "PredefinedMetricSpecification": { "PredefinedMetricType":
"LambdaProvisionedConcurrencyUtilization" } }'
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{
  "PolicyARN": "arn:aws:autoscaling:us-
east-2:123456789012:scalingPolicy:12266dbb-1524-xmpl-a64e-9a0a34b996fa:resource/lambda/
function:my-function:BLUE:policyName/my-policy",
```

```
"Alarms": [
  {
    "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmHigh-aed0e274-
xmpl-40fe-8cba-2e78f000c0a7",
    "AlarmARN": "arn:aws:cloudwatch:us-
east-2:123456789012:alarm:TargetTracking-function:my-function:BLUE-AlarmHigh-aed0e274-
xmpl-40fe-8cba-2e78f000c0a7"
  },
  {
    "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmLow-7e1a928e-
xmpl-4d2b-8c01-782321bc6f66",
    "AlarmARN": "arn:aws:cloudwatch:us-
east-2:123456789012:alarm:TargetTracking-function:my-function:BLUE-AlarmLow-7e1a928e-
xmpl-4d2b-8c01-782321bc6f66"
  }
]
```

Application Auto Scaling crea due allarmi in CloudWatch. Il primo allarme si attiva quando l'utilizzo della simultaneità fornita supera costantemente il 70%. In questo caso, Application Auto Scaling alloca più simultaneità fornita per ridurre l'utilizzo. Il secondo allarme si attiva quando l'utilizzo è costantemente inferiore al 63% (90% dell'obiettivo del 70%). In questo caso, Application Auto Scaling riduce la simultaneità di provisioning dell'alias.

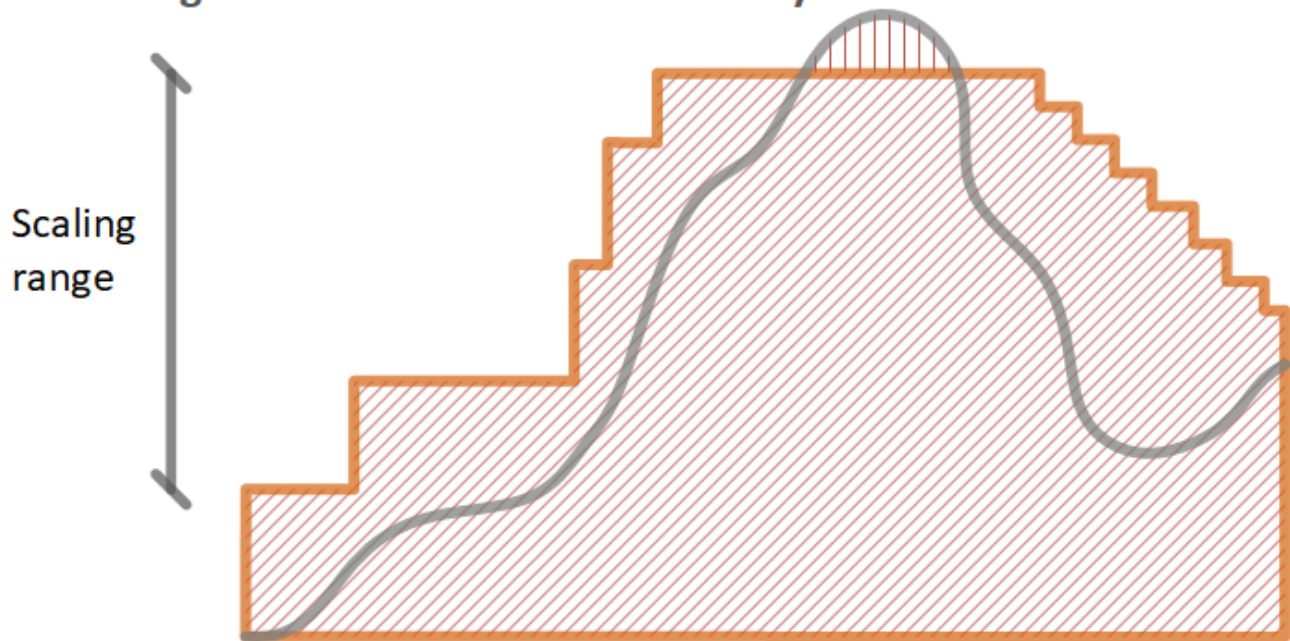
#### Note

Lambda emette la `ProvisionedConcurrencyUtilization` metrica solo quando la funzione è attiva e riceve richieste. Durante i periodi di inattività, non viene emessa alcuna metrica e gli allarmi con ridimensionamento automatico entrano in stato.

**INSUFFICIENT\_DATA** Di conseguenza, Application Auto Scaling non sarà in grado di regolare la concorrenza assegnata alla funzione. Ciò potrebbe comportare una fatturazione imprevista.

Nell'esempio seguente, una funzione si dimensiona tra una quantità minima e massima di concorrenza di cui è stato eseguito il provisioning in base all'utilizzo.

## Autoscaling with Provisioned Concurrency



### Legenda

-  Istanze di funzione
-  Richieste aperte
-  Simultaneità fornita
-  Simultaneità standard

Quando il numero di richieste aperte aumenta, Application Auto Scaling aumenta la simultaneità fornita a grandi scaglioni fino a raggiungere il massimo configurato. Successivamente, la funzione può continuare a dimensionarsi in base alla simultaneità non riservata standard se non hai raggiunto il limiti di simultaneità dell'account. Quando l'utilizzo scende e rimane basso, Application Auto Scaling riduce la simultaneità fornita a piccoli scaglioni periodici.

Entrambi gli allarmi di Application Auto Scaling utilizzano per impostazione predefinita la statistica media. Le funzioni che provocano rapide interruzioni di traffico potrebbero non attivare questi allarmi. Ad esempio, supponiamo che la funzione Lambda venga eseguita velocemente (ovvero in 20-100 ms) e che il traffico presenti picchi rapidi. In tal caso, il numero di richieste supera la simultaneità fornita allocata durante il picco.

Tuttavia, Application Auto Scaling richiede che il burst load venga mantenuto per almeno 3 minuti per eseguire il provisioning di ambienti aggiuntivi. Inoltre, entrambi gli CloudWatch allarmi richiedono 3 punti dati che raggiungano la media prevista per attivare la politica di scalabilità automatica. Se la funzione registra rapidi picchi di traffico, l'utilizzo della statistica Maximum anziché della statistica Average può essere più efficace nel dimensionare la simultaneità fornita per ridurre al minimo gli avvii a freddo.

Per ulteriori informazioni sulle policy di dimensionamento con monitoraggio degli obiettivi, consulta la sezione [Policy di dimensionamento con monitoraggio degli obiettivi per Application Auto Scaling](#).

## Comportamento del dimensionamento Lambda

Man mano che la funzione riceve altre richieste, Lambda aumenta automaticamente il numero di ambienti di esecuzione per gestire le richieste fino al raggiungimento della quota di simultaneità dell'account. Tuttavia, per proteggersi dall'eccessivo aumento in risposta a improvvisi picchi di traffico, Lambda limita la velocità di dimensionamento delle funzioni. Questa velocità di scalabilità della simultaneità è la velocità massima alla quale le funzioni del tuo account possono scalare in risposta all'aumento delle richieste. Si tratta della velocità con cui Lambda può creare nuovi ambienti di esecuzione. Il tasso di scalabilità della simultaneità è diverso dal limite di simultaneità a livello di account, che è la quantità totale di simultaneità disponibile per le funzioni.

### Velocità di dimensionamento della simultaneità

In ogni Regione AWS e per ogni funzione, la velocità di dimensionamento della simultaneità è di 1.000 istanze dell'ambiente di esecuzione ogni 10 secondi (o 10.000 richieste al secondo ogni 10 secondi). In altre parole, ogni 10 secondi, Lambda può allocare al massimo 1.000 istanze aggiuntive dell'ambiente di esecuzione o gestire 10.000 richieste aggiuntive al secondo, per ciascuna delle tue funzioni.

Di solito, non è necessario preoccuparsi di questa limitazione. La velocità di dimensionamento di Lambda è sufficiente per la maggior parte dei casi d'uso.

È importante sottolineare che la velocità di scalabilità della simultaneità è un limite a livello di funzione. Ciò significa che ogni funzione del tuo account può dimensionarsi indipendentemente dalle altre funzioni.

#### Note

In pratica, Lambda fa del suo meglio per ricaricare la velocità di dimensionamento della simultaneità in modo continuo nel tempo, anziché con una singola ricarica di 1.000 unità ogni 10 secondi.

Lambda non accumula porzioni inutilizzate della velocità di dimensionamento della simultaneità. Ciò significa che in qualsiasi momento, la velocità di dimensionamento è sempre al massimo di 1.000 unità di simultaneità. Ad esempio, se non utilizzi nessuna delle 1.000 unità di simultaneità disponibili in un intervallo di 10 secondi, non accumulerai 1.000 unità aggiuntive nel successivo intervallo di 10 secondi. La tua velocità di dimensionamento della simultaneità sarà ancora di 1.000 unità nel successivo intervallo di 10 secondi.



Finché la tua funzione continua a ricevere un numero crescente di richieste, Lambda si dimensionerà alla massima velocità disponibile, fino al limite di simultaneità del tuo account. [Puoi limitare la quantità di simultaneità che le singole funzioni possono utilizzare configurando la simultaneità riservata.](#)

Quando le richieste arrivano più velocemente della capacità di dimensionamento della funzione, oppure quando la funzione ha raggiunto la simultaneità massima, le altre richieste restituiscono esito negativo con un errore di limitazione della larghezza di banda della rete (429).

# Monitoraggio della simultaneità

Lambda emette CloudWatch parametri Amazon per aiutarti a monitorare la concorrenza per le tue funzioni. Questo argomento illustra questi parametri e spiega come interpretarli.

## Sections

- [Parametri generici di simultaneità](#)
- [Parametri della simultaneità fornita](#)
- [Lavorare con il parametro ClaimedAccountConcurrency](#)

## Parametri generici di simultaneità

Per monitorare la simultaneità delle funzioni Lambda, utilizza i seguenti parametri. La granularità di ogni parametro è di 1 minuto.

- **ConcurrentExecutions**: il numero di chiamate simultanee attive in un determinato momento. Lambda emette questo parametro per tutte le funzioni, gli alias e le versioni. Per qualsiasi funzione nella console Lambda, Lambda visualizza il grafico di **ConcurrentExecutions** in modo nativo nella scheda Monitoraggio, sotto Parametri. Visualizza questo parametro utilizzando MAX.
- **UnreservedConcurrentExecutions**: il numero di chiamate simultanee attive che utilizzano la simultaneità non riservata. Lambda emette questo parametro per tutte le funzioni in una regione. Visualizza questo parametro utilizzando MAX.
- **ClaimedAccountConcurrency**: la quantità di simultaneità che non è disponibile per le invocazioni on demand. **ClaimedAccountConcurrency** corrisponde a **UnreservedConcurrentExecutions** più la quantità di simultaneità allocata (ovvero la simultaneità totale riservata più la simultaneità totale fornita). Se **ClaimedAccountConcurrency** supera il limite di simultaneità dell'account, puoi [richiedere un limite di simultaneità più elevato](#).

Visualizza questo parametro utilizzando MAX. Per ulteriori informazioni, consulta [Lavorare con il parametro ClaimedAccountConcurrency](#).

## Parametri della simultaneità fornita

Per monitorare le funzioni Lambda che utilizzano la simultaneità fornita, utilizza i seguenti parametri. La granularità di ogni parametro è di 1 minuto.

- **ProvisionedConcurrentExecutions**: il numero di istanze dell'ambiente di esecuzione che stanno attivamente elaborando una chiamata con la simultaneità fornita. Lambda emette questo parametro per ogni versione e alias di funzione per cui sia configurata la simultaneità fornita. Visualizza questo parametro utilizzando MAX.

**ProvisionedConcurrentExecutions** non è uguale al numero totale di unità di simultaneità fornita allocate. Ad esempio, si supponga di allocare 100 unità di simultaneità fornita a una versione di funzione. In un dato minuto, se al massimo 50 di questi 100 ambienti di esecuzione gestivano le chiamate simultaneamente, il valore di MAX (**ProvisionedConcurrentExecutions**) è 50.

- **ProvisionedConcurrencyInvocations**: il numero di volte in cui Lambda richiama il codice di funzione tramite la simultaneità fornita. Lambda emette questo parametro per ogni versione e alias di funzione per cui sia configurata la simultaneità fornita. Visualizza questo parametro utilizzando SUM.

**ProvisionedConcurrencyInvocations** differisce da **ProvisionedConcurrentExecutions** per il fatto che **ProvisionedConcurrencyInvocations** conta il numero totale di chiamate, mentre **ProvisionedConcurrentExecutions** conta il numero di ambienti attivi. Per comprendere questa distinzione, esamina i seguenti scenari:



In questo esempio, supponiamo di ricevere 1 chiamata al minuto e che ogni chiamata richieda 2 minuti per essere completata. Ogni barra orizzontale arancione rappresenta una singola richiesta. Si supponga di allocare 10 unità di simultaneità fornita a questa funzione, in modo che ogni richiesta venga eseguita in base alla simultaneità fornita.

Tra i minuti 0 e 1, arriva la Request 1. Al minuto 1, il valore di MAX (ProvisionedConcurrentExecutions) è 1, poiché nell'ultimo minuto era attivo al massimo 1 ambiente di esecuzione. Anche il valore di SUM (ProvisionedConcurrencyInvocations) è 1, poiché nell'ultimo minuto è arrivata una nuova richiesta.

Tra i minuti 1 e 2, arriva la Request 2 mentre la Request 1 è ancora in esecuzione. Al minuto 2, il valore di MAX (ProvisionedConcurrentExecutions) è 2, poiché nell'ultimo minuto erano attivi al massimo 2 ambienti di esecuzione. Tuttavia, il valore di SUM (ProvisionedConcurrencyInvocations) è 1, poiché nell'ultimo minuto è arrivata solo una nuova richiesta. I parametri continuano a comportarsi in questo modo fino alla fine dell'esempio.

- **ProvisionedConcurrencySpilloverInvocations**: il numero di volte in cui Lambda richiama la funzione con la simultaneità (riservata o fornita) standard quando è in uso tutta la simultaneità fornita. Lambda emette questo parametro per ogni versione e alias di funzione per cui sia configurata la simultaneità fornita. Visualizza questo parametro utilizzando SUM. Il valore di ProvisionedConcurrencyInvocations + ProvisionedConcurrencySpilloverInvocations deve essere uguale al numero totale di chiamate delle funzioni (cioè il parametro Invocations).

**ProvisionedConcurrencyUtilization**: la percentuale di simultaneità fornita in uso (ossia il valore di ProvisionedConcurrentExecutions diviso per la quantità totale di simultaneità fornita allocata). Lambda emette questo parametro per ogni versione e alias di funzione per cui sia configurata la simultaneità fornita. Visualizza questo parametro utilizzando MAX.

Ad esempio, si supponga di allocare 100 unità di simultaneità fornita a una versione di funzione. In un dato minuto, se al massimo 60 di questi 100 ambienti di esecuzione gestivano le chiamate contemporaneamente, il valore di MAX (ProvisionedConcurrentExecutions) è 60 e il valore di MAX (ProvisionedConcurrencyUtilization) è 0,6.

Un valore elevato per ProvisionedConcurrencySpilloverInvocations potrebbe indicare che è necessario allocare una concorrenza aggiuntiva assegnata per la funzione. In alternativa, è possibile [configurare Application Auto Scaling per gestire il dimensionamento automatico della simultaneità fornita](#) in base a soglie predefinite.

Al contrario, valori costantemente bassi di per ProvisionedConcurrencyUtilization potrebbero indicare una sovraallocazione della concorrenza assegnata per la funzione.

## Lavorare con il parametro **ClaimedAccountConcurrency**

Lambda utilizza il parametro `ClaimedAccountConcurrency` per determinare la quantità di simultaneità disponibile nell'account per le invocazioni on demand. Lambda calcola `ClaimedAccountConcurrency` utilizzando la formula seguente:

$$\text{ClaimedAccountConcurrency} = \text{UnreservedConcurrentExecutions} + (\text{allocated concurrency})$$

`UnreservedConcurrentExecutions` corrisponde al numero di invocazioni simultanee attive che utilizzano la simultaneità non riservata. La simultaneità allocata è la somma delle due parti seguenti (sostituendo RC come "simultaneità riservata" e PC come "simultaneità fornita"):

- Il valore RC totale in tutte le funzioni di una Regione.
- Il valore PC totale in tutte le funzioni di una Regione che utilizzano PC, escluse le funzioni che utilizzano RC.

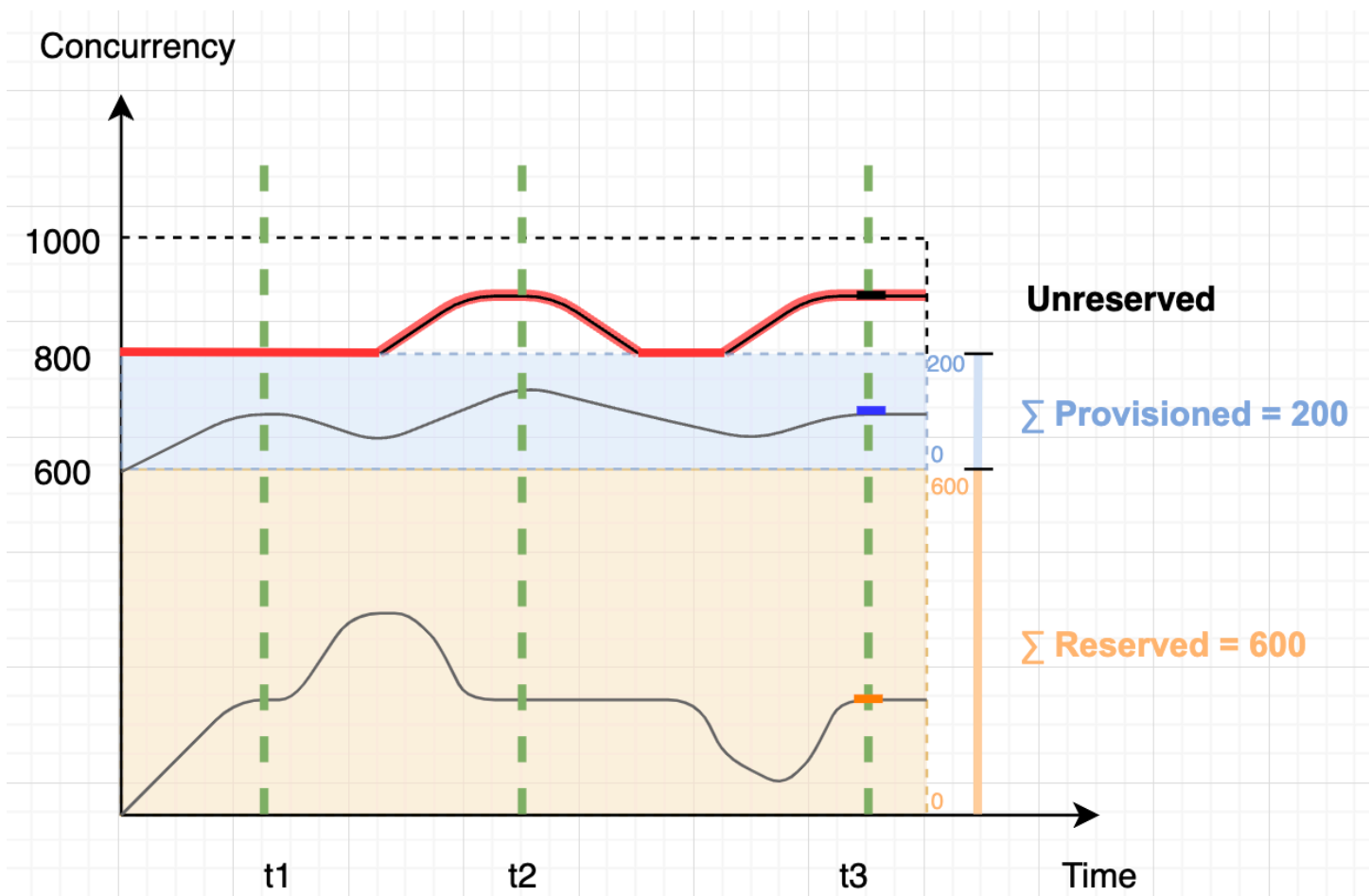
### Note

Non è possibile allocare più di quanto stanziato per una funzione. PC RC Pertanto, quella di una funzione RC è sempre maggiore o uguale alla sua PC. Per calcolare il contributo alla simultaneità allocata per tali funzioni con sia PC che RC, Lambda considera solo RC, che è il valore massimo tra i due.

Per determinare la quantità di simultaneità disponibile per le invocazioni on demand, Lambda utilizza il parametro `ClaimedAccountConcurrency` anziché `ConcurrentExecutions`. Sebbene sia utile per tenere traccia del numero di invocazioni simultanee attive, il parametro `ConcurrentExecutions` non sempre riflette la disponibilità di simultaneità effettiva. Questo perché Lambda considera anche la simultaneità riservata e la simultaneità fornita per determinare la disponibilità.

Per illustrare il funzionamento di `ClaimedAccountConcurrency`, immaginiamo uno scenario in cui configuri una grande quantità di simultaneità riservata e simultaneità fornita nelle funzioni che rimane per buona parte inutilizzata. Nell'esempio seguente, supponiamo che il limite di simultaneità dell'account sia 1.000 e che vi siano due funzioni principali nell'account: `function-orange` e `function-blue`. Assegna 600 unità di simultaneità riservata per `function-orange`. Assegna 200

unità di simultaneità fornita per function-blue. Supponiamo che, nel corso del tempo, implementi ulteriori funzioni e osservi lo schema di traffico seguente:



Nel diagramma precedente, le linee nere indicano l'uso effettivo della simultaneità nel tempo, mentre la linea rossa indica il valore di `ClaimedAccountConcurrency` nel tempo. In tutto lo scenario, il valore minimo di `ClaimedAccountConcurrency` è 800, nonostante lo scarso utilizzo effettivo di simultaneità nelle funzioni. Questo perché hai assegnato 800 unità totali di simultaneità per function-orange e function-blue. Dal punto di vista di Lambda, hai "rivendicato" questa simultaneità per l'uso, quindi ti rimangono in effetti solo 200 unità di simultaneità per le altre funzioni.

In questo scenario, la concorrenza allocata è 800 nella formula `ClaimedAccountConcurrency`. Possiamo quindi ricavare il valore di `ClaimedAccountConcurrency` in corrispondenza di vari punti del diagramma:

- In t1, `ClaimedAccountConcurrency` è 800 ( $800 + 0$  `UnreservedConcurrentExecutions`).
- In t2, `ClaimedAccountConcurrency` è 900 ( $800 + 100$  `UnreservedConcurrentExecutions`).

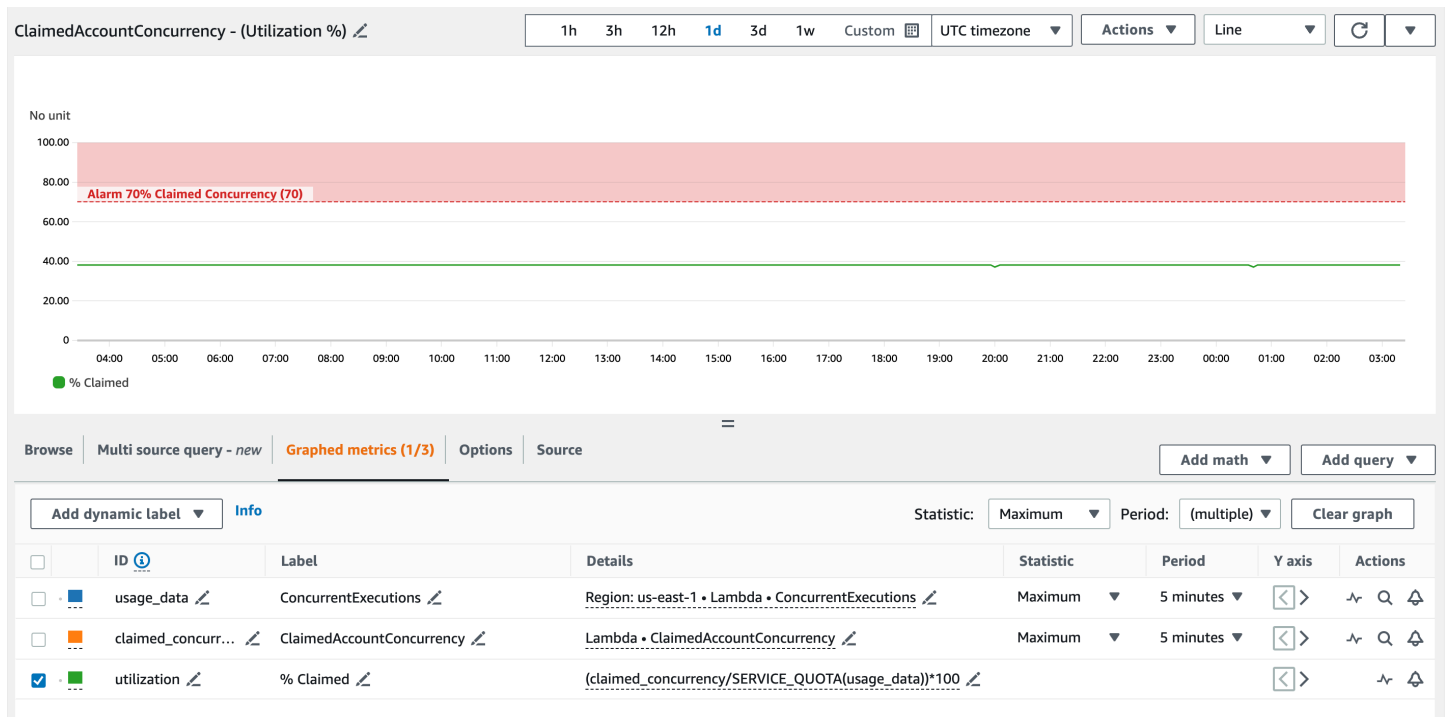
- In `t3`, `ClaimedAccountConcurrency` è di nuovo 900 (800 + 100 `UnreservedConcurrentExecutions`).

## Impostazione della **ClaimedAccountConcurrency** metrica in CloudWatch

Lambda emette la metrica in `ClaimedAccountConcurrency` CloudWatch. Utilizza questo parametro insieme al valore di `SERVICE_QUOTA(ConcurrentExecutions)` per ottenere la percentuale di utilizzo della simultaneità nel tuo account, come mostrato nella formula seguente:

$$\text{Utilization} = (\text{ClaimedAccountConcurrency} / \text{SERVICE\_QUOTA}(\text{ConcurrentExecutions})) * 100\%$$

La schermata seguente illustra come è possibile rappresentare graficamente questa formula in CloudWatch. La linea `claim_utilization` verde rappresenta l'utilizzo della simultaneità nell'account, che è pari a circa il 40%:



La schermata precedente include anche un CloudWatch allarme che si attiva quando l'utilizzo della concorrenza supera il 70%. Puoi utilizzare il parametro `ClaimedAccountConcurrency` insieme ad allarmi simili per determinare in modo proattivo quando potrebbe essere necessario richiedere un limite di simultaneità dell'account più elevato.

# Monitoraggio, debug e risoluzione dei problemi delle funzioni Lambda

AWS Lambda si integra con altri Servizi AWS per aiutarti a monitorare e risolvere i problemi delle tue funzioni Lambda. Lambda monitora automaticamente le funzioni Lambda per tuo conto e riporta le metriche tramite Amazon. CloudWatch Per monitorare il codice durante la sua esecuzione, Lambda tiene traccia automaticamente del numero di richieste, della durata della chiamata di ogni richiesta e del numero di richieste che restituiscono un errore.

Puoi utilizzarne altri Servizi AWS per risolvere i problemi delle tue funzioni Lambda. In questa sezione viene descritto come utilizzare questi Servizi AWS per monitorare, tracciare, eseguire il debug e risolvere i problemi relativi alle funzioni e alle applicazioni Lambda. Per dettagli sulla registrazione delle funzioni e sugli errori in ogni runtime, consulta le singole sezioni del runtime. Gli argomenti includono il monitoraggio delle [applicazioni](#), i [segnali delle](#) applicazioni e la [registrazione delle chiamate API Lambda con. CloudTrail](#)

## Sections

- [Prezzi](#)
- [Utilizzo delle CloudWatch metriche con Lambda](#)
- [Utilizzo dei log delle funzioni Lambda](#)
- [Registrazione dei log AWS Lambda Chiamate API utilizzando AWS CloudTrail](#)
- [Visualizza le invocazioni delle funzioni Lambda utilizzando AWS X-Ray](#)
- [Monitora le prestazioni delle funzioni con Amazon CloudWatch Lambda Insights](#)
- [Monitoraggio delle applicazioni Lambda](#)
- [Monitora le prestazioni delle applicazioni con Amazon CloudWatch Application Signals](#)
- [Esecuzione da remoto del debug delle funzioni Lambda con Visual Studio Code](#)

## Prezzi

CloudWatch ha un livello gratuito perpetuo. Oltre la soglia del livello gratuito, vengono CloudWatch addebitati parametri, dashboard, allarmi, registri e approfondimenti. Per ulteriori informazioni, consulta [i prezzi di Amazon. CloudWatch](#)



## Utilizzo delle CloudWatch metriche con Lambda

Quando la AWS Lambda funzione termina l'elaborazione di un evento, Lambda invia automaticamente le metriche relative all'invocazione ad Amazon. CloudWatch Non è necessario concedere autorizzazioni aggiuntive al ruolo di esecuzione per ricevere i parametri delle funzioni e non sono previsti costi aggiuntivi per questi parametri.

Esistono molti tipi di parametri associati alle funzioni Lambda. Tra questi vi sono i parametri di invocazione, parametri delle prestazioni, parametri di simultaneità, parametri di invocazione asincrona e parametri dello strumento di mappatura dell'origine degli eventi. Per ulteriori informazioni, consulta [the section called “Tipi di parametri”](#).

Nella CloudWatch console, puoi [visualizzare queste metriche](#) e utilizzarle per creare grafici e dashboard. È possibile impostare allarmi per rispondere alle modifiche apportate alle percentuali di utilizzo, prestazioni o errore. Lambda invia i dati delle metriche a intervalli di 1 minuto. CloudWatch

Se desideri una visione più immediata della funzione Lambda, è possibile creare [parametri personalizzati ad alta risoluzione](#). Le metriche e gli allarmi personalizzati sono a pagamento. CloudWatch Per ulteriori informazioni, consulta [Amazon CloudWatch Pricing](#).

### Visualizzazione di parametri per le funzioni Lambda

Usa la CloudWatch console per visualizzare le metriche delle tue funzioni Lambda. Nella console, puoi filtrare e ordinare le metriche delle funzioni per nome della funzione, alias, versione o UUID di mappatura della sorgente dell'evento.

Per visualizzare le metriche sulla console CloudWatch

1. Apri la pagina delle [metriche](#) (AWS/Lambdamespace) della console. CloudWatch
2. Nella scheda Sfoglia, in Parametri, scegli una delle seguenti dimensioni:
  - Per nome funzione (FunctionName) – Visualizza i parametri aggregati per tutte le versioni e gli alias di una funzione.
  - Per risorsa (Resource) – Visualizza i parametri per una versione o un alias di una funzione.
  - Per version eseguita (ExecutedVersion) – Visualizza i parametri per una combinazione di alias e versione. Utilizzare la dimensione ExecutedVersion per confrontare le percentuali di errore per due versioni di una funzione che sono entrambe destinazioni di un [alias ponderato](#).
  - Per UUID di mapping di origine di eventi (EventSourceMappingUUID): visualizza i parametri per uno strumento di mappatura dell'origine degli eventi.

- In tutte le funzioni (nessuna): visualizza le metriche aggregate per tutte le funzioni della versione corrente. Regione AWS
3. Scegli un parametro. Il parametro dovrebbe apparire automaticamente nel grafico visivo e nella scheda Parametri nel grafico.

Per impostazione predefinita, i grafici utilizzano la statistica Sum per tutti i parametri. Per scegliere un parametro diverso e personalizzare il grafico, utilizzare le opzioni nella scheda Graphed metrics (Parametri grafico).

#### Note

Il timestamp su un parametro riflette quando la funzione è stata invocata. A seconda della durata della chiamata, possono essere necessari diversi minuti prima dell'emissione del parametro. Se, ad esempio, la funzione ha un timeout di 10 minuti, per trovare i parametri accurati bisogna cercare a ritroso oltre i 10 minuti.

Per ulteriori informazioni CloudWatch, consulta la [Amazon CloudWatch User Guide](#).

## Tipi di parametri per le funzioni Lambda

Questa sezione descrive i tipi di metriche Lambda disponibili nella CloudWatch console.

### Argomenti

- [Parametri di invocazione](#)
- [Metriche di distribuzione](#)
- [Metriche delle prestazioni](#)
- [Parametri di concorrenza](#)
- [Parametri di chiamata asincrona](#)
- [Parametri dello strumento di mappatura dell'origine degli eventi](#)

### Parametri di invocazione

I parametri di invocazione sono indicatori binari del risultato di una chiamata alla funzione Lambda. Visualizza questi parametri con la statistica Sum. Ad esempio, se la funzione restituisce un errore, Lambda invia il parametro Errors con un valore pari a 1. Per ottenere un conteggio del numero di

errori di funzione che si sono verificati ogni minuto, visualizzare la somma Sum del parametro `Errors` con un periodo di un minuto.

- **Invocations:** il numero di volte in cui viene chiamato il codice di funzione, incluse le chiamate riuscite e le chiamate che determinano un errore di funzione. Le chiamate non vengono registrate se la richiesta di chiamata è limitata o altrimenti viene generato un errore di chiamata. Il valore di `Invocations` equivale al numero di richieste fatturate.
- **Errors:** il numero di chiamate che provocano un errore di funzione. Gli errori di funzione includono eccezioni generate dal codice e eccezioni generate dal runtime Lambda. Il runtime restituisce errori per problemi quali timeout ed errori di configurazione. Per calcolare la percentuale di errore, dividere il valore di `Errors` per il valore di `Invocations`. Tieni presente che il timestamp di un parametro di errore riflette quando è stata richiamata la funzione, non quando si è verificato l'errore.
- **DeadLetterErrors**— Per l'invocazione [asincrona](#), il numero di volte in cui Lambda tenta di inviare un evento a una coda di lettere morte (DLQ) ma non riesce. Dead-letter possono verificarsi errori dovuti a risorse o limiti di dimensione impostati in modo errato.
- **DestinationDeliveryFailures:** per la chiamata asincrona e per lo [strumento di mappatura dell'origine degli eventi](#) supportato, indica il numero di tentativi di invio non riusciti da parte di Lambda di un evento a una [destinazione](#). Per gli strumenti di mappatura dell'origine degli eventi, Lambda supporta destinazioni per le origini di flusso (DynamoDB e Kinesis). Gli errori di recapito possono verificarsi a causa di errori di autorizzazioni, risorse configurate erroneamente o limiti di dimensione. Gli errori possono verificarsi anche se la destinazione che hai configurato è di tipo non supportato, ad esempio una coda FIFO di Amazon SQS o un argomento FIFO di Amazon SNS.
- **Throttles:** il numero di richieste di chiamata con throttling. Quando tutte le istanze di funzione elaborano le richieste e non è disponibile alcuna simultaneità per l'aumento, Lambda rifiuta le richieste aggiuntive con un errore `TooManyRequestsException`. Le richieste con limitazione e altri errori di chiamata non contano come `Invocations` o `Errors`.

#### Note

Con [Lambda Managed Instances](#), Lambda fornisce metriche granulari sull'accelerazione che identificano il vincolo specifico che causa l'accelerazione. Quando si verifica un'accelerazione nell'ambiente di esecuzione, viene emessa esattamente una delle seguenti metriche secondarie con un valore pari a 1, mentre le restanti tre vengono emesse con un valore pari a 0. La `Throttles` metrica viene sempre emessa insieme a queste metriche secondarie.

- **CPUThrottles**— Chiamate limitate a causa dell'esaurimento della CPU nell'ambiente di esecuzione.
  - **MemoryThrottles**— Chiamate limitate a causa dell'esaurimento della memoria nell'ambiente di esecuzione.
  - **DiskThrottles**— Chiamate limitate a causa dell'esaurimento del disco nell'ambiente di esecuzione.
  - **ConcurrencyThrottles**— Le chiamate vengono limitate quando viene raggiunto il limite di concorrenza dell'ambiente di esecuzione.
- **OversizedRecordCount**: per le origini di eventi di Amazon DocumentDB, il numero di eventi che la funzione riceve dal flusso di modifiche è superiore a 6 MB. Lambda elimina il messaggio ed emette questo parametro.
  - **ProvisionedConcurrencyInvocations**: il numero di volte in cui il codice di funzione viene richiamato tramite la [simultaneità fornita](#).
  - **ProvisionedConcurrencySpilloverInvocations**: il numero di volte in cui il codice di funzione viene chiamato tramite la simultaneità standard quando è in uso tutta la simultaneità fornita.
  - **RecursiveInvocationsDropped**: il numero di volte in cui Lambda ha interrotto l'invocazione della funzione perché ha rilevato che la funzione fa parte di un ciclo ricorsivo infinito. Il rilevamento del ciclo ricorsivo monitora quante volte una funzione viene richiamata come parte di una catena di richieste tracciando i metadati aggiunti dagli SDK supportati. AWS Per impostazione predefinita, se la funzione viene richiamata come parte di una catena di richieste circa 16 volte, Lambda interrompe l'invocazione successiva. Se disabiliti il rilevamento del ciclo ricorsivo, questo parametro non viene emesso. Per ulteriori informazioni sull'utilizzo di questa caratteristica, consulta [Usa il rilevamento di un ciclo ricorsivo Lambda per prevenire loop infiniti](#).

## Metriche di distribuzione

Le metriche di distribuzione forniscono informazioni sugli eventi di distribuzione delle funzioni Lambda e sui relativi processi di convalida.

- **SignatureValidationErrors**— Il numero di volte in cui si è verificata la distribuzione di un pacchetto di codice con errori di convalida della firma quando la politica di configurazione della firma del codice è impostata su. Warn Questa metrica viene emessa quando i controlli di scadenza, mancata corrispondenza o revoca hanno esito negativo, ma la distribuzione è comunque consentita a causa dell'impostazione della politica. Warn Per ulteriori informazioni sulla

firma del codice, vedere. [Utilizzo della firma del codice per verificare l'integrità del codice con Lambda](#)

## Metriche delle prestazioni

I parametri delle prestazioni forniscono dettagli delle prestazioni relativi a una singola chiamata della funzione. Ad esempio, il parametro `Duration` indica il tempo in millisecondi che la funzione impiega per l'elaborazione di un evento. Per avere un'idea della velocità con cui la funzione elabora gli eventi, visualizzare questi parametri con la statistica `Average` o `Max`.

- `Duration` – La quantità di tempo che il codice della funzione impiega durante l'elaborazione di un evento. La durata fatturata per una invocazione è il valore di `Duration` arrotondato per eccesso al millisecondo più vicino. `Duration` non include il tempo di avvio a freddo.
- `PostRuntimeExtensionsDuration` – La quantità cumulativa di tempo che il runtime trascorre eseguendo il codice per le estensioni dopo il completamento del codice funzione.
- `IteratorAge`: per le origini eventi DynamoDB, Kinesis e Amazon DocumentDB, l'età (in millisecondi) dell'ultimo record dell'evento. Questo parametro misura il tempo che passa tra il momento in cui il flusso riceve il record e il momento in cui lo strumento di mappatura dell'origine degli eventi invia l'evento alla funzione.
- `OffsetLag`: per le origini eventi Apache Kafka autogestite e Streaming gestito da Amazon per Apache Kafka (Amazon MSK), la differenza di offset tra l'ultimo record scritto su un argomento e l'ultimo record elaborato dal gruppo di consumer della funzione. Sebbene un argomento di Kafka possa avere più partizioni, questo parametro misura il ritardo di offset a livello di argomento.

`Duration` supporta anche le statistiche percentili (p). Utilizzare i percentili per escludere valori estremi che incideranno sulle statistiche `Average` e `Maximum`. Ad esempio, la statistica `p95` mostra la durata massima del 95% delle chiamate, escludendo il 5% più lento. Per ulteriori informazioni, consulta [Percentiles](#) nella Amazon CloudWatch User Guide.

## Parametri di concorrenza

Lambda segnala i parametri di simultaneità come conteggio aggregato del numero di istanze che elaborano eventi in una funzione, una versione, un alias o una Regione AWS. Per vedere quanto sei vicino al superamento dei [limiti di simultaneità](#), visualizza questi parametri con la statistica `Max`.

- `ConcurrentExecutions` – Il numero di istanze di funzione che stanno elaborando gli eventi. Se questo numero raggiunge la [quota di esecuzioni simultanee](#) per la regione o il limite di [simultaneità riservato](#) configurato per la funzione, Lambda limita le richieste di chiamata aggiuntive.
- `ProvisionedConcurrentExecutions`: il numero di istanze di funzione che stanno elaborando eventi tramite la [simultaneità fornita](#). Per ogni chiamata di un alias o versione con la simultaneità fornita, Lambda emette il conteggio corrente. Se la tua funzione è inattiva o non riceve richieste, Lambda non emette questa metrica.
- `ProvisionedConcurrencyUtilization`: per una versione o un alias, il valore di `ProvisionedConcurrentExecutions` diviso per la quantità totale di simultaneità fornita configurata. Ad esempio, se configuri una simultaneità fornita pari a 10 per la funzione e `ProvisionedConcurrentExecutions` è 7, allora `ProvisionedConcurrencyUtilization` è 0,7.

Se la tua funzione è inattiva o non riceve richieste, Lambda non emette questa metrica perché è basata su `ProvisionedConcurrentExecutions`. Tienilo a mente se la usi `ProvisionedConcurrencyUtilization` come base per gli allarmi. CloudWatch

- `UnreservedConcurrentExecutions`: per una regione, il numero di eventi che vengono elaborati da funzioni che non dispongono di simultaneità riservata.
- `ClaimedAccountConcurrency`: per una Regione, la quantità di simultaneità non disponibile per le invocazioni on demand. `ClaimedAccountConcurrency` corrisponde a `UnreservedConcurrentExecutions` più la quantità di simultaneità allocata (ovvero la simultaneità totale riservata più la simultaneità totale fornita). Per ulteriori informazioni, consulta [Lavorare con il parametro ClaimedAccountConcurrency](#).

## Parametri di chiamata asincrona

I parametri di chiamata asincrona forniscono dettagli sulle chiamate asincrone da origini di eventi e sulle chiamate dirette. Puoi impostare le soglie e gli allarmi per la notifica di alcuni cambiamenti. Ad esempio, quando si verifica un aumento indesiderato del numero di eventi in coda per l'elaborazione (`AsyncEventsReceived`). Oppure, quando un evento aspetta da molto tempo di essere elaborato (`AsyncEventAge`).

- `AsyncEventsReceived`: il numero di eventi che Lambda mette correttamente in coda per l'elaborazione. Questo parametro fornisce informazioni sul numero di eventi ricevuti da una funzione Lambda. Monitora questo parametro e imposta gli allarmi relativi alle soglie per verificare eventuali problemi. Ad esempio, per rilevare un numero indesiderato di eventi inviati a Lambda

e diagnosticare rapidamente i problemi derivanti da configurazioni errate di trigger o funzioni. Le discrepanze tra `AsyncEventsReceived` e `Invocations` possono indicare una disparità nell'elaborazione, l'eliminazione degli eventi o un potenziale arretrato della coda.

- `AsyncEventAge`: il tempo che intercorre tra il momento in cui Lambda mette in coda correttamente l'evento e il momento in cui la funzione viene richiamata. Il valore di questo parametro aumenta quando gli eventi vengono ritentati a causa di errori di chiamata o limitazioni. Monitora questo parametro e imposta allarmi per rilevare le soglie su diverse statistiche relative a quando si verifica un accumulo di code. Per risolvere un aumento di questo parametro, consulta il parametro `Errors` per identificare gli errori della funzione e il parametro `Throttles` per identificare i problemi di simultaneità.
- `AsyncEventsDropped`: il numero di eventi eliminati senza eseguire correttamente la funzione. Se configuri una coda DLQ o una destinazione `OnFailure`, gli eventi vengono inviati lì prima di essere eliminati. Gli eventi vengono eliminati per diversi motivi. Ad esempio, possono superare la durata massima o esaurire il numero massimo di tentativi oppure la simultaneità riservata potrebbe essere impostata su 0. Per risolvere il problema relativo all'eliminazione degli eventi, consulta il parametro `Errors` per identificare gli errori della funzione e il parametro `Throttles` per identificare i problemi di simultaneità.

## Parametri dello strumento di mappatura dell'origine degli eventi

I parametri dello strumento di mappatura dell'origine degli eventi forniscono informazioni sul comportamento di elaborazione dello strumento di mappatura dell'origine degli eventi.

Attualmente, le metriche di mappatura delle sorgenti di eventi sono disponibili per Amazon SQS, Kinesis, DynamoDB, Amazon MSK e le fonti di eventi Apache Kafka autogestite.

Per la mappatura delle sorgenti degli eventi con la configurazione delle metriche, puoi anche controllare tutte le metriche relative a ESM nella scheda Monitor dalla pagina Console Lambda > Risorse aggiuntive > mappature delle sorgenti degli eventi ora.

Per abilitare i parametri o uno strumento di mappatura dell'origine degli eventi (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione per la quale desideri abilitare i parametri.
3. Scegli la scheda Configurazione, quindi scegli Trigger.
4. Scegli lo strumento di mappatura dell'origine degli eventi per cui desideri abilitare i parametri, quindi scegli Modifica.

5. In Configurazione della mappatura delle sorgenti degli eventi, scegli Abilita metriche o selezionale dall'elenco a discesa Metriche.
6. Scegli Save (Salva).

In alternativa, puoi abilitare le metriche per la mappatura della sorgente dell'evento a livello di codice utilizzando l'oggetto nel tuo. [EventSourceMappingMetricsConfig](#) [EventSourceMappingConfiguration](#) Ad esempio, il seguente comando [UpdateEventSourceMapping](#) CLI abilita le metriche per la mappatura della sorgente di un evento:

```
aws lambda update-event-source-mapping \
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
  --metrics-config Metrics=EventCount
```

Esistono 3 gruppi di metriche: 1EventCount, 1 e 3 ErrorCount e KafkaMetrics ogni gruppo ha più metriche. Non tutte le metriche sono disponibili per ogni origine di evento. La tabella seguente riassume le metriche supportate per ogni tipo di origine di eventi.

È necessario attivare il gruppo di metriche per ricevere le metriche relative alle metriche. ad esempio, impostato EventCount nella configurazione delle metriche per avere: (PolledEventCount,,,, e). FilteredOutEventCount InvokedEventCount FailedInvokeEventCount DroppedEventCount OnFailureDestinationDeliveredEventCount DeletedEventCount

| Parametro dello strumento di mappatura dell'origine degli eventi | Gruppo di metriche | Amazon SQS | Stream Kinesis e DynamoDB | Amazon MSK e Apache Kafka autogestito |
|------------------------------------------------------------------|--------------------|------------|---------------------------|---------------------------------------|
| PolledEventCount                                                 | EventCount         | Sì         | Sì                        | Sì                                    |
| FilteredOutEventCount                                            | EventCount         | Sì         | Sì                        | Sì                                    |
| InvokedEventCount                                                | EventCount         | Sì         | Sì                        | Sì                                    |



| Parametro dello strumento di mappatura dell'origine degli eventi | Gruppo di metriche | Amazon SQS | Stream Kinesis e DynamoDB | Amazon MSK e Apache Kafka autogestito |
|------------------------------------------------------------------|--------------------|------------|---------------------------|---------------------------------------|
| FailedInvokeEventCount                                           | EventCount         | Sì         | Sì                        | Sì                                    |
| DroppedEventCount                                                | EventCount         | No         | Sì                        | Sì                                    |
| OnFailureDestinationDeliveredEventCount                          | EventCount         | No         | Sì                        | Sì                                    |
| DeletedEventCount                                                | EventCount         | Sì         | No                        | No                                    |
| CommittedEventCount                                              | EventCount         | No         | No                        | Sì                                    |
| PollingErrorCount                                                | ErrorCount         | No         | No                        | Sì                                    |
| InvokeErrorCount                                                 | ErrorCount         | No         | No                        | Sì                                    |
| OnFailureDestinationDeliveryErrorCount                           | ErrorCount         | No         | No                        | Sì                                    |

| Parametro dello strumento di mappatura dell'origine degli eventi | Gruppo di metriche | Amazon SQS | Stream Kinesis e DynamoDB | Amazon MSK e Apache Kafka autogestito |
|------------------------------------------------------------------|--------------------|------------|---------------------------|---------------------------------------|
| SchemaRegistryErrorCount                                         | ErrorCount         | No         | No                        | Sì                                    |
| CommitErrorCount                                                 | ErrorCount         | No         | No                        | Sì                                    |
| MaxOffsetLag                                                     | KafkaMetrics       | No         | No                        | Sì                                    |
| SumOffsetLag                                                     | KafkaMetrics       | No         | No                        | Sì                                    |

Inoltre, se lo strumento di mappatura dell'origine degli eventi delle origini eventi è in [modalità provisioning](#), Lambda fornisce il seguente parametro:

- **ProvisionedPollers**: per lo strumento di mappatura dell'origine degli eventi in modalità provisioning, il numero di poller di eventi che sono attivamente in esecuzione. Visualizza questa metrica usando la matematica. MAX
- (Solo Amazon MSK e sorgenti di eventi Apache Kafka autogestite)**EventPollerUnit**: per la mappatura delle sorgenti di eventi in modalità provisioning, indica il numero di unità di poller di eventi in esecuzione attiva. Visualizza SUM questa metrica usando la matematica.
- (Amazon MSK e sorgenti di eventi Apache Kafka autogestite)**EventPollerThroughputInBytes**: per la mappatura delle sorgenti di eventi in modalità provisioned, la dimensione totale dei record dei poller di eventi interrogati dalla fonte dell'evento. Può dirti l'attuale throughput di polling. Visualizza questa metrica usando la matematica. SUM

Ecco maggiori dettagli su ciascuna metrica:

- **PolledEventCount**: il numero di eventi che Lambda legge correttamente dall'origine eventi. Se Lambda esegue un polling di eventi ma riceve un polling vuoto (nessun nuovo record), Lambda emette un valore 0 per questo parametro. Usa questo parametro per rilevare se lo strumento di mappatura dell'origine degli eventi esegue correttamente il polling dei nuovi eventi.
- **FilteredOutEventCount**: per lo strumento di mappatura dell'origine degli eventi con un [criterio di filtro](#), il numero di eventi filtrati in base a tali criteri di filtro. Utilizza questo parametro per rilevare se lo strumento di mappatura dell'origine degli eventi filtra correttamente gli eventi. Per gli eventi che soddisfano i criteri di filtro, Lambda emette un parametro 0.
- **InvokedEventCount**: il numero di eventi che hanno richiamato la funzione Lambda. Usa questo parametro per verificare che gli eventi stiano richiamando correttamente la tua funzione. Se un evento genera un errore di funzione o una limitazione, **InvokedEventCount** potrebbe essere conteggiato più volte per lo stesso evento sottoposto a polling a causa dei tentativi automatici.

#### Warning

Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per questo motivo, gli eventi potrebbero essere contati più volte nelle metriche che coinvolgono il conteggio degli eventi.

- **FailedInvokeEventCount**: il numero di eventi con cui Lambda ha provato a richiamare la funzione ma senza riuscirci. Le invocazioni possono avere esito negativo per motivi quali problemi di configurazione della rete, autorizzazioni non corrette o una funzione, una versione o un alias Lambda eliminati. Se lo strumento di mappatura dell'origine degli eventi ha abilitato le [risposte in batch parziali](#), **FailedInvokeEventCount** include qualsiasi evento con un valore non vuoto **BatchItemFailures** nella risposta.

#### Note

Il timestamp per il parametro **FailedInvokeEventCount** rappresenta la fine dell'invocazione della funzione. Questo comportamento è diverso dagli altri parametri di errore di invocazione Lambda, che hanno un timestamp all'inizio dell'invocazione della funzione.

- **DroppedEventCount**: il numero di eventi che Lambda ha interrotto a causa della scadenza o dell'esaurimento dei nuovi tentativi. In particolare, si tratta del numero di record che superano i valori configurati per **MaximumRecordAgeInSeconds** o **MaximumRetryAttempts**. È importante sottolineare che questo non include il numero di record che scadono a causa del superamento

delle impostazioni di conservazione dell'origine eventi. Gli eventi eliminati escludono anche gli eventi inviati a una [destinazione in errore](#). Utilizza questo parametro per rilevare un aumento del backlog di eventi.

- **OnFailureDestinationDeliveredEventCount**: per lo strumento di mappatura dell'origine degli eventi con una [destinazione in errore](#) configurata, il numero di eventi inviati a tale destinazione. Utilizza questo parametro per monitorare gli errori di funzione relativi alle chiamate da questa origine eventi. Se la consegna alla destinazione non riesce, Lambda gestisce i parametri come segue:
  - Lambda non emette il parametro **OnFailureDestinationDeliveredEventCount**.
  - Per il parametro **DestinationDeliveryFailures**, Lambda emette un 1.
  - Per il parametro **DroppedEventCount**, Lambda emette un numero pari al numero di eventi che hanno avuto esito negativo nella consegna.
- **DeletedEventCount**: il numero di eventi che Lambda elimina correttamente in seguito all'elaborazione. Se Lambda prova a eliminare un evento ma fallisce, Lambda emette un parametro 0. Utilizza questo parametro per assicurarti che gli eventi elaborati correttamente vengano eliminati dall'origine eventi.
- **CommittedEventCount**— Il numero di eventi che Lambda ha eseguito correttamente dopo l'elaborazione. È una somma dei delta dell'ultimo e dell'attuale offset commesso da ciascuna partizione nella mappatura della sorgente degli eventi di Kafka.
- **PollingErrorCount**— Il numero di errori dovuti al mancato rilevamento delle richieste da parte di Lambda dall'origine dell'evento. Lambda emette questi dati metrici solo quando si è verificato un errore.
- **InvokeErrorCount**— Il numero di errori dovuti al mancato richiamo della funzione da parte di Lambda. Nota che la chiamata è costituita da record in batch. Il numero è a livello di batch, non a livello di numero di record. Lambda emette questi dati metrici solo quando si verifica un errore.
- **SchemaRegistryErrorCount**— Il numero di errori che Lambda non è riuscito a recuperare lo schema o a deserializzare con lo schema. Lambda emette questi dati metrici solo quando si verifica un errore.
- **CommitErrorCount**— Il numero di errori che Lambda non è riuscito a commettere nel cluster Kafka. Lambda emette questi dati metrici solo quando si è verificato un errore.
- **MaxOffsetLag**— Il numero massimo di ritardi di offset (differenza tra gli offset più recenti e quelli confermati) su tutte le partizioni nella mappatura della sorgente dell'evento.
- **SumOffsetLag**— La somma dei ritardi di offset su tutte le partizioni nella mappatura della sorgente dell'evento.

Se lo strumento di mappatura dell'origine degli eventi è disabilitato, non riceverai i parametri dello strumento. Potresti inoltre visualizzare delle metriche mancanti se la disponibilità di Lambda è CloudWatch ridotta.

## Utilizzo dei log delle funzioni Lambda

Per aiutarti a risolvere i guasti, monitora AWS Lambda automaticamente le funzioni Lambda per tuo conto. Puoi visualizzare i log delle funzioni Lambda utilizzando la console Lambda, la console, la CloudWatch (), l' AWS Command Line Interface API. AWS CLI CloudWatch Puoi anche configurare Lambda per inviare i log ad Amazon S3 e Firehose.

Se il ruolo di [esecuzione della tua funzione](#) dispone delle autorizzazioni necessarie, Lambda acquisisce i log di tutte le richieste gestite dalla tua funzione e li invia ad Amazon CloudWatch Logs, che è la destinazione predefinita. Puoi anche utilizzare la console Lambda per configurare Amazon S3 o Firehose come destinazioni di registrazione.

- CloudWatch Logs è la destinazione di registrazione predefinita per le funzioni Lambda. CloudWatch Logs fornisce funzionalità di visualizzazione e analisi dei log in tempo reale, con supporto per la creazione di metriche e allarmi in base ai dati di registro.
- Amazon S3 è economico per lo storage a lungo termine e servizi come Athena possono essere utilizzati per analizzare i log. La latenza è in genere superiore.
- Firehose offre lo streaming gestito dei log verso varie destinazioni. Se devi inviare i log ad altri AWS servizi (ad esempio, OpenSearch Service o Redshift Data API) o piattaforme di terze parti (come Datadog, New Relic o Splunk), Firehose semplifica tale processo fornendo integrazioni predefinite. Puoi anche eseguire lo streaming su endpoint HTTP personalizzati senza configurare un'infrastruttura aggiuntiva.

### Scelta della destinazione del servizio a cui inviare i log

Quando scegli un servizio come destinazione per i log delle funzioni, considera i seguenti fattori chiave:

- La gestione dei costi varia in base al servizio. Amazon S3 offre in genere l'opzione più economica per lo storage a lungo termine, mentre CloudWatch Logs offre la possibilità di visualizzare i log, elaborare i log e impostare avvisi in tempo reale. I costi di Firehose includono sia il servizio di streaming che i costi associati alla configurazione su cui lo si configura per lo streaming.
- Le funzionalità di analisi variano a seconda dei servizi. CloudWatch Logs eccelle nel monitoraggio in tempo reale e si integra in modo nativo con altre CloudWatch funzionalità, come Logs Insights e Live Tail. Amazon S3 funziona bene con strumenti di analisi come Athena e può integrarsi con vari servizi, anche se potrebbe richiedere una configurazione aggiuntiva. Firehose semplifica lo streaming diretto verso AWS servizi specifici (come OpenSearch Service e Redshift Data API) e

piattaforme di terze parti supportate (come Datadog e Splunk) fornendo integrazioni predefinite, che potenzialmente riducono il lavoro di configurazione.

- La configurazione e la facilità d'uso variano in base al servizio. CloudWatch Logs è la destinazione predefinita dei log: funziona immediatamente senza configurazioni aggiuntive e fornisce una visualizzazione e un'analisi semplici dei log tramite la CloudWatch console. Se hai bisogno di inviare i log ad Amazon S3, dovrai eseguire alcune impostazioni iniziali nella console Lambda e configurare le autorizzazioni del bucket. Se hai bisogno di log inviati direttamente a servizi come OpenSearch Service o piattaforme di analisi di terze parti, Firehose può semplificare tale processo.

## Configurazione delle destinazioni dei log

AWS Lambda supporta più destinazioni per i registri delle funzioni. Questa guida spiega le destinazioni di registrazione disponibili e ti aiuta a scegliere l'opzione giusta per le tue esigenze. Indipendentemente dalla destinazione scelta, Lambda offre opzioni per controllare il formato, il filtraggio e la consegna dei log.

Lambda supporta sia il formato JSON che il formato di testo normale per i log della funzione. I log strutturati JSON offrono una migliore ricercabilità e consentono un'analisi automatica, mentre i log in testo normale offrono semplicità e costi di archiviazione potenzialmente ridotti. Puoi controllare quali log Lambda invia alla destinazione prescelta configurando i livelli di log sia per i log di sistema che per quelli delle applicazioni. Il filtraggio aiuta a gestire i costi di archiviazione e semplifica la ricerca delle voci di registro pertinenti durante il debug.

Per istruzioni dettagliate sulla configurazione per ciascuna destinazione, consulta le seguenti sezioni:

- [Invio dei log delle funzioni Lambda ai CloudWatch log](#)
- [Invio dei log delle funzioni Lambda a Firehose](#)
- [Invio di log delle funzioni Lambda ad Amazon S3](#)

## Configurazione dei controlli di registrazione avanzati per le funzioni Lambda

Per darti un maggiore controllo su come i log delle funzioni vengono acquisiti, elaborati e consumati, Lambda offre le seguenti opzioni di configurazione dei log:

- Formato di registro: seleziona tra testo normale e formato JSON strutturato per i log della tua funzione.

- Livello di registro: per i log strutturati JSON, scegli il livello di dettaglio dei log a cui Lambda invia CloudWatch, ad esempio FATAL, ERROR, WARN, INFO, DEBUG e TRACE.
- Gruppo di log: scegli il gruppo di CloudWatch log a cui la tua funzione invia i log.

Per ulteriori informazioni sulla configurazione dei controlli di registrazione avanzati, consulta le seguenti sezioni:

- [Configurazione dei formati di log JSON e testo normale](#)
- [Log-level filtraggio](#)
- [Configurazione dei gruppi di CloudWatch log](#)

## Configurazione dei formati di log JSON e testo normale

L'acquisizione degli output log come coppie chiave-valore JSON semplifica la ricerca e il filtraggio durante il debug delle funzioni. Con i log in formato JSON, puoi aggiungere ai log anche tag e informazioni contestuali. Questo può aiutarti a eseguire analisi automatizzate di grandi volumi di dati di log. A meno che il flusso di lavoro di sviluppo non si basi su strumenti esistenti che utilizzano i log Lambda in testo normale, ti consigliamo di selezionare JSON come formato di log.

### Istanze Lambda gestite

Le istanze gestite Lambda supportano solo il formato di log JSON. Quando si crea una funzione Managed Instances, Lambda configura automaticamente il formato di log in JSON e non è possibile modificarlo in testo normale. Per ulteriori informazioni sulle istanze gestite, consulta [the section called "Istanze gestite"](#)

Per tutti i runtime gestiti da Lambda, puoi scegliere se inviare i log di sistema della tua funzione a CloudWatch Logs in testo semplice non strutturato o in formato JSON. I log di sistema sono i log generati da Lambda e a volte vengono chiamati log eventi della piattaforma.

Per i [runtime supportati](#), quando si utilizza uno dei metodi di registrazione integrati supportati, Lambda può anche generare i log delle applicazioni della funzione (i log generati dal codice della funzione) in formato JSON strutturato. Quando configuri il formato di log della funzione per questi runtime, la configurazione scelta si applica sia ai log di sistema sia a quelli delle applicazioni.



Per i runtime supportati, se la funzione utilizza una libreria o un metodo di registrazione supportato, non è necessario apportare modifiche al codice esistente per Lambda per acquisire i log in formato JSON strutturato.

#### Note

L'utilizzo della formattazione dei log JSON aggiunge metadati aggiuntivi e codifica i messaggi di log come oggetti JSON contenenti una serie di coppie chiave-valore. Per questo motivo, la dimensione dei messaggi di log della funzione può aumentare.

## Runtime e metodi di registrazione supportati

Lambda attualmente supporta l'opzione di generare i log delle applicazioni in formato JSON strutturato per i seguenti runtime.

| Lingua  | Versioni supportate                                   |
|---------|-------------------------------------------------------|
| Java    | Tutti i runtime Java eccetto Java 8 su Amazon Linux 1 |
| .NET    | .NET 8 e versioni successive                          |
| Node.js | Node.js 16 e versioni successive                      |
| Python  | Python 3.8 e versioni successive                      |
| Ruby    | Ruby 4.0 e versioni successive                        |
| Rust    | n/a                                                   |

Affinché Lambda possa inviare i log delle applicazioni della funzione CloudWatch in formato JSON strutturato, la funzione deve utilizzare i seguenti strumenti di registrazione integrati per generare i log:

- Java: il `LambdaLogger` logger o Log4j2. Per ulteriori informazioni, consulta [the section called “Registrazione dei log”](#).
- .NET: l'`ILambdaLogger` istanza sull'oggetto di contesto. Per ulteriori informazioni, consulta [Registrazione e monitorare le funzioni Lambda con C#](#).

- Node.js- I metodi della console `console.trace`, `console.debug`, `console.log`, `console.info`, `console.error`, e `console.warn`. Per ulteriori informazioni, consulta [the section called “Registrazione dei log”](#).
- Python: la libreria Python logging standard. Per ulteriori informazioni, consulta [the section called “Registrazione dei log”](#).
- Ruby: la libreria Ruby standard. Logger Per ulteriori informazioni, consulta [the section called “Registrazione dei log”](#).
- Rust: La cassa. `tracing` Per ulteriori informazioni, consulta [the section called “Registrazione dei log”](#).

Per altri runtime Lambda gestiti, Lambda attualmente supporta nativamente solo l'acquisizione dei log di sistema in formato JSON strutturato. Tuttavia, è comunque possibile acquisire i log delle applicazioni in formato JSON strutturato in qualsiasi runtime utilizzando strumenti di registrazione come Powertools per AWS Lambda gli output di log in formato JSON in uscita.

### Formati di log predefiniti

Attualmente, il formato di log predefinito per tutti i runtime Lambda è il testo normale. Per le istanze gestite da Lambda, il formato del registro è sempre JSON e non può essere modificato.

Se stai già utilizzando librerie di registrazione come Powertools AWS Lambda per generare i log delle funzioni in formato strutturato JSON, non è necessario modificare il codice se selezioni la formattazione dei log JSON. Lambda non codifica due volte i log già codificati in JSON, quindi i log delle applicazioni della tua funzione continuano a essere acquisiti come prima.

### Formato JSON per i log di sistema

Quando configuri il formato di log della funzione come JSON, ogni elemento del log di sistema (evento della piattaforma) viene acquisito come un oggetto JSON contenente coppie chiave-valore con le seguenti chiavi:

- `"time"`: l'ora in cui è stato generato il messaggio di log
- `"type"`: il tipo di evento che viene registrato
- `"record"`: il contenuto dell'output log

Il formato del valore `"record"` varia in base al tipo di evento registrato. Per ulteriori informazioni, consulta [the section called “Tipi di oggetti Event dell'API di telemetria ”](#). Per ulteriori informazioni sui

livelli di log assegnati ai log eventi di sistema, consulta la pagina [the section called “Strumento di mappatura degli eventi a livello di log di sistema”](#).

A titolo di confronto, i due esempi seguenti mostrano lo stesso output log in formato di testo normale e in formato JSON strutturato. Tieni presente che, nella maggior parte dei casi, i log eventi di sistema contengono più informazioni quando vengono emessi in formato JSON rispetto a quando vengono emessi in testo normale.

Example testo normale:

```
2024-03-13 18:56:24.046000 fbe8c1 INIT_START Runtime Version:
python:3.12.v18 Runtime Version ARN: arn:aws:lambda:eu-
west-1::runtime:edb5a058bfa782cb9cedc6d534ac8b8c193bc28e9a9879d9f5ebaaf619cd0fc0
```

Example JSON strutturato:

```
{
  "time": "2024-03-13T18:56:24.046Z",
  "type": "platform.initStart",
  "record": {
    "initializationType": "on-demand",
    "phase": "init",
    "runtimeVersion": "python:3.12.v18",
    "runtimeVersionArn": "arn:aws:lambda:eu-
west-1::runtime:edb5a058bfa782cb9cedc6d534ac8b8c193bc28e9a9879d9f5ebaaf619cd0fc0"
  }
}
```

#### Note

L'[the section called “API di telemetria”](#) emette sempre eventi della piattaforma come START e REPORT in formato JSON. La configurazione del formato dei log di sistema a cui Lambda invia CloudWatch non influisce sul comportamento dell'API Lambda Telemetry.

## Formato JSON per i log delle applicazioni

Quando configuri il formato di log della funzione come JSON, gli output di log delle applicazioni scritti utilizzando le librerie e i metodi di registrazione di log supportati vengono acquisiti come oggetti JSON che contengono coppie chiave-valore con le chiavi riportate di seguito.

- "timestamp": l'ora in cui è stato generato il messaggio di log
- "level": il livello di log assegnato al messaggio
- "message": il contenuto del messaggio di log
- "requestId"(Python, .NET e Node.js) o "AWSrequestId" (Java): l'ID univoco della richiesta per l'invocazione della funzione

A seconda del metodo di runtime e di registrazione utilizzato dalla funzione, questo oggetto JSON potrebbe contenere anche coppie di chiavi aggiuntive. Ad esempio, in Node.js, se la funzione utilizza console metodi per registrare oggetti di errore utilizzando più argomenti, l'oggetto JSON contiene coppie di valori chiave aggiuntive con le chiavi `errorMessage` e `errorType` `stackTrace`. Per ulteriori informazioni sui log in formato JSON in diversi runtime Lambda, consulta [the section called “Registrazione dei log”](#), [the section called “Registrazione dei log”](#) e [the section called “Registrazione dei log”](#).

#### Note

La chiave utilizzata da Lambda per il valore del timestamp è diversa per i log di sistema e i log delle applicazioni. Per i log di sistema, Lambda utilizza la chiave "time" per mantenere la coerenza con l'API di telemetria. Per i log delle applicazioni, Lambda segue le convenzioni dei runtime supportati e utilizza "timestamp".

A titolo di confronto, i due esempi seguenti mostrano lo stesso output log in formato di testo normale e in formato JSON strutturato.

Example testo normale:

```
2024-10-27T19:17:45.586Z 79b4f56e-95b1-4643-9700-2807f4e68189 INFO some log message
```

Example JSON strutturato:

```
{
  "timestamp": "2024-10-27T19:17:45.586Z",
  "level": "INFO",
  "message": "some log message",
  "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

## Impostazione del formato di log della funzione

Per configurare il formato di log per la tua funzione, puoi usare la console Lambda o AWS Command Line Interface (AWS CLI). Puoi anche configurare il formato di registro di una funzione utilizzando i comandi API [CreateFunction](#) e [UpdateFunctionConfiguration](#) Lambda, la risorsa AWS Serverless Application Model (AWS SAM) [AWS::Serverless::Function](#) e la risorsa CloudFormation [AWS::Lambda::Function](#).

La modifica del formato di registro della funzione non influisce sui log esistenti archiviati in Logs. CloudWatch Solo i nuovi log utilizzano il formato aggiornato.

Se modifichi il formato di log della funzione in JSON e non imposti il livello di log, Lambda imposta in automatico il livello di log dell'applicazione e il livello di log del sistema della funzione su INFO. Ciò significa che Lambda invia solo output di log di livello INFO o inferiore a CloudWatch Logs. Per ulteriori informazioni sul filtraggio a livello di log di applicazioni e sistemi, consulta [the section called “Log-level filtraggio”](#)

### Note

Per i runtime Python, quando il formato di log della funzione è impostato su testo semplice, l'impostazione predefinita a livello di log è WARN. Ciò significa che Lambda invia solo output di log di livello WARN e inferiore a Logs. CloudWatch La modifica del formato di log della funzione in JSON modifica questo comportamento predefinito. Per ulteriori informazioni sulla registrazione di log in Python, consulta [the section called “Registrazione dei log”](#).

Per Node.js le funzioni che emettono log in formato metrico incorporato (EMF), la modifica del formato di registro della funzione in CloudWatch JSON potrebbe comportare l'impossibilità di riconoscere le metriche.

### Important

Se la tua funzione utilizza Powertools for AWS Lambda (TypeScript) o le librerie client EMF open source per emettere log EMF, aggiorna le librerie [Powertools](#) ed [EMF](#) alle versioni più recenti per assicurarti di poter continuare ad analizzare correttamente i log. CloudWatch Se passi al formato di log JSON, ti consigliamo anche di eseguire dei test per garantire la compatibilità con i parametri incorporati della tua funzione. Per ulteriori consigli sulle funzioni

node.js che emettono log EMF, consulta la pagina [the section called “Utilizzo di librerie client formato del parametro incorporato \(EMF\) con log JSON strutturati”](#).

Per configurare il formato di registro di una funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione
3. Nella pagina di configurazione della funzione, scegli Strumenti di monitoraggio e gestione.
4. Nel riquadro Configurazione della registrazione, scegli Modifica.
5. In Contenuto del log, per Formato del log seleziona Testo o JSON.
6. Scegli Save (Salva).

Per modificare il formato di registro di una funzione esistente (AWS CLI)

- Per modificare il formato di log di una funzione esistente, utilizza il comando [update-function-configuration](#). Imposta l'opzione LogFormat in LoggingConfig su JSON o Text.

```
aws lambda update-function-configuration \  
  --function-name myFunction \  
  --logging-config LogFormat=JSON
```

Per impostare il formato del registro quando si crea una funzione (AWS CLI)

- Per configurare il formato di log quando crei una nuova funzione, utilizza l'opzione `--logging-config` nel comando [create-function](#). Impostare LogFormat su JSON o su Text. Il seguente comando di esempio crea una Node.js funzione che genera i log in formato JSON strutturato.

Se non specifichi un formato di registro quando crei una funzione, Lambda utilizza il formato di registro predefinito per la versione di runtime selezionata. Per informazioni sui formati di registrazione predefiniti, consulta la pagina [the section called “Formati di log predefiniti”](#).

```
aws lambda create-function \  
  --function-name myFunction \  
  --runtime nodejs24.x \  
  --handler index.handler \  
  --zip-file fileb://function.zip \  
  --logging-config LogFormat=JSON
```

```
--role arn:aws:iam::123456789012:role/LambdaRole \  
--logging-config LogFormat=JSON
```

## Log-level filtraggio

Lambda può filtrare i log della funzione in modo che solo i log con un determinato livello di dettaglio o inferiore vengano inviati a CloudWatch Logs. Puoi configurare il filtraggio a livello di log separatamente per i log di sistema della funzione (i log generati da Lambda) e i log delle applicazioni (i log generati dal codice della funzione).

Per [the section called “Runtime e metodi di registrazione supportati”](#), non è necessario apportare modifiche al codice della funzione per Lambda per filtrare i log delle applicazioni della funzione in Lambda.

Per tutti gli altri runtime e metodi di registrazione, il codice della funzione deve generare log eventi in `stdout` o `stderr` come oggetti in formato JSON contenenti una coppia chiave-valore con la chiave `"level"`. Ad esempio, Lambda interpreta il seguente output su `stdout` come un log di livello `DEBUG`.

```
print({'level': "debug", "msg": "my debug log", "timestamp":  
      "2024-11-02T16:51:31.587199Z"})
```

Se il campo `"level"` valore non è valido o manca, Lambda assegna all'output del log il livello `INFO`. Affinché Lambda utilizzi il campo `timestamp`, è necessario specificare l'ora in un formato timestamp [RFC 3339](#) valido. Se non fornisci un timestamp valido, Lambda assegna al log il livello `INFO` e aggiunge un timestamp per te.

Quando assegni un nome alla chiave `timestamp`, segui le convenzioni del runtime che stai utilizzando. Lambda supporta le convenzioni di denominazione più comuni utilizzate dai runtime gestiti.

### Note

Per utilizzare il filtraggio a livello di log, la funzione deve essere configurata per utilizzare il formato di log JSON. Attualmente, il formato di log predefinito per tutti i runtime gestiti da Lambda è il testo normale. Per informazioni su come impostare il formato di log della funzione su JSON, consulta la pagina [the section called “Impostazione del formato di log della funzione”](#).

Per i log delle applicazioni (i log generati dal codice della funzione), puoi scegliere tra i seguenti livelli di log.

| Livello di log            | Utilizzo standard                                                                               |
|---------------------------|-------------------------------------------------------------------------------------------------|
| TRACE (dettaglio massimo) | Le informazioni più dettagliate utilizzate per tracciare il percorso di esecuzione del codice   |
| DEBUG                     | Informazioni dettagliate per il debug del sistema                                               |
| INFO                      | Messaggi che registrano il normale funzionamento della funzione                                 |
| WARN                      | Messaggi su potenziali errori che potrebbero portare a comportamenti imprevisti se non risolti  |
| ERRORE                    | Messaggi relativi a problemi che impediscono al codice di funzionare come previsto              |
| FATAL (dettaglio minimo)  | Messaggi relativi a errori gravi che causano l'interruzione del funzionamento dell'applicazione |

Quando si seleziona un livello di registro, Lambda invia i log a tale livello e inferiore a CloudWatch Logs. Ad esempio, se imposti il livello di log dell'applicazione di una funzione su WARN, Lambda non invia output di log ai livelli INFO e DEBUG. Il livello di log dell'applicazione predefinito per il filtraggio dei log è INFO.

Quando Lambda filtra i log delle applicazioni della funzione, ai messaggi di log privi di livello viene assegnato il livello di log INFO.

Per i log di sistema (i log generati dal servizio Lambda), puoi scegliere tra i seguenti livelli di log.

| Livello di log            | Utilizzo                                          |
|---------------------------|---------------------------------------------------|
| DEBUG (dettaglio massimo) | Informazioni dettagliate per il debug del sistema |



| Livello di log          | Utilizzo                                                                                       |
|-------------------------|------------------------------------------------------------------------------------------------|
| INFO                    | Messaggi che registrano il normale funzionamento della funzione                                |
| WARN (dettaglio minimo) | Messaggi su potenziali errori che potrebbero portare a comportamenti imprevisti se non risolti |

Quando si seleziona un livello di log, Lambda invia i log di quel livello e di livello inferiore. Ad esempio, se imposti il livello di log di sistema di una funzione su INFO, Lambda non invia output di log a livello DEBUG.

Per impostazione predefinita, Lambda imposta il livello di log del sistema su INFO. Con questa impostazione, Lambda invia "start" e "report" registra automaticamente i messaggi a. CloudWatch Per ricevere log di sistema più o meno dettagliati, modifica il livello di log in DEBUG o WARN. Per visualizzare un elenco dei livelli di log a cui Lambda mappa i diversi log eventi di sistema, consulta la pagina [the section called “Strumento di mappatura degli eventi a livello di log di sistema”](#).

### Configurazione del filtraggio a livello di log

Per configurare il filtro a livello di log dell'applicazione e del sistema per la tua funzione, puoi utilizzare la console Lambda o (). AWS Command Line Interface AWS CLI Puoi anche configurare il livello di log di una funzione utilizzando i comandi API [CreateFunction](#) e [UpdateFunctionConfiguration](#) Lambda, la risorsa AWS Serverless Application Model (AWS SAM) [AWS::Serverless::Function](#) e la risorsa: :Function. CloudFormation [AWS::Lambda](#)

Tieni presente che se imposti il livello di log della funzione nel codice, questa impostazione ha la precedenza su qualsiasi altra impostazione del livello di log che configuri. Ad esempio, se utilizzi il metodo `logging.setLevel()` di Python per impostare il livello di registrazione di log della funzione su INFO, questa impostazione ha la precedenza su un'impostazione di WARN configurata utilizzando la console Lambda.

Per configurare il livello di registro dell'applicazione o del sistema (console) di una funzione esistente

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Nella pagina di configurazione della funzione, scegli Strumenti di monitoraggio e gestione.

4. Nel riquadro Configurazione della registrazione, scegli Modifica.
5. In Contenuto del log, per Formato del log assicurati che sia selezionato JSON.
6. Utilizzando i pulsanti di opzione, seleziona i valori di Livello di log dell'applicazione e Livello di log del sistema desiderati per la funzione.
7. Scegli Save (Salva).

Per configurare l'applicazione o il livello di registro di sistema di una funzione esistente (AWS CLI)

- Per modificare il livello di log dell'applicazione o del sistema di una funzione esistente, utilizza il comando [update-function-configuration](#). Utilizza `--logging-config` per impostare `SystemLogLevel` su `DEBUG`, `INFO` o `WARN`. Imposta `ApplicationLogLevel` su `DEBUG`, `INFO`, `WARN`, `ERROR` o `FATAL`.

```
aws lambda update-function-configuration \  
  --function-name myFunction \  
  --logging-config LogFormat=JSON,ApplicationLogLevel=ERROR,SystemLogLevel=WARN
```

Configurazione del filtraggio a livello di log durante la creazione di una funzione

- Per configurare il filtraggio a livello di log quando crei una nuova funzione, utilizza `--logging-config` per impostare le chiavi `SystemLogLevel` e `ApplicationLogLevel` nel comando [create-function](#). Imposta `SystemLogLevel` su `DEBUG`, `INFO` o `WARN`. Imposta `ApplicationLogLevel` su `DEBUG`, `INFO`, `WARN`, `ERROR` o `FATAL`.

```
aws lambda create-function \  
  --function-name myFunction \  
  --runtime nodejs24.x \  
  --handler index.handler \  
  --zip-file fileb://function.zip \  
  --role arn:aws:iam::123456789012:role/LambdaRole \  
  --logging-config LogFormat=JSON,ApplicationLogLevel=ERROR,SystemLogLevel=WARN
```

Strumento di mappatura degli eventi a livello di log di sistema

Per gli eventi di log a livello di sistema generati da Lambda, la tabella seguente definisce il livello di log assegnato a ciascun evento. Per ulteriori informazioni sugli eventi elencati nella tabella, consulta la pagina [the section called “Riferimento allo schema Event”](#)

| Nome evento              | Condizione                     | Livello di log assegnato |
|--------------------------|--------------------------------|--------------------------|
| initStart                | runtimeVersion è impostato     | INFO                     |
| initStart                | runtimeVersion non è impostato | DEBUG                    |
| init RuntimeDone         | status=success                 | DEBUG                    |
| init RuntimeDone         | status=success                 | WARN                     |
| initReport               | initializationType!=on-demand  | INFO                     |
| initReport               | initializationType=on-demand   | DEBUG                    |
| initReport               | status=success                 | WARN                     |
| restoreStart             | runtimeVersion è impostato     | INFO                     |
| restoreStart             | runtimeVersion non è impostato | DEBUG                    |
| ripristinare RuntimeDone | status=success                 | DEBUG                    |
| ripristinare RuntimeDone | status=success                 | WARN                     |
| restoreReport            | status=success                 | INFO                     |
| restoreReport            | status=success                 | WARN                     |
| rapida                   | -                              | INFO                     |
| runtimeDone              | status=success                 | DEBUG                    |
| runtimeDone              | status=success                 | WARN                     |
| report                   | status=success                 | INFO                     |
| report                   | status=success                 | WARN                     |
| Estensione               | state=success                  | INFO                     |

| Nome evento           | Condizione     | Livello di log assegnato |
|-----------------------|----------------|--------------------------|
| Estensione            | state!=success | WARN                     |
| logSubscription       | -              | INFO                     |
| telemetrySubscription | -              | INFO                     |
| logsDropped           | -              | WARN                     |

### Note

L'[the section called “API di telemetria”](#) emette sempre il set completo di eventi della piattaforma. La configurazione del livello dei log di sistema a cui Lambda invia CloudWatch non influisce sul comportamento dell'API Lambda Telemetry.

## Filtraggio delle applicazioni a livello di log con runtime personalizzati

Quando configuri il filtraggio a livello di log dell'applicazione per la tua funzione, dietro le quinte Lambda imposta il livello di log dell'applicazione nel runtime utilizzando la variabile di ambiente `AWS_LAMBDA_LOG_LEVEL`. Lambda imposta anche il formato di log della funzione utilizzando la variabile di ambiente `AWS_LAMBDA_LOG_FORMAT`. Puoi utilizzare queste variabili per integrare i controlli di registrazione avanzati di Lambda in un [runtime personalizzato](#).

Per poter configurare le impostazioni di registrazione per una funzione utilizzando un runtime personalizzato con la console Lambda e le API Lambda AWS CLI, configura il tuo runtime personalizzato per verificare il valore di queste variabili di ambiente. È quindi possibile configurare i logger del runtime in base al formato di log e ai livelli di log selezionati.

## Invio dei log delle funzioni Lambda ai CloudWatch log

Per impostazione predefinita, Lambda acquisisce automaticamente i log per tutte le chiamate di funzione e li invia a CloudWatch Logs, a condizione che il ruolo di esecuzione della funzione disponga delle autorizzazioni necessarie. Questi log sono, per impostazione predefinita, archiviati in un gruppo di log denominato `//. aws/lambda <function-name>`. Per migliorare il debug, puoi inserire istruzioni di registrazione personalizzate nel tuo codice, che Lambda integra perfettamente con Logs. CloudWatch. Se necessario, puoi configurare la tua funzione per inviare i log a un gruppo

diverso utilizzando la console Lambda o l'API Lambda. AWS CLI Per ulteriori informazioni, consulta [the section called “Configurare i log CloudWatch delle funzioni”](#).

Puoi visualizzare i log per le funzioni Lambda utilizzando la console Lambda, la CloudWatch console, la AWS Command Line Interface (AWS CLI) o l'API. CloudWatch Per ulteriori informazioni, consulta to. [Visualizzazione dei CloudWatch log per le funzioni Lambda](#)

#### Note

La visualizzazione dei log dopo l'invocazione di una funzione può richiedere da 5 a 10 minuti.

### Autorizzazioni IAM richieste

Il tuo ruolo di [esecuzione](#) richiede le seguenti autorizzazioni per caricare i log in Logs: CloudWatch

- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`

Per ulteriori informazioni, consulta [Using Identity-Based Policy \(politiche IAM\) for CloudWatch Logs](#) nella Amazon User Guide. CloudWatch

Puoi aggiungere queste autorizzazioni per CloudWatch i log utilizzando la policy `AWSLambdaBasicExecutionRole` AWS gestita fornita da Lambda. Per aggiungere questa policy al ruolo, esegui il seguente comando:

```
aws iam attach-role-policy --role-name your-role --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

Per ulteriori informazioni, consulta [the section called “AWS politiche gestite”](#).

### Prezzi

Non sono previsti costi aggiuntivi per l'utilizzo dei log Lambda; tuttavia, si applicano le tariffe standard CloudWatch dei log. Per ulteriori informazioni, consultare [Prezzi di CloudWatch](#).

## Configurazione dei gruppi di CloudWatch log

Per impostazione predefinita, crea CloudWatch automaticamente un gruppo di log denominato `/aws/lambda/<function name>` per la funzione quando viene richiamata per la prima volta. Per configurare la tua funzione per l'invio dei log a un gruppo di log esistente o per creare un nuovo gruppo di log per la funzione, puoi utilizzare la console Lambda o la AWS CLI. Puoi anche configurare gruppi di log personalizzati utilizzando i comandi API [CreateFunction](#) e [UpdateFunctionConfiguration](#) Lambda e la risorsa AWS Serverless Application Model (AWS SAM) [AWS::Serverless::Function](#).

È possibile configurare più funzioni Lambda per inviare i log allo stesso gruppo di log. CloudWatch Ad esempio, è possibile utilizzare un singolo gruppo di log per archiviare i log per tutte le funzioni Lambda che costituiscono una particolare applicazione. Quando si utilizza un gruppo di log personalizzato per una funzione Lambda, i flussi di log creati da Lambda includono il nome e la versione della funzione. Ciò garantisce che la mappatura tra i messaggi di log e le funzioni venga preservata, anche se si utilizza lo stesso gruppo di log per più funzioni.

Il formato di denominazione per i flussi di log per i gruppi di log personalizzati segue questa convenzione:

```
YYYY/MM/DD/<function_name>[<function_version>][<execution_environment_GUID>]
```

Tieni presente che quando configuri un gruppo di log personalizzato, il nome selezionato per il tuo gruppo di log deve seguire le regole di denominazione dei [CloudWatch log](#). Inoltre, i nomi dei gruppi di log personalizzati non devono cominciare con la stringa `aws/`. Se crei un gruppo di log personalizzato che comincia con `aws/`, Lambda non sarà in grado di crearlo. Di conseguenza, i log della tua funzione non verranno inviati a CloudWatch.

### Modifica del gruppo di log di una funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Nella pagina di configurazione della funzione, scegli Strumenti di monitoraggio e gestione.
4. Nel riquadro Configurazione della registrazione, scegli Modifica.
5. Nel riquadro del gruppo di registrazione, per il gruppo di CloudWatch log, scegli Personalizzato.
6. In Gruppo di log personalizzato, inserisci il nome del gruppo di CloudWatch log a cui desideri che la funzione invii i log. Se immetti il nome di un gruppo di log esistente, la funzione utilizzerà

quel gruppo. Se non esiste alcun gruppo di log con il nome immesso, Lambda creerà un nuovo gruppo di log per la funzione con tale nome.

Per modificare il gruppo di log di una funzione (AWS CLI)

- Per modificare il gruppo di log di una funzione esistente, utilizza il comando [update-function-configuration](#).

```
aws lambda update-function-configuration \  
  --function-name myFunction \  
  --logging-config LogGroup=myLogGroup
```

Per specificare un gruppo di log personalizzato quando si crea una funzione (AWS CLI)

- Per specificare un gruppo di log personalizzato quando crei una nuova funzione Lambda utilizzando AWS CLI, usa l'--logging-config opzione. Il seguente comando di esempio crea una funzione Node.js Lambda che invia i log a un gruppo di log denominato. myLogGroup

```
aws lambda create-function \  
  --function-name myFunction \  
  --runtime nodejs24.x \  
  --handler index.handler \  
  --zip-file fileb://function.zip \  
  --role arn:aws:iam::123456789012:role/LambdaRole \  
  --logging-config LogGroup=myLogGroup
```

## Autorizzazioni del ruolo di esecuzione

Affinché la funzione possa inviare i CloudWatch log a Logs, deve disporre dell'PutLogEvents [autorizzazione](#) logs:. Quando configuri il gruppo di log della tua funzione utilizzando la console Lambda, Lambda aggiunge questa autorizzazione al ruolo nelle seguenti condizioni:

- La destinazione del servizio è impostata su Logs CloudWatch
- Il ruolo di esecuzione della tua funzione non dispone delle autorizzazioni per caricare i log su CloudWatch Logs (la destinazione predefinita)

 Note

Lambda non aggiunge alcuna autorizzazione Put per le destinazioni di log di Amazon S3 o Firehose.

Quando Lambda aggiunge questa autorizzazione, concede alla funzione il permesso di inviare log a qualsiasi CloudWatch gruppo di log Logs.

Per evitare che Lambda aggiorni automaticamente il ruolo di esecuzione della funzione e modificarlo invece manualmente, espandi Autorizzazioni e deseleziona Aggiungi autorizzazioni richieste.

Quando configuri il gruppo di log della tua funzione utilizzando il AWS CLI, Lambda non aggiungerà automaticamente l'`logs:PutLogEvents` autorizzazione. Aggiungi l'autorizzazione al ruolo di esecuzione della tua funzione, se non ne dispone già. Questa autorizzazione è inclusa nella policy [AWSLambdaBasicExecutionRole](#) gestita.

### CloudWatch registrazione per Lambda Managed Instances

Quando si utilizzano le istanze gestite [Lambda](#), ci sono ulteriori considerazioni per l'invio dei log ai log: CloudWatch

### Requisiti di rete VPC

Le istanze gestite Lambda vengono eseguite su istanze EC2 di proprietà del cliente all'interno del tuo VPC. Per inviare i CloudWatch log a Logs e le tracce a X-Ray, devi assicurarti che queste AWS API siano instradabili dal tuo VPC. A questo scopo, sono disponibili numerose opzioni:

- AWS PrivateLink (consigliato): [AWS PrivateLink](#) da utilizzare per creare endpoint VPC per log e servizi. CloudWatch X-Ray Ciò consente alle istanze di accedere a questi servizi privatamente senza richiedere un gateway Internet o un gateway NAT. Per ulteriori informazioni, consulta [Utilizzo dei CloudWatch log con gli endpoint VPC dell'interfaccia](#).
- Gateway NAT: configura un gateway NAT per consentire l'accesso a Internet in uscita dalle tue sottoreti private.
- Gateway Internet: per le sottoreti pubbliche, assicurati che sul tuo VPC sia configurato un gateway Internet.

Se CloudWatch i log o le X-Ray API non sono instradabili dal tuo VPC, i log e le tracce delle funzioni non vengono consegnati.



## Richiami simultanei e attribuzione dei log

Gli ambienti di esecuzione di Lambda Managed Instances possono elaborare più invocazioni contemporaneamente. Quando più invocazioni vengono eseguite contemporaneamente, le relative voci di registro vengono interlacciate nello stesso flusso di log. Per filtrare e analizzare in modo efficace i log provenienti da chiamate simultanee, è necessario assicurarsi che ogni voce di registro includa l'ID della richiesta. AWS

Consigliamo uno dei seguenti approcci:

- Usa i logger di runtime Lambda predefiniti (consigliato): le librerie di registrazione predefinite fornite dai runtime gestiti da Lambda includono automaticamente l'ID della richiesta in ogni voce di registro.
- Implementa la registrazione JSON strutturata: se stai creando un runtime [personalizzato](#) o hai bisogno di una registrazione personalizzata, implementa i JSON-formatted log che includono l'ID della richiesta in ogni voce. Le istanze gestite Lambda supportano solo il formato di log JSON. Includi il `requestId` campo nei tuoi log JSON per abilitare il filtraggio per invocazione:

```
{
  "timestamp": "2025-01-15T10:30:00.000Z",
  "level": "INFO",
  "requestId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "message": "Processing request"
}
```

Con l'attribuzione dell'ID della richiesta, puoi filtrare le voci di CloudWatch log di Logs per una chiamata specifica utilizzando le query Logs Insights. CloudWatch Ad esempio:

```
fields @timestamp, @message
| filter requestId = "a1b2c3d4-e5f6-7890-abcd-ef1234567890"
| sort @timestamp asc
```

Per ulteriori informazioni sui requisiti di registrazione di Lambda Managed Instances, consulta.

[Comprendere l'ambiente di esecuzione delle istanze gestite Lambda](#)

## Visualizzazione dei CloudWatch log per le funzioni Lambda

Puoi visualizzare CloudWatch i log di Amazon per la tua funzione Lambda utilizzando la console Lambda, la CloudWatch console o AWS Command Line Interface (AWS CLI). Segui le istruzioni riportate nelle sezioni seguenti per accedere ai log della funzione.

Trasmetti in streaming i log delle funzioni con Logs Live CloudWatch Tail

Amazon CloudWatch Logs Live Tail ti aiuta a risolvere rapidamente i problemi delle tue funzioni visualizzando un elenco in streaming di nuovi eventi di registro direttamente nella console Lambda. Puoi visualizzare e filtrare i log importati dalle funzioni Lambda in tempo reale, in modo da poter rilevare e risolvere rapidamente i problemi.

### Note

Le sessioni Live Tail comportano costi al minuto in base al tempo di utilizzo della sessione. Per ulteriori informazioni sui prezzi, consulta [Amazon Pricing. CloudWatch](#)

## Confronto tra Live Tail e --log-type Tail

Esistono diverse differenze tra CloudWatch Logs Live Tail e l'[opzione](#) LogType: Tail nell'API Lambda (--log-type Tail nell'AWS CLI):

- --log-type Tail restituisce solo i primi 4 KB dei log di invocazione. Live Tail non condivide questo limite e può ricevere fino a 500 eventi del log al secondo.
- --log-type Tail acquisisce e invia i log con la risposta, il che può influire sulla latenza di risposta della funzione. Live Tail non influisce sulla latenza di risposta della funzione.
- --log-type Tail supporta solo chiamate sincrone. Live Tail funziona sia per le invocazioni sincrone che asincrone.

### Note

[Lambda Managed Instances non](#) supporta l'opzione. --log-type Tail Usa CloudWatch Logs Live Tail o interroga direttamente CloudWatch i log per visualizzare i log per le funzioni di Managed Instances.

## Permissions

Le seguenti autorizzazioni sono necessarie per avviare e interrompere le sessioni di CloudWatch Logs Live Tail:

- `logs:DescribeLogGroups`
- `logs:StartLiveTail`
- `logs:StopLiveTail`

### Avviare una sessione Live Tail nella console Lambda

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere il nome della funzione.
3. Seleziona la scheda Test.
4. Nel riquadro degli eventi di test, scegli CloudWatch Logs Live Tail.
5. Per Seleziona gruppi di log, il gruppo di log della funzione è selezionato per impostazione predefinita. È possibile selezionare fino a cinque gruppi di log alla volta.
6. (Facoltativo) Per visualizzare solo gli eventi del log che contengono determinate parole o altre stringhe, inserisci la parola o la stringa nella casella Aggiungi modello di filtro. I filtri fanno distinzione tra maiuscole e minuscole. Puoi includere più termini e operatori di modelli in questo campo, comprese le espressioni regolari (regex). Per ulteriori informazioni sulla sintassi dei pattern, consulta [Filter pattern syntax](#) nella Amazon CloudWatch Logs User Guide.
7. Scegli Avvia. I log eventi corrispondenti vengono visualizzati nella finestra.
8. Per arrestare la sessione Live Tail, scegli Arresta.

#### Note

La sessione Live Tail si interrompe automaticamente dopo 15 minuti di inattività o quando la sessione della console Lambda scade.

### Accedere ai log della funzione tramite la console

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Seleziona una funzione.

3. Selezionare la scheda Monitor (Monitora).
4. Scegli Visualizza CloudWatch i log per aprire la console. CloudWatch
5. Scorri verso il basso e scegli il flusso di log per le invocazioni delle funzioni che desideri esaminare.

| Log streams (14)                                                     |                                           |                                      | Delete                                | Create log stream    | Search all log streams |
|----------------------------------------------------------------------|-------------------------------------------|--------------------------------------|---------------------------------------|----------------------|------------------------|
| <input type="text" value="Filter log streams or try prefix search"/> |                                           | <input type="checkbox"/> Exact match | <input type="checkbox"/> Show expired | <a href="#">Info</a> | < 1 >                  |
| <input type="checkbox"/>                                             | Log stream                                | Last event time                      |                                       |                      |                        |
| <input type="checkbox"/>                                             | <a href="#">2024/04/30/[\$LATEST]e0fa</a> | 2024-04-30 17:24:16 (UTC)            |                                       |                      |                        |
| <input type="checkbox"/>                                             | <a href="#">2024/04/19/[\$LATEST]e9a</a>  | 2024-04-19 20:59:06 (UTC)            |                                       |                      |                        |
| <input type="checkbox"/>                                             | <a href="#">2024/02/22/[\$LATEST]cf0</a>  | 2024-02-22 18:38:41 (UTC)            |                                       |                      |                        |
| <input type="checkbox"/>                                             | <a href="#">2024/02/21/[1]d132c4d</a>     | 2024-02-21 21:37:01 (UTC)            |                                       |                      |                        |
| <input type="checkbox"/>                                             | <a href="#">2024/02/21/[1]5ad</a>         | 2024-02-21 21:37:01 (UTC)            |                                       |                      |                        |

Ogni istanza di una funzione Lambda ha un flusso di log dedicato. Se una funzione aumenta, ogni istanza simultanea ha il suo flusso di log. Ogni volta che viene creato un nuovo ambiente di esecuzione in risposta a una chiamata, viene generato un nuovo flusso di log. La convenzione di denominazione per i flussi di log è:

```
YYYY/MM/DD[Function version][Execution environment GUID]
```

Un singolo ambiente di esecuzione scrive sullo stesso flusso di log durante il suo ciclo di vita. Il flusso di log contiene i messaggi provenienti da quell'ambiente di esecuzione e anche qualsiasi output dal codice della funzione Lambda. Ogni messaggio ha un timestamp, compresi i log personalizzati. Anche se la funzione non registra alcun output dal codice, vengono generate tre istruzioni di log minime per ogni invocazione (START, END e REPORT):

|   |                               |                                                                                                                        |                                                                                                                                                                          |                      |
|---|-------------------------------|------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|
| ▼ | 2020-10-08T15:52:11.447-04:00 | START RequestId: 345a1711-d325-4af6-b01f-b0648975743f Version: \$LATEST                                                | START RequestId: 345a1711-d325-4af6-b01f-b0648975743f Version: \$LATEST                                                                                                  | <a href="#">Copy</a> |
| ▼ | 2020-10-08T15:52:12.452-04:00 | END RequestId: 345a1711-d325-4af6-b01f-b0648975743f                                                                    | END RequestId: 345a1711-d325-4af6-b01f-b0648975743f                                                                                                                      | <a href="#">Copy</a> |
| ▼ | 2020-10-08T15:52:12.452-04:00 | REPORT RequestId: 345a1711-d325-4af6-b01f-b0648975743f Duration: 1004.58 ms Billed Duration: 1100 ms Memory Size: 1... | REPORT RequestId: 345a1711-d325-4af6-b01f-b0648975743f Duration: 1004.58 ms Billed Duration: 1100 ms Memory Size: 128 MB Max Memory Used: 64 MB Init Duration: 295.85 ms | <a href="#">Copy</a> |

Questi log mostrano:

- **RequestId**— si tratta di un ID univoco generato per richiesta. Se la funzione Lambda ritenta una richiesta, questo ID non cambia e viene visualizzato nei log per ogni tentativo successivo.
- **Start/End**— contrassegnano una singola chiamata ai segnalibri, quindi ogni riga di registro tra queste appartiene alla stessa invocazione.
- **Durata**: il tempo totale di chiamata per la funzione handler, codice escluso. **INIT**
- **Durata fatturata**: applica la logica di arrotondamento ai fini della fatturazione.
- **Dimensione della memoria**: la quantità di memoria allocata alla funzione.
- **Memoria massima utilizzata**: la quantità massima di memoria utilizzata durante l'invocazione.
- **Init Duration**: il tempo impiegato per eseguire la **INIT** sezione di codice, al di fuori del gestore principale.

## Registri di accesso con AWS CLI

AWS CLI È uno strumento open source che puoi utilizzare per interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della [AWS CLI versione 2](#).

È possibile utilizzare [AWS CLI](#) per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

### Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
  "StatusCode": 200,
  "LogResult":
    "U1RBULQgUmVxdWVzdElk0iA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2l1vb...",
  "ExecutedVersion": "$LATEST"
}
```

## Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'base64 utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

L'`cli-binary-format` opzione è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
Duration: 80 ms      Memory Size: 128 MB      Max Memory Used: 73 MB
```

L'utilità `base64` è disponibile su Linux, macOS e [Ubuntu su Windows](#). Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

## Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'`cli-binary-format` opzione è obbligatoria se stai usando la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

### Example(solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il comando seguente per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

## Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version: $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r ...",
```

```

        "ingestionTime": 1559763018353
    },
    {
        "timestamp": 1559763003173,
        "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
        "ingestionTime": 1559763018353
    },
    {
        "timestamp": 1559763003218,
        "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
        "ingestionTime": 1559763018353
    },
    {
        "timestamp": 1559763003218,
        "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
MB\t\n",
        "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
  "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}

```

## Analisi dei log e registrazione strutturata

Con CloudWatch Logs Insights, puoi cercare e analizzare i dati di registro utilizzando una sintassi di [query specializzata](#). Esegue interrogazioni su più gruppi di log e fornisce potenti filtri utilizzando il pattern matching tra espressioni [glob](#) e [regolari](#).

Puoi sfruttare queste funzionalità implementando la registrazione strutturata nelle tue funzioni Lambda. La registrazione strutturata organizza i log in un formato predefinito, facilitandone l'interrogazione. L'utilizzo dei livelli di log è un primo passo importante per generare registri facili da filtrare che separano i messaggi informativi da avvisi o errori. Ad esempio, considera il codice seguente: Node.js

```

exports.handler = async (event) => {
  console.log("console.log - Application is fine")
  console.info("console.info - This is the same as console.log")
  console.warn("console.warn - Application provides a warning")
  console.error("console.error - An error occurred")
}

```



```
}
```

Il file di CloudWatch registro risultante contiene un campo separato che specifica il livello di registro:

```
START RequestId: 99d91f9b-2dff-40ad-b9c8-664020094109 Version: $LATEST
2020-10-22T12:21:28.268Z      99d91f9b-2dff-40ad-b9c8-664020094109  INFO  console.log - Application is fine
2020-10-22T12:21:28.268Z      99d91f9b-2dff-40ad-b9c8-664020094109  INFO  console.info - This is the same as console.log
2020-10-22T12:21:28.268Z      99d91f9b-2dff-40ad-b9c8-664020094109  WARN  console.warn - Application provides a warning
2020-10-22T12:21:28.268Z      99d91f9b-2dff-40ad-b9c8-664020094109  ERROR  console.error - An error occurred
END RequestId: 99d91f9b-2dff-40ad-b9c8-664020094109
REPORT RequestId: 99d91f9b-2dff-40ad-b9c8-664020094109 Duration: 3.13 ms      Billed Duration: 100 ms Memory Size: 128 MB
Max Memory Used: 64 MB Init Duration: 142.18 ms
```

Una query CloudWatch Logs Insights può quindi filtrare a livello di registro. Ad esempio, per ricercare solo gli errori, puoi utilizzare la seguente query:

```
fields @timestamp, @message
| filter @message like /ERROR/
| sort @timestamp desc
```

## Registrazione strutturata JSON

JSON è comunemente usato per fornire una struttura per i log delle applicazioni. Nell'esempio seguente, i log sono stati convertiti in JSON per generare tre valori distinti:

```
START RequestId: 22fda338-7b46-4c49-9531-efd6e8568480 Version: $LATEST
2020-10-22T11:35:59.216Z      22fda338-7b46-4c49-9531-efd6e8568480  INFO  { uploadedBytes: 5453396,
invocation: 5, uploadTimeMS: 4519 }
END RequestId: 22fda338-7b46-4c49-9531-efd6e8568480
REPORT RequestId: 22fda338-7b46-4c49-9531-efd6e8568480 Duration: 1.25 ms      Billed Duration: 100 ms Memory
Size: 128 MB      Max Memory Used: 65 MB
```

La funzione CloudWatch Logs Insights rileva automaticamente i valori nell'output JSON e analizza i messaggi come campi, senza la necessità di globuli o espressioni regolari personalizzati. Utilizzando JSON-structured i log, la seguente query trova le chiamate in cui il file caricato era più grande di 1 MB, il tempo di caricamento era superiore a 1 secondo e l'invocazione non era un avvio a freddo:

```
fields @message
| filter @message like /INFO/
| filter uploadedBytes > 1000000
| filter uploadTimeMS > 1000
| filter invocation != 1
```

Questa query potrebbe produrre il seguente risultato:

The screenshot shows the AWS CloudWatch Logs Insights console. On the left is a navigation menu with options like CloudWatch, Dashboards, Alarms, and Insights. The main area displays a query editor with the following query:

```
fields @message
| filter @message like /INFO/
| filter uploadedBytes > 1000000
| filter uploadTimeMS > 1000
| filter invocation != 1
```

Below the query editor, the results are shown in a table with columns #, @timestamp, @logStream, @message, and @type. The table displays 6 records of log events.

On the right side, a 'Discovered fields' panel lists various fields found in the logs, such as @ingestionTime, @logStream, @message, @timestamp, @requestId, @type, uploadTime, @billedDuration, @duration, @maxMemoryUsed, @memorySize, invocation, uploadedBytes, uploadTimeMS, and @initDuration. The fields 'invocation', 'uploadedBytes', and 'uploadTimeMS' are highlighted with a red box.

I campi scoperti in JSON vengono compilati automaticamente nel menu Campi scoperti sul lato destro. I campi standard emessi dal servizio Lambda hanno il prefisso '@' e puoi eseguire query su questi campi allo stesso modo. I log Lambda includono sempre i campi @timestamp, @logStream, @message, @requestId, @duration, @billedDuration, @type,,,,, @memorySize. MemoryUsed @max Se X-Ray è abilitato per una funzione, i log includono anche IsN TraceId e LM. SegmentId @xray @xray

Quando una fonte di AWS eventi come Amazon S3, Amazon SQS o Amazon EventBridge richiama la tua funzione, l'intero evento viene fornito alla funzione come input di un oggetto JSON. Registrando questo evento nella prima riga della funzione, puoi quindi interrogare uno qualsiasi dei campi annidati utilizzando Logs Insights. CloudWatch

## Query utili di Insights

La tabella seguente mostra esempi di query Insights che possono essere utili per il monitoraggio delle funzioni Lambda.

| Description                                        | Esempio di sintassi della query                                                                              |
|----------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| Gli ultimi 100 errori                              | <pre>fields Timestamp, LogLevel, Message   filter LogLevel == "ERR"   sort @timestamp desc   limit 100</pre> |
| Le prime 100 invocazioni con il fatturato più alto | <pre>filter @type = "REPORT"</pre>                                                                           |

| Description                                                | Esempio di sintassi della query                                                                                                                                                                                                                                 |
|------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                            | <pre>  fields @requestId, @billedDuration   sort by @billedDuration desc   limit 100</pre>                                                                                                                                                                      |
| Percentuale di avvii a freddo sul totale delle invocazioni | <pre>filter @type = "REPORT"   stats sum(strcontains(@message,   "Init Duration"))/count(*) * 100 as   coldStartPct, avg(@duration)   by bin(5m)</pre>                                                                                                          |
| Rapporto percentile della durata Lambda                    | <pre>filter @type = "REPORT"   stats   avg(@billedDuration) as Average,   percentile(@billedDuration, 99)   as NinetyNinth,   percentile(@billedDuration, 95)   as NinetyFifth,   percentile(@billedDuration, 90)   as Ninetieth   by bin(30m)</pre>            |
| Rapporto percentile sull'utilizzo della memoria Lambda     | <pre>filter @type="REPORT"   stats avg(@maxMemoryUsed/1024/1024) as mean_MemoryUsed,   min(@maxMemoryUsed/1024/1024) as   min_MemoryUsed,   max(@maxMemoryUsed/1024/1024) as   max_MemoryUsed,   percentile(@maxMemoryUsed/1024/1024, 95) as Percentile95</pre> |
| Invocazioni che utilizzano il 100% della memoria assegnata | <pre>filter @type = "REPORT" and @maxMemoryUsed=@memorySize   stats   count_distinct(@requestId)   by bin(30m)</pre>                                                                                                                                            |

| Description                                     | Esempio di sintassi della query                                                                                                                                                                                                                                                                                                                                            |
|-------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Memoria media utilizzata tra le invocazioni     | <pre>avgMemoryUsedPERC,     avg(@billedDuration) as avgDurationMS     by bin(5m)</pre>                                                                                                                                                                                                                                                                                     |
| Visualizzazione delle statistiche sulla memoria | <pre>filter @type = "REPORT"   stats     max(@maxMemoryUsed / 1024 / 1024)     as maxMemMB,     avg(@maxMemoryUsed / 1024 / 1024)     as avgMemMB,     min(@maxMemoryUsed / 1024 / 1024)     as minMemMB,     (avg(@maxMemoryUsed / 1024 / 1024) / max(@memorySize / 1024 / 1024)) * 100 as avgMemUsedPct,     avg(@billedDuration) as avgDurationMS     by bin(30m)</pre> |
| Invocazioni in cui Lambda è uscita              | <pre>filter @message like /Process exited/   stats count() by bin(30m)</pre>                                                                                                                                                                                                                                                                                               |
| Invocazioni scadute                             | <pre>filter @message like /Task timed out/   stats count() by bin(30m)</pre>                                                                                                                                                                                                                                                                                               |
| Rapporto sulla latenza                          | <pre>filter @type = "REPORT"   stats avg(@duration), max(@duration), min(@duration)     by bin(5m)</pre>                                                                                                                                                                                                                                                                   |

| Description              | Esempio di sintassi della query                                                                                                                                                                                                                                                                                                                    |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Over-provisioned memoria | <pre> filter @type = "REPORT"   stats max(@memorySize / 1024 / 1024) as provisionedMemMB,         min(@maxMemoryUsed / 1024 / 1024) as smallestMemReqMB,         avg(@maxMemoryUsed / 1024 / 1024) as avgMemUsedMB,         max(@maxMemoryUsed / 1024 / 1024) as maxMemUsedMB,         provisionedMemMB - maxMemUsedMB as overProvisionedMB </pre> |

## Visualizzazione dei log e pannelli di controllo

Per qualsiasi richiesta di CloudWatch Logs Insights, puoi esportare i risultati in formato markdown o CSV. In alcuni casi, potrebbe essere più utile creare [visualizzazioni a partire dalle query](#), a condizione che esista almeno una funzione di aggregazione. Con la `stats` funzione, è possibile definire aggregazioni e raggruppamenti.

L'esempio precedente di `logInsightsJSON` filtrava in base alla dimensione e al tempo di caricamento ed escludeva le prime invocazioni. Ciò ha prodotto una tabella di dati. Per monitorare un sistema di produzione, potrebbe essere più utile visualizzare le dimensioni minime, massime e medie dei file per individuare i valori anomali. Per fare ciò, applica la funzione `stats` agli aggregati richiesti e raggruppa in base a un valore temporale come ogni minuto:

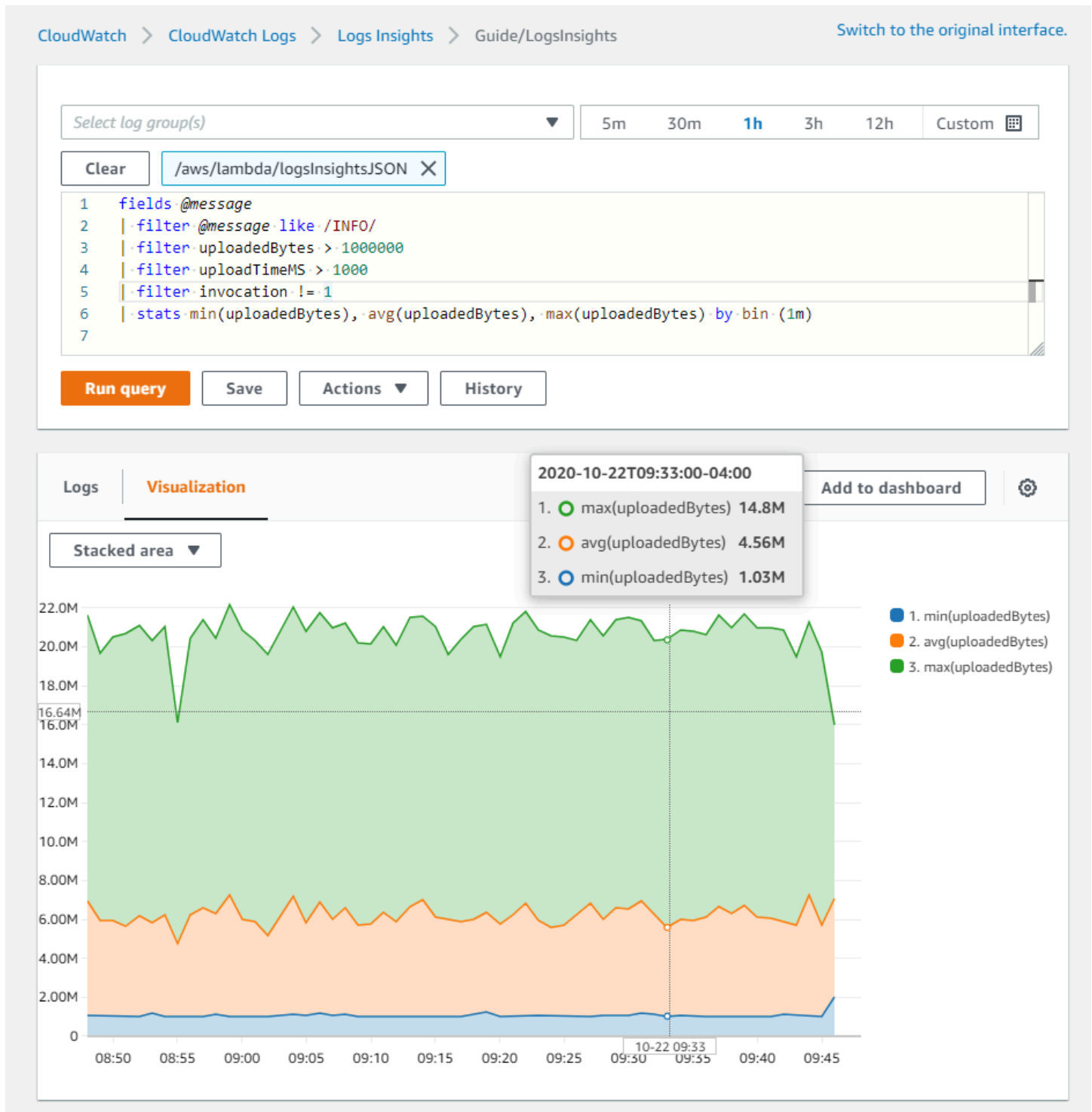
Ad esempio, considera la seguente query. Questa è la stessa query di esempio contenuta nella [the section called “Registrazione strutturata JSON”](#) sezione, ma con funzioni di aggregazione aggiuntive:

```

fields @message
| filter @message like /INFO/
| filter uploadedBytes > 1000000
| filter uploadTimeMS > 1000
| filter invocation != 1
| stats min(uploadedBytes), avg(uploadedBytes), max(uploadedBytes) by bin (1m)

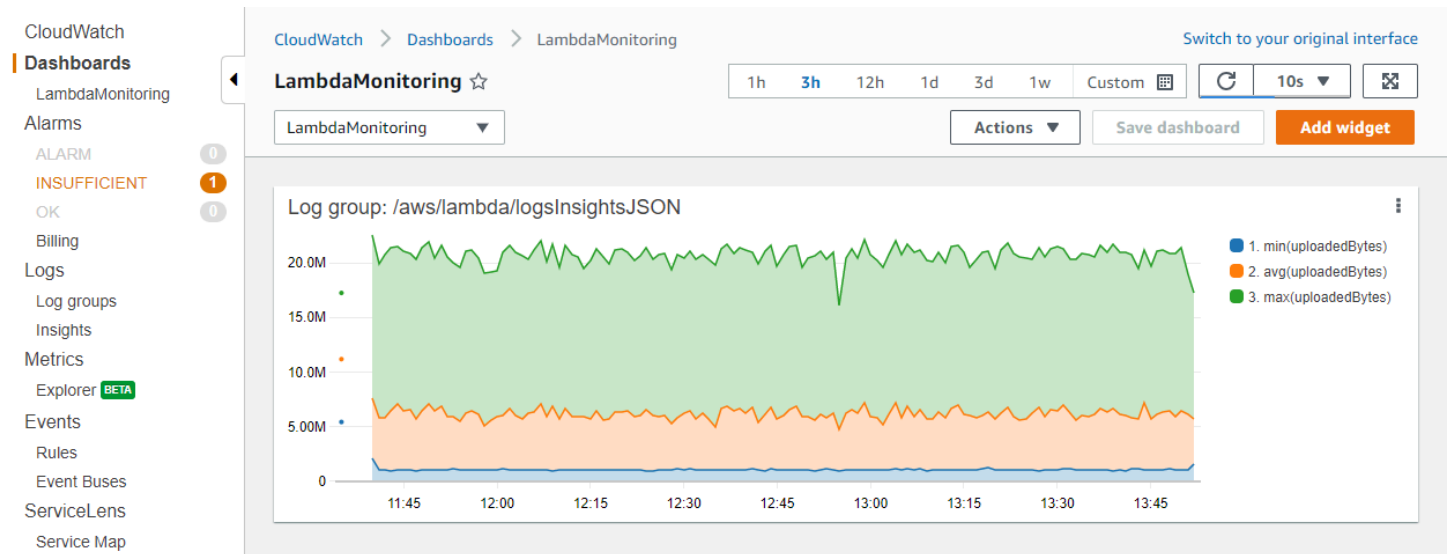
```

Abbiamo incluso questi aggregati perché potrebbe essere più utile visualizzare le dimensioni minime, massime e medie dei file per trovare valori anomali. Puoi visualizzare i risultati nella scheda Visualizzazione:



Dopo aver completato la creazione della visualizzazione, puoi facoltativamente aggiungere il grafico a una dashboard. CloudWatch Per fare ciò, scegli Aggiungi al pannello di controllo sopra la

visualizzazione. In questo modo la query viene aggiunta come widget ed è possibile selezionare intervalli di aggiornamento automatici, semplificando il monitoraggio continuo dei risultati:



## Invio dei log delle funzioni Lambda a Firehose

La console Lambda offre ora la possibilità di inviare i log delle funzioni a Firehose. Ciò consente lo streaming in tempo reale dei log verso varie destinazioni supportate da Firehose, inclusi strumenti di analisi di terze parti ed endpoint personalizzati.

### Note

È possibile configurare i log delle funzioni Lambda da inviare a Firehose utilizzando la console Lambda e tutti gli AWS CLI SDK. AWS CloudFormation AWS

## Prezzi

Per informazioni dettagliate sui prezzi, consulta i prezzi di Amazon. [CloudWatch](#)

## Autorizzazioni richieste per la destinazione dei log di Firehose

Quando si utilizza la console Lambda per configurare Firehose come destinazione del registro della funzione, è necessario:

1. Le autorizzazioni IAM [richieste](#) per utilizzare CloudWatch Logs con Lambda.
2. Per [configurare i filtri di abbonamento con Firehose](#). Questo filtro definisce quali eventi di registro vengono inviati al tuo stream Firehose.

## Invio dei log delle funzioni Lambda a Firehose

Nella console Lambda, è possibile inviare i log delle funzioni direttamente a Firehose dopo aver creato una nuova funzione. A tale scopo, completa questi passaggi:

1. Accedi alla console di AWS gestione e apri la console Lambda.
2. Scegli il nome della tua funzione.
3. Scegli la scheda Configurazione.
4. Scegli la scheda Strumenti di monitoraggio e operazioni.
5. Nel riquadro Configurazione della registrazione, scegli Modifica.
6. Nella sezione «Contenuto del registro», seleziona un formato di registro.
7. Nella sezione «Destinazione del registro», completa i seguenti passaggi:
  - a. Seleziona un servizio di destinazione.
  - b. Scegli un gruppo di log esistente o creane uno nuovo.

### Note

Se scegli un gruppo di log esistente per una destinazione Firehose, assicurati che il gruppo di log scelto sia un tipo di gruppo di Delivery log.

- c. Scegli uno stream Firehose.
  - d. Apparirà il gruppo di CloudWatch Delivery log.
8. Scegli Save (Salva).

### Note

Se il ruolo IAM fornito nella console non dispone dell'autorizzazione richiesta, la configurazione della destinazione non riesce. Per risolvere questo problema, fai riferimento a Autorizzazioni richieste per la destinazione del registro di Firehose per fornire le autorizzazioni richieste.



## Cross-Account Registrazione

È possibile configurare Lambda per inviare i log al flusso di distribuzione di Firehose in un altro account. AWS Ciò richiede l'impostazione di una destinazione e la configurazione delle autorizzazioni appropriate in entrambi gli account.

Per istruzioni dettagliate sulla configurazione della registrazione tra più account, inclusi i ruoli e le policy IAM richiesti, consulta [Configurazione di una nuova sottoscrizione](#) tra account nella documentazione di Logs. CloudWatch

## Invio di log delle funzioni Lambda ad Amazon S3

Puoi configurare la tua funzione Lambda per inviare i log direttamente ad Amazon S3 utilizzando la console Lambda. Questa funzionalità offre una soluzione conveniente per l'archiviazione di log a lungo termine e consente potenti opzioni di analisi utilizzando servizi come Athena.

### Note

Puoi configurare i log delle funzioni Lambda da inviare ad Amazon S3 utilizzando la console Lambda e tutti gli SDK AWS CLI. AWS CloudFormation AWS

## Prezzi

Per informazioni dettagliate sui prezzi, consulta i prezzi di Amazon. [CloudWatch](#)

## Autorizzazioni richieste per la destinazione dei log di Amazon S3

Quando utilizzi la console Lambda per configurare Amazon S3 come destinazione del log della tua funzione, devi:

1. Le autorizzazioni IAM [richieste](#) per utilizzare CloudWatch Logs with Lambda.
2. A [Configura un filtro di sottoscrizioni Logs per inviare CloudWatch i log delle funzioni Lambda ad Amazon S3](#). Questo filtro definisce quali eventi di log vengono inviati al tuo bucket Amazon S3.

## Configura un filtro di sottoscrizioni Logs per inviare CloudWatch i log delle funzioni Lambda ad Amazon S3

Per inviare i CloudWatch log da Logs ad Amazon S3, devi creare un filtro di abbonamento. Questo filtro definisce quali eventi di log vengono inviati al tuo bucket Amazon S3. Il bucket Amazon S3 deve trovarsi nella stessa regione del gruppo di log.

### Creazione di un filtro di sottoscrizione per Amazon S3

1. Crea un bucket Amazon Simple Storage Service (Amazon S3). Ti consigliamo di utilizzare un bucket creato appositamente per Logs. CloudWatch Tuttavia, se intendi utilizzare un bucket esistente, puoi passare alla fase 2.

Esegui il comando seguente, sostituendo il segnaposto Regione con la Regione che desideri utilizzare:

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket2 --create-bucket-configuration LocationConstraint=region
```

#### Note

amzn-s3-demo-bucket2 è un esempio di nome di bucket Amazon S3. È riservato. Affinché questa procedura funzioni, è necessario sostituirla con il nome univoco del bucket Amazon S3.

Di seguito è riportato un output di esempio:

```
{
  "Location": "/amzn-s3-demo-bucket2"
}
```

2. Crea il ruolo IAM che CloudWatch concede a Logs l'autorizzazione a inserire dati nel tuo bucket Amazon S3. Questa policy include una chiave di contesto aws: SourceArn global condition per aiutare a prevenire il confuso problema di sicurezza secondario. Per ulteriori informazioni consulta [Prevenzione del problema "confused deputy"](#).
  - a. Usa un editor di testo per creare una politica di fiducia in un file `~/TrustPolicyForCWL.json` come segue:

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Condition": {
      "StringLike": {
        "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
      }
    },
    "Action": "sts:AssumeRole"
  }
}
```

- b. Utilizza il comando `create-role` per creare un ruolo IAM, specificando il file della policy di attendibilità. Nota del `Role.Arn` valore restituito, in quanto ti servirà in un passaggio successivo:

```
aws iam create-role \
  --role-name CWLtoS3Role \
  --assume-role-policy-document file://~/TrustPolicyForCWL.json
{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Statement": {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": "logs.amazonaws.com"
        },
        "Condition": {
          "StringLike": {
            "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
          }
        }
      }
    },
    "RoleId": "AA0IIAH450GAB4HC5F431",
    "CreateDate": "2015-05-29T13:46:29.431Z",
    "RoleName": "CWLtoS3Role",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/CWLtoS3Role"
  }
}
```

3. Crea una politica di autorizzazioni per definire le azioni che CloudWatch Logs può eseguire sul tuo account. In primo luogo, utilizza un editor di testo per creare una policy di autorizzazione in un file `~/PermissionsForCWL.json`:

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:PutObject"],
      "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket2/*"]
    }
  ]
}
```

Associa la policy di autorizzazioni al ruolo inserendo il comando seguente `put-role-policy`:

```
aws iam put-role-policy --role-name CWLtoS3Role --policy-name Permissions-Policy-For-S3 --policy-document file://~/PermissionsForCWL.json
```

4. Scegli un gruppo di log `Delivery` esistente o creane uno nuovo `Delivery`.

```
aws logs create-log-group --log-group-name my-logs --log-group-class DELIVERY --region REGION_NAME
```

5. Configurazione della destinazione `PutSubscriptionFilter`

```
aws logs put-subscription-filter
--log-group-name my-logs
--filter-name my-lambda-delivery
--filter-pattern ""
--destination-arn arn:aws:s3:::amzn-s3-demo-bucket2
--role-arn arn:aws:iam::123456789012:role/CWLtoS3Role
--region REGION_NAME
```

## Invio dei log delle funzioni Lambda ad Amazon S3

Nella console Lambda, puoi inviare i log delle funzioni direttamente ad Amazon S3 dopo aver creato una nuova funzione. A tale scopo, completa questi passaggi:

1. Accedi alla console di AWS gestione e apri la console Lambda.

2. Scegli il nome della tua funzione.
3. Scegli la scheda Configurazione.
4. Scegli la scheda Strumenti di monitoraggio e operazioni.
5. Nel riquadro Configurazione della registrazione, scegli Modifica.
6. Nella sezione «Contenuto del registro», seleziona un formato di registro.
7. Nella sezione «Destinazione del registro», completa i seguenti passaggi:
  - a. Seleziona un servizio di destinazione.
  - b. Scegli un gruppo di log esistente o creane uno nuovo.

 Note

Se scegli un gruppo di log esistente per una destinazione Amazon S3, assicurati che il gruppo di log scelto sia un tipo di gruppo di Delivery log.

- c. Scegli un bucket Amazon S3 come destinazione per i tuoi log di funzione.
  - d. Viene visualizzato il gruppo di CloudWatch Delivery log.
8. Scegli Save (Salva).

 Note

Se il ruolo IAM fornito nella console non dispone delle autorizzazioni richieste, la configurazione della destinazione non riesce. Per risolvere questo problema, consulta Autorizzazioni [richieste per la destinazione dei log di Amazon S3](#).

## Cross-Account Registrazione

Puoi configurare Lambda per inviare i log a un bucket Amazon S3 in un account diverso. AWS Ciò richiede l'impostazione di una destinazione e la configurazione delle autorizzazioni appropriate in entrambi gli account.

Per istruzioni dettagliate sulla configurazione della registrazione tra più account, inclusi i ruoli e le policy IAM richiesti, consulta [Configurazione di una nuova sottoscrizione](#) tra account nella documentazione di Logs. CloudWatch

# Registrazione dei log AWS Lambda Chiamate API utilizzando AWS CloudTrail

AWS Lambda è integrato con [AWS CloudTrail](#) un servizio che fornisce una registrazione delle azioni intraprese da un utente, un ruolo o un Servizio AWS. CloudTrail acquisisce le chiamate API per Lambda come eventi. Le chiamate acquisite includono le chiamate dalla console di Lambda e le chiamate di codice alle operazioni delle API di Lambda. Utilizzando le informazioni raccolte da CloudTrail, è possibile determinare la richiesta effettuata a Lambda, l'indirizzo IP da cui è stata effettuata la richiesta, quando è stata effettuata e dettagli aggiuntivi.

Ogni evento o voce di log contiene informazioni sull'utente che ha generato la richiesta. Le informazioni di identità consentono di determinare quanto segue:

- Se la richiesta è stata effettuata con le credenziali utente root o utente.
- Se la richiesta è stata effettuata per conto di un utente del Centro identità IAM.
- Se la richiesta è stata effettuata con le credenziali di sicurezza temporanee per un ruolo o un utente federato.
- Se la richiesta è stata effettuata da un altro Servizio AWS.

CloudTrail è attivo nel tuo account Account AWS quando crei l'account e hai automaticamente accesso alla cronologia degli CloudTrail eventi. La cronologia CloudTrail degli eventi fornisce un record visualizzabile, ricercabile, scaricabile e immutabile degli ultimi 90 giorni di eventi di gestione registrati in un. Regione AWS Per ulteriori informazioni, vedere [Utilizzo della cronologia degli CloudTrail eventi nella Guida per l'utente](#). AWS CloudTrail La visualizzazione della cronologia degli eventi è gratuita. CloudTrail

Per una registrazione continua degli eventi negli Account AWS ultimi 90 giorni, crea un trail o un archivio dati sugli [eventi di](#) CloudTrail Lake.

## CloudTrail sentieri

Un percorso consente di CloudTrail inviare file di log a un bucket Amazon S3. Tutti i percorsi creati utilizzando Console di gestione AWS sono multiregionali. È possibile creare un trail per una singola Regione o per più Regioni tramite AWS CLI. La creazione di un percorso multiregionale è consigliata perché registri tutte le attività del tuo Regioni AWS account. Se si crea un trail per una singola Regione, è possibile visualizzare solo gli eventi registrati nella Regione AWS del trail. Per

ulteriori informazioni sui trail, consulta [Creating a trail for your Account AWS](#) e [Creating a trail for an organization](#) nella Guida per l'utente di AWS CloudTrail .

Puoi inviare gratuitamente una copia degli eventi di gestione in corso al tuo bucket Amazon S3 CloudTrail creando un percorso, tuttavia sono previsti costi di archiviazione di Amazon S3. Per ulteriori informazioni sui CloudTrail prezzi, consulta Prezzi. [AWS CloudTrail](#) Per informazioni sui prezzi di Amazon S3, consulta [Prezzi di Amazon S3](#).

## CloudTrail Archivi dati per eventi sul lago

CloudTrail Lake ti consente di eseguire SQL-based query sui tuoi eventi. CloudTrail Lake converte gli eventi esistenti in formato JSON basato su righe in formato Apache ORC. <https://orc.apache.org/> ORC è un formato di archiviazione a colonne ottimizzato per il recupero rapido dei dati. Gli eventi vengono aggregati in archivi di dati degli eventi, che sono raccolte di eventi immutabili basate sui criteri selezionati applicando i [selettori di eventi avanzati](#). I selettori applicati a un archivio di dati degli eventi controllano quali eventi persistono e sono disponibili per l'esecuzione della query. Per ulteriori informazioni su CloudTrail Lake, vedere [Lavorare con AWS CloudTrail Lake](#) nella Guida per l'utente. AWS CloudTrail

CloudTrail Gli archivi di dati e le interrogazioni relative agli eventi di Lake comportano dei costi. Quando crei un datastore di eventi, scegli l'[opzione di prezzo](#) da utilizzare per tale datastore. L'opzione di prezzo determina il costo per l'importazione e l'archiviazione degli eventi, nonché il periodo di conservazione predefinito e quello massimo per il datastore di eventi. Per ulteriori informazioni sui CloudTrail prezzi, vedi [AWS CloudTrail Prezzi](#).

## Eventi relativi ai dati Lambda in CloudTrail

Gli [eventi di dati](#) forniscono informazioni sulle operazioni delle risorse eseguite su o in una risorsa (ad esempio, lettura o scrittura su un oggetto Amazon S3). Queste operazioni sono definite anche operazioni del piano dei dati. Gli eventi di dati sono spesso attività che interessano volumi elevati di dati. Per impostazione predefinita, CloudTrail non registra la maggior parte degli eventi relativi ai dati e la cronologia degli CloudTrail eventi non li registra.

Un evento di CloudTrail dati registrato per impostazione predefinita per i servizi supportati è `LambdaESMDisabled`. Per ulteriori informazioni sull'utilizzo di questo evento per risolvere i problemi relativi agli strumenti di mappatura dell'origine degli eventi Lambda, consulta [the section called "Utilizzo CloudTrail per risolvere i problemi relativi alle sorgenti di eventi Lambda disabilitate"](#).

Per gli eventi di dati sono previsti costi aggiuntivi. Per ulteriori informazioni sui CloudTrail prezzi, consulta [AWS CloudTrail Prezzi](#).

Puoi registrare gli eventi relativi ai dati per il tipo di AWS::Lambda::Function risorsa utilizzando la CloudTrail AWS CLI console o le operazioni CloudTrail API. Per ulteriori informazioni su come registrare gli eventi di dati, consulta [Registrazione degli eventi di dati con Console di gestione AWS](#) e [Registrazione degli eventi di dati con AWS Command Line Interface](#) nella Guida per l'utente di AWS CloudTrail.

La tabella seguente elenca il tipo di risorse Lambda per i quali è possibile registrare gli eventi relativi ai dati. La colonna Tipo di evento di dati (console) mostra il valore da scegliere dall'elenco dei tipi di eventi di dati sulla CloudTrail console. La colonna del valore resources.type mostra il resources.type valore da specificare durante la configurazione dei selettori di eventi avanzati utilizzando le API or. AWS CLI CloudTrail La CloudTrail colonna Data API loggate mostra le chiamate API registrate per il tipo di risorsa. CloudTrail

| Tipo di evento di dati (console) | valore resources.type | API di dati a cui è stato effettuato l'accesso CloudTrail |
|----------------------------------|-----------------------|-----------------------------------------------------------|
| Lambda                           | AWS::Lambda::Function | <a href="#">Invoke</a>                                    |

È possibile configurare selettori di eventi avanzati per filtrare in base ai campi eventName, readOnly, e resources.ARN per registrare solo gli eventi che sono importanti per l'utente. L'esempio seguente è la vista JSON di una configurazione di eventi di dati che registra gli eventi solo per una funzione specifica. Per ulteriori informazioni su questi campi, consulta [AdvancedFieldSelector](#) nella Documentazione di riferimento delle API di AWS CloudTrail.

```
[
  {
    "name": "function-invokes",
    "fieldSelectors": [
      {
        "field": "eventCategory",
        "equals": [
          "Data"
        ]
      },
      {
        "field": "resources.type",
        "equals": [
          "AWS::Lambda::Function"
        ]
      }
    ]
  }
]
```



```

    ]
  },
  {
    "field": "resources.ARN",
    "equals": [
      "arn:aws:lambda:us-east-1:111122223333:function:hello-world"
    ]
  }
]
}
]

```

## Eventi di gestione Lambda in CloudTrail

[Gli eventi di gestione](#) forniscono informazioni sulle operazioni di gestione eseguite sulle risorse presenti nel tuo Account AWS. Queste operazioni sono definite anche operazioni del piano di controllo (control-plane). Per impostazione predefinita, CloudTrail registra gli eventi di gestione.

Lambda supporta la registrazione delle seguenti azioni come eventi di gestione nei CloudTrail file di registro.

### Note

Nel file di CloudTrail registro, eventName possono includere informazioni sulla data e sulla versione, ma si riferisce comunque alla stessa azione API pubblica. Ad esempio, l'operazione GetFunction potrebbe apparire come GetFunction20150331v2. L'elenco seguente specifica quando il nome dell'evento è diverso dal nome dell'operazione API.

- [AddLayerVersionPermission](#)
- [AddPermission](#)(nome dell'evento:AddPermission20150331v2)
- [CreateAlias](#)(nome dell'evento:CreateAlias20150331)
- [CreateEventSourceMapping](#)(nome dell'evento:CreateEventSourceMapping20150331)
- [CreateFunction](#)(nome dell'evento:CreateFunction20150331)

(I parametri Environment e ZipFile sono omessi dai log CloudTrail per CreateFunction.)

- [CreateFunctionUrlConfig](#)
- [DeleteAlias](#)(nome dell'evento>DeleteAlias20150331)
- [DeleteCodeSigningConfig](#)

- [DeleteEventSourceMapping](#)(nome dell'evento:DeleteEventSourceMapping20150331)
- [DeleteFunction](#)(nome dell'evento:DeleteFunction20150331)
- [DeleteFunctionConcurrency](#)(nome dell'evento:DeleteFunctionConcurrency20171031)
- [DeleteFunctionUrlConfig](#)
- [DeleteProvisionedConcurrencyConfig](#)
- [GetAlias](#)(nome dell'evento:GetAlias20150331)
- [GetEventSourceMapping](#)
- [GetFunction](#)
- [GetFunctionUrlConfig](#)
- [GetFunctionConfiguration](#)
- [GetLayerVersionPolicy](#)
- [GetPolicy](#)
- [ListEventSourceMappings](#)
- [ListFunctions](#)
- [ListFunctionUrlConfigs](#)
- [PublishLayerVersion](#)(nome dell'evento:PublishLayerVersion20181031)

(Il ZipFile parametro viene omissso dai CloudTrail registri diPublishLayerVersion.)

- [PublishVersion](#)(nome dell'evento:PublishVersion20150331)
- [PutFunctionConcurrency](#)(nome dell'evento:PutFunctionConcurrency20171031)
- [PutFunctionCodeSigningConfig](#)
- [PutFunctionEventInvokeConfig](#)
- [PutProvisionedConcurrencyConfig](#)
- [PutRuntimeManagementConfig](#)
- [RemovePermission](#)(nome dell'evento:RemovePermission20150331v2)
- [TagResource](#)(nome dell'evento:TagResource20170331v2)
- [UntagResource](#)(nome dell'evento:UntagResource20170331v2)
- [UpdateAlias](#)(nome dell'evento:UpdateAlias20150331)
- [UpdateCodeSigningConfig](#)
- [UpdateEventSourceMapping](#)(nome dell'evento:UpdateEventSourceMapping20150331)
- [UpdateFunctionCode](#)(nome dell'evento:UpdateFunctionCode20150331v2)

(Il `ZipFile` parametro viene omissso dai CloudTrail registri di `UpdateFunctionCode`.)

- [UpdateFunctionConfiguration](#)(nome dell'evento: `UpdateFunctionConfiguration20150331v2`)

(Il `Environment` parametro viene omissso dai CloudTrail registri di `UpdateFunctionConfiguration`.)

- [UpdateFunctionEventInvokeConfig](#)
- [UpdateFunctionUrlConfig](#)

## Utilizzo CloudTrail per risolvere i problemi relativi alle sorgenti di eventi Lambda disabilitate

Quando si modifica lo stato della mappatura della sorgente di un evento utilizzando l'azione [UpdateEventSourceMapping](#) API, la chiamata API viene registrata come evento di gestione. CloudTrail Gli strumenti di mappatura dell'origine degli eventi possono anche passare direttamente allo stato `Disabled` a causa di errori.

Per i seguenti servizi, Lambda pubblica l'evento di `LambdaESMDisabled` dati CloudTrail quando la sorgente dell'evento passa allo stato `Disabilitato`:

- Amazon Simple Queue Service (Amazon SQS)
- Amazon DynamoDB
- Amazon Kinesis

Lambda non supporta questo evento per alcun altro tipo di strumento di mappatura dell'origine degli eventi.

Per ricevere avvisi quando le mappature delle sorgenti di eventi per i servizi supportati passano allo `Disabled` stato, configura un allarme in Amazon utilizzando l'evento. CloudWatch `LambdaESMDisabled` CloudTrail Per saperne di più sulla configurazione di un CloudWatch allarme, consulta [Creazione di CloudWatch allarmi per CloudTrail eventi: esempi](#).

L'entità `serviceEventDetails` nel messaggio dell'evento `LambdaESMDisabled` contiene uno dei seguenti codici di errore.

### **RESOURCE\_NOT\_FOUND**

La risorsa specificata nella richiesta non esiste.

## FUNCTION\_NOT\_FOUND

La funzione associata all'origine eventi non esiste.

## REGION\_NAME\_NOT\_VALID

Il nome di una regione fornito all'origine o alla funzione evento non è valido.

## AUTHORIZATION\_ERROR

Le autorizzazioni non sono state impostate o non sono configurate correttamente.

## FUNCTION\_IN\_FAILED\_STATE

Il codice della funzione non viene compilato, ha rilevato un'eccezione irrecuperabile o si è verificata una distribuzione non valida.

## Esempi di eventi Lambda

Un evento rappresenta una singola richiesta da qualsiasi fonte e include informazioni sull'operazione API richiesta, la data e l'ora dell'operazione, i parametri della richiesta e così via. CloudTrail i file di registro non sono una traccia ordinata dello stack delle chiamate API pubbliche, quindi gli eventi non vengono visualizzati in un ordine specifico.

L'esempio seguente mostra le voci di CloudTrail registro per le DeleteFunction azioni GetFunction and.

### Note

La voce eventName potrebbe includere informazioni sulla data e sulla versione, ad esempio "GetFunction20150331", ma si riferisce comunque alla stessa API pubblica.

```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "A1B2C3D4E5F6G7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
```

```

    "userName": "myUserName"
  },
  "eventTime": "2015-03-18T19:03:36Z",
  "eventSource": "lambda.amazonaws.com",
  "eventName": "GetFunction",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "Python-httpplib2/0.8 (gzip)",
  "errorCode": "AccessDenied",
  "errorMessage": "User: arn:aws:iam::111122223333:user/myUserName is not
authorized to perform: lambda:GetFunction on resource: arn:aws:lambda:us-
west-2:111122223333:function:other-acct-function",
  "requestParameters": null,
  "responseElements": null,
  "requestID": "7aebcd0f-cda1-11e4-aaa2-e356da31e4ff",
  "eventID": "e92a3e85-8ecd-4d23-8074-843aabfe89bf",
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
},
{
  "eventVersion": "1.03",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "A1B2C3D4E5F6G7EXAMPLE",
    "arn": "arn:aws:iam::111122223333:user/myUserName",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "myUserName"
  },
  "eventTime": "2015-03-18T19:04:42Z",
  "eventSource": "lambda.amazonaws.com",
  "eventName": "DeleteFunction20150331",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "Python-httpplib2/0.8 (gzip)",
  "requestParameters": {
    "functionName": "basic-node-task"
  },
  "responseElements": null,
  "requestID": "a2198ecc-cda1-11e4-aaa2-e356da31e4ff",
  "eventID": "20b84ce5-730f-482e-b2b2-e8fcc87ceb22",
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}

```

```
]
}
```

Per informazioni sui contenuti dei CloudTrail record, vedere i contenuti dei [CloudTrail record](#) nella Guida AWS CloudTrail per l'utente.

## Visualizza le invocazioni delle funzioni Lambda utilizzando AWS X-Ray

Puoi utilizzarlo AWS X-Ray per visualizzare i componenti della tua applicazione, identificare i colli di bottiglia nelle prestazioni e risolvere le richieste che hanno provocato un errore. Le tue funzioni Lambda inviano dati di traccia a ed elaborano i dati per X-Ray generare una mappa del X-Ray servizio e riepiloghi delle tracce ricercabili.

Lambda supporta due modalità di tracciamento per: e. X-Ray Active PassThrough Con il Active tracing, Lambda crea automaticamente segmenti di traccia per le invocazioni di funzioni e li invia a. X-Ray PassThroughmode, d'altra parte, propaga semplicemente il contesto di tracciamento ai servizi a valle.

Se hai abilitato la Active traccia per la tua funzione, Lambda invia automaticamente le tracce per le richieste campionate. X-Ray In genere, un servizio upstream, come Amazon API Gateway o un'applicazione ospitata su Amazon EC2 strumentata con l' X-Ray SDK, decide se le richieste in arrivo devono essere tracciate, quindi aggiunge la decisione di campionamento come intestazione di tracciamento. Lambda utilizza quell'intestazione per decidere se inviare tracce o meno. Le tracce dei produttori di messaggi upstream, come Amazon SQS, vengono automaticamente collegate alle tracce delle funzioni Lambda downstream, creando una visione completa dell'intera applicazione. Per ulteriori informazioni, consulta [Tracciamento delle applicazioni guidate dagli eventi](#) nella Guida per gli sviluppatori di AWS X-Ray .

### Note

X-Ray la traccia non è attualmente supportata per le funzioni Lambda con Amazon Managed Streaming for Apache Kafka (Amazon MSK), Apache Kafka autogestito, Amazon MQ con ActiveMQ e RabbitMQ o le mappature delle sorgenti di eventi Amazon DocumentDB.

Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).

4. In Strumenti di monitoraggio aggiuntivi scegli Modifica.
5. In AWS X-Ray Application Signals e, seleziona Abilita per le tracce CloudWatch del servizio Lambda.
6. Selezionare Salva.

La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al [ruolo di esecuzione](#) della funzione. Altrimenti, aggiungi la [AWSXRayDaemonWriteAccess](#) policy al ruolo di esecuzione.

X-Ray non traccia tutte le richieste all'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento sia efficiente, fornendo comunque un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non puoi configurare la frequenza di X-Ray campionamento per le tue funzioni.

## Comprensione delle tracce X-Ray

In X-Ray, una traccia registra le informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda.

Il segmento registrato per il servizio Lambda, `AWS::Lambda`, copre tutti i passaggi necessari per preparare l'ambiente di esecuzione Lambda. Ciò include la pianificazione di MicroVM, la creazione o lo sblocco di un ambiente di esecuzione con le risorse configurate, nonché il download del codice per la funzione e di tutti i livelli.



Il segmento `AWS::Lambda::Function` è destinato al lavoro svolto dalla funzione.

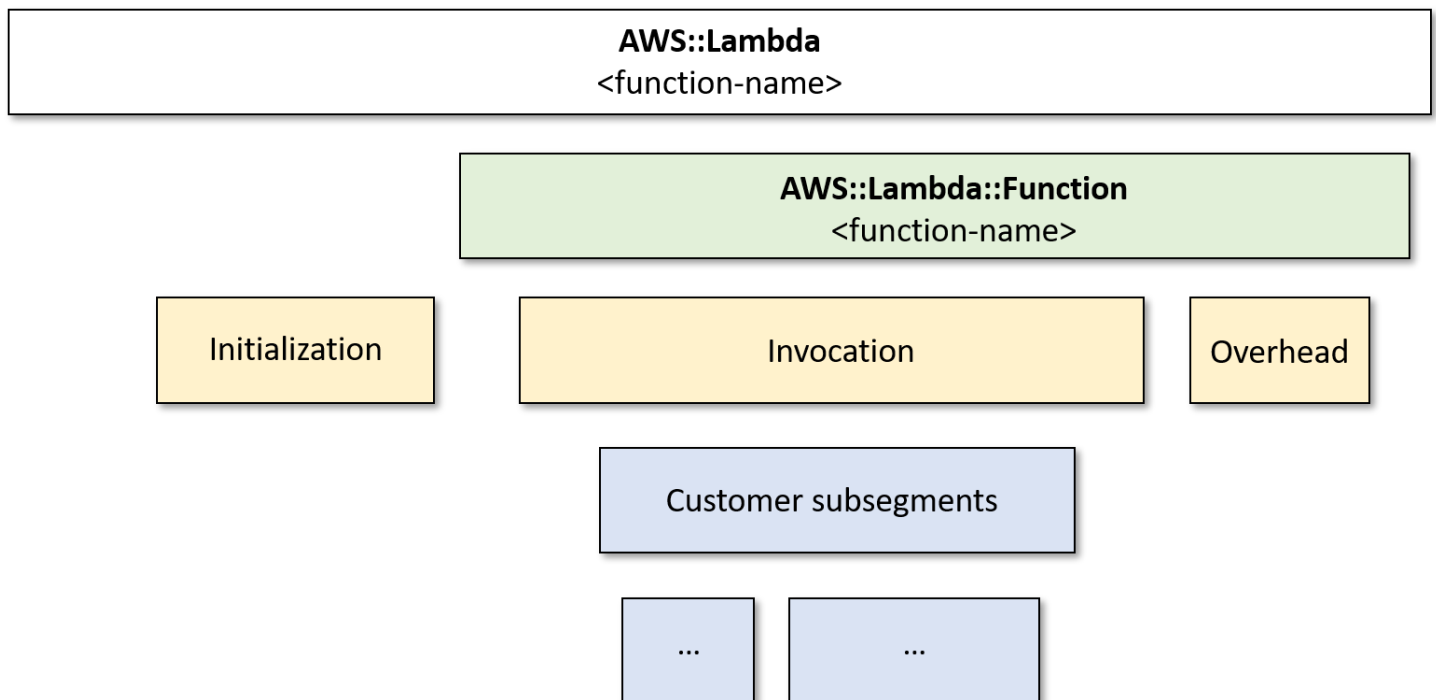
**Note**

AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. Questa modifica influisce sui sottosegmenti del segmento della funzione. I paragrafi seguenti descrivono sia il vecchio che il nuovo formato per questi sottosegmenti.

Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regioni, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia nel nuovo formato.

### Old-style AWS X-Ray Struttura del segmento Lambda

La X-Ray struttura vecchio stile del `AWS::Lambda` segmento è la seguente:



In questo formato, il segmento della funzione ha sottosegmenti per **Initialization**, **Invocation** e **Overhead**. Solo per [SnapStart](#), c'è anche un sottosegmento **Restore** (non mostrato in questo diagramma).

Il sottosegmento `Initialization` rappresenta la fase iniziale del ciclo di vita dell'ambiente di esecuzione Lambda. In questa fase, Lambda inizializza le estensioni, inizializza il runtime ed esegue il codice di inizializzazione della funzione.

Il sottosegmento `Invocation` rappresenta la fase di invocazione in cui Lambda richiama il gestore di funzioni. Questo inizia con la registrazione del runtime e dell'estensione e termina quando il runtime è pronto per inviare la risposta.

( SnapStart Solo Lambda) Il `Restore` sottosegmento mostra il tempo impiegato da Lambda per ripristinare un'istanza, caricare il runtime ed eseguire eventuali hook di runtime dopo il ripristino. [???](#) Il processo di ripristino degli snapshot può includere il tempo dedicato ad attività esterne alla MicroVM. Questa volta è riportato nel segmento secondario `Restore`. Non ti viene addebitato il tempo trascorso all'esterno della MicroVM per ripristinare un'istanza.

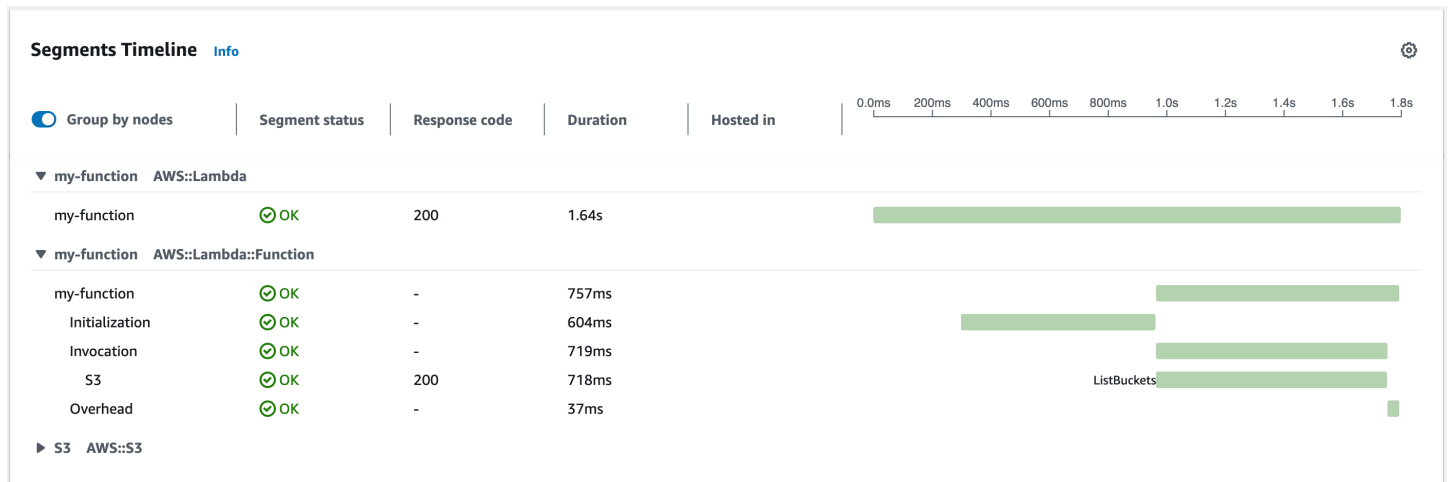
Il sottosegmento `Overhead` rappresenta la fase che si verifica tra il momento in cui il runtime invia la risposta e il segnale per la successiva invocazione. Durante questo periodo, il runtime termina tutte le attività correlate a un'invocazione e si prepara a congelare la sandbox.

 Important

Puoi utilizzare l' X-Ray SDK per estendere il `Invocation` sottosegmento con sottosegmenti aggiuntivi per chiamate, annotazioni e metadati a valle. Non è possibile accedere direttamente al segmento di funzione o registrare la parola eseguita al di fuori dell'ambito di chiamata del gestore.

Per ulteriori informazioni sulle fasi dell'ambiente di esecuzione Lambda, consulta [the section called “Ambiente di esecuzione”](#).

Un esempio di traccia che utilizza la struttura vecchio stile è mostrato nel diagramma seguente X-Ray .



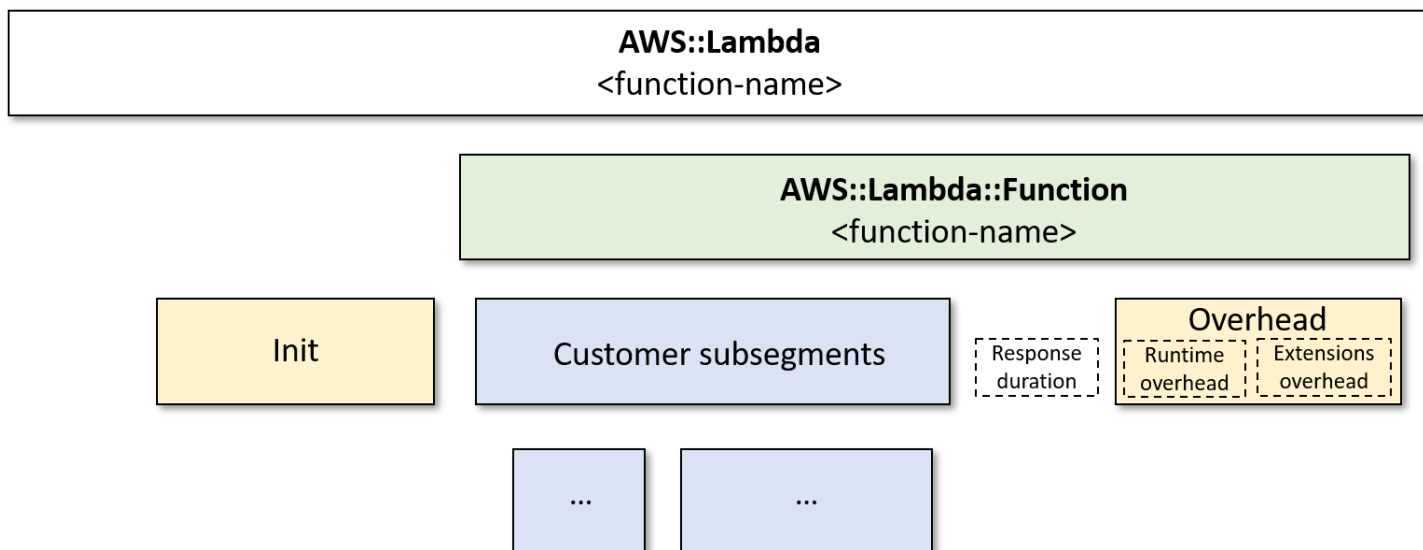
Nota i due segmenti dell'esempio. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.

### Note

Di tanto in tanto, potreste notare un ampio divario tra le fasi di inizializzazione della funzione e di invocazione nelle tracce. X-Ray Per le funzioni che utilizzano la [simultaneità fornita](#), ciò è dovuto al fatto che Lambda inizializza le istanze della funzione con largo anticipo rispetto all'invocazione. Per le funzioni che utilizzano la concorrenza [non riservata \(su richiesta\)](#), Lambda potrebbe inizializzare in modo proattivo un'istanza di funzione, anche se non viene richiamata. Visivamente, entrambi questi casi si presentano come un intervallo di tempo tra le fasi di inizializzazione e invocazione.

### New-style AWS X-Ray Struttura del segmento Lambda

La X-Ray struttura di nuovo stile per il `AWS::Lambda` segmento è la seguente:

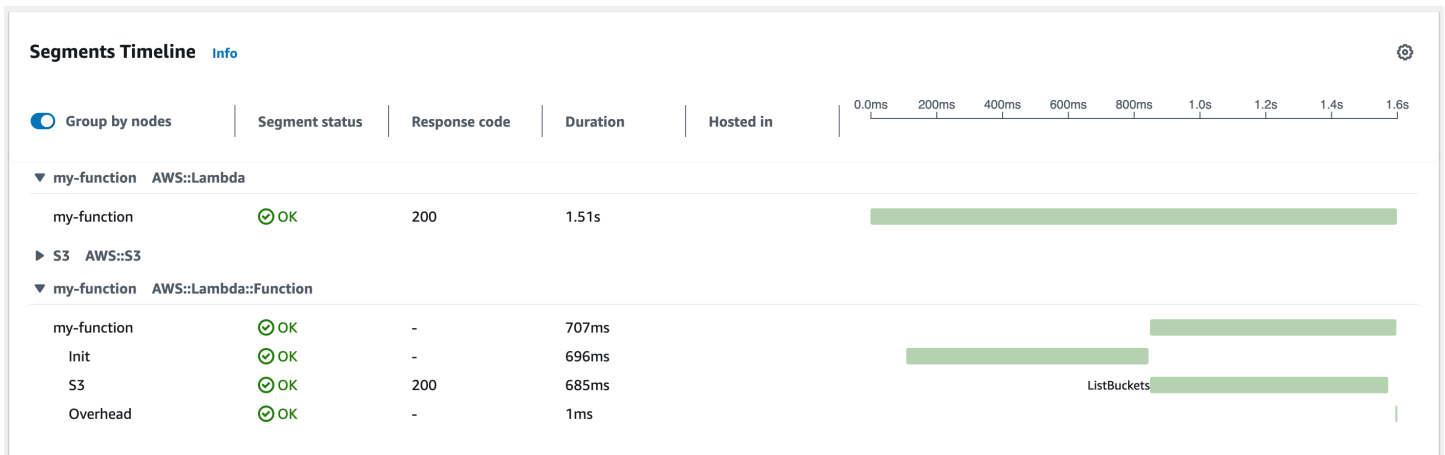


In questo nuovo formato, il sottosegmento `Init` rappresenta la fase iniziale del ciclo di vita dell'ambiente di esecuzione Lambda.

Nel nuovo formato non è presente alcun segmento di invocazione. I sottosegmenti dei clienti sono invece collegati direttamente al segmento `AWS::Lambda::Function`. Questo segmento contiene i seguenti parametri come annotazioni:

- `aws.responseLatency`: il tempo impiegato per l'esecuzione della funzione
- `aws.responseDuration`: il tempo impiegato per trasferire la risposta al cliente
- `aws.runtimeOverhead`: la quantità di tempo supplementare necessaria al completamento del runtime
- `aws.extensionOverhead`: la quantità di tempo supplementare necessaria al completamento delle estensioni

Un esempio di traccia che utilizza la X-Ray struttura di nuovo stile è mostrato nel diagramma seguente.



Nota i due segmenti dell'esempio. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.

Vedere i seguenti argomenti per un'introduzione specifica della lingua all'analisi in Lambda:

- [Codice di strumentazione in Node.js AWS Lambda](#)
- [Strumentazione del codice Python in AWS Lambda](#)
- [Strumentazione del codice Ruby in AWS Lambda](#)
- [Strumentazione del codice Java in AWS Lambda](#)
- [Strumentazione del codice Go AWS Lambda](#)
- [Strumentazione del codice C# in AWS Lambda](#)

Per un elenco completo dei servizi che supportano la strumentazione attiva, consulta [Servizi Servizi AWS](#) supportati nella Guida per gli sviluppatori di AWS X-Ray .

## Comportamento di tracciamento predefinito in Lambda

Se non hai attivato il Active tracciamento, Lambda passa automaticamente alla modalità di tracciamento. `PassThrough`

In `PassThrough` modalità, Lambda inoltra l'intestazione di X-Ray tracciamento ai servizi a valle, ma non invia le tracce automaticamente. Ciò è vero anche se l'intestazione di tracciamento contiene la decisione di campionare la richiesta. Se il servizio upstream non fornisce un'intestazione di X-Ray tracciamento, Lambda genera un'intestazione e decide di non effettuare il campionamento. Tuttavia, puoi inviare le tue tracce richiamando le librerie di tracciamento dal tuo codice funzione.

 Note

In precedenza, Lambda inviava le tracce automaticamente quando i servizi upstream, come Amazon API Gateway, aggiungevano un'intestazione di tracciamento. Non inviando tracce automaticamente, Lambda ti dà il controllo necessario per tracciare le funzioni che sono importanti per te. Se la tua soluzione dipende da questo comportamento di tracciamento passivo, passa al **Active** tracciamento.

## Autorizzazioni del ruolo di esecuzione

Lambda necessita delle seguenti autorizzazioni per inviare i dati di traccia a X-Ray. Aggiungerle al [ruolo di esecuzione](#) della funzione.

- [Radiografia: PutTraceSegments](#)
- [radiografia: PutTelemetryRecords](#)

Queste autorizzazioni sono incluse nella politica [AWSXRayDaemonWriteAccess](#) gestita.

## Abilitazione del **tracciamento** attivo con l'API Lambda

Per gestire la configurazione del tracciamento con AWS CLI or AWS SDK, utilizza le seguenti operazioni API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

Il seguente AWS CLI comando di esempio abilita la traccia attiva su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

## Abilitazione della traccia attiva **con** CloudFormation

Per attivare la traccia su una `AWS::Lambda::Function` risorsa in un CloudFormation modello, utilizzate la `TracingConfig` proprietà.

Example [function-inline.yml](#) — Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Per una `AWS::Serverless::Function` risorsa AWS Serverless Application Model (AWS SAM), usa la `Tracing` proprietà.

Example [template.yml](#) — Configurazione di tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

# Monitora le prestazioni delle funzioni con Amazon CloudWatch Lambda Insights

Amazon CloudWatch Lambda Insights raccoglie e aggrega le metriche e i log delle prestazioni di runtime delle funzioni Lambda per le tue applicazioni serverless. In questa pagina viene descritto come abilitare e utilizzare Lambda Insights per eseguire la diagnosi dei problemi relativi alle funzioni Lambda.

## Sections

- [Come Lambda Insights monitora le applicazioni serverless](#)
- [Prezzi](#)
- [Runtime supportati](#)
- [Attivazione di Lambda Insights nella console Lambda](#)
- [Attivazione di Lambda Insights a livello di programmazione](#)
- [Uso del pannello di controllo di Lambda Insights](#)
- [Esempio di flusso di lavoro per rilevare anomalie delle funzioni](#)
- [Esempio di flusso di lavoro utilizzando query per risolvere i problemi di una funzione](#)
- [Fasi successive](#)

## Come Lambda Insights monitora le applicazioni serverless

CloudWatch Lambda Insights è una soluzione di monitoraggio e risoluzione dei problemi per le applicazioni serverless in esecuzione su AWS Lambda. La soluzione raccoglie, aggrega e riassume le metriche a livello di sistema, tra cui il tempo della CPU, la memoria, l'utilizzo del disco e della rete. Vengono inoltre raccolte, aggregate e riepilogate informazioni diagnostiche quali avvii a freddo e arresti del worker Lambda per aiutare a isolare i problemi con le funzioni Lambda e risolverli rapidamente.

Lambda Insights utilizza una nuova [estensione CloudWatch Lambda Insights](#), fornita come layer Lambda. [???](#) Quando abiliti questa estensione su una funzione Lambda per un runtime supportato, raccoglie metriche a livello di sistema ed emette un singolo evento di registro delle prestazioni per ogni chiamata di quella funzione Lambda. CloudWatch utilizza la formattazione delle metriche incorporata per estrarre le metriche dagli eventi del registro. Per ulteriori informazioni, consulta [Utilizzo delle estensioni. AWS Lambda](#)



Il livello Lambda Insights estende `CreateLogStream` e `PutLogEvents` per il gruppo di log `/aws/lambda-insights/`.

## Prezzi

Quando abiliti Lambda Insights per la tua funzione Lambda, Lambda Insights riporta 8 metriche per funzione e ogni chiamata di funzione invia circa 1 KB di dati di registro a CloudWatch. Paghi solo in base ai parametri e ai registri riportati per la tua funzione da Lambda Insights. Non sono previste tariffe minime né policy di utilizzo del servizio obbligatorie. Non si paga per Lambda Insights se la funzione non viene richiamata. Per un esempio di prezzo, consulta i prezzi di Amazon. [CloudWatch](#)

## Runtime supportati

È possibile utilizzare Lambda Insights con qualsiasi runtime che supporta le [estensioni Lambda](#).

## Attivazione di Lambda Insights nella console Lambda

È possibile abilitare il monitoraggio Lambda Insights avanzato su funzioni Lambda nuove ed esistenti. Quando abiliti Lambda Insights su una funzione nella console Lambda per un runtime supportato, Lambda aggiunge l'[estensione Lambda Insights](#) come livello alla tua funzione e verifica o tenta di collegare la [CloudWatchLambdaInsightsExecutionRolePolicy](#) policy al ruolo di esecuzione della funzione. <https://docs.aws.amazon.com/lambda/latest/dg/lambda-intro-execution-role.html>

Per attivare Lambda Insights nella console Lambda

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere la funzione.
3. Scegli la scheda Configurazione.
4. Dal menu a sinistra, scegli Strumenti di monitoraggio e operazioni.
5. Nel riquadro Strumenti di monitoraggio aggiuntivi scegli Modifica.
6. In CloudWatch Lambda Insights, attivare Enhanced monitoring (Monitoraggio avanzato).
7. Scegli Save (Salva).

## Attivazione di Lambda Insights a livello di programmazione

Puoi anche abilitare Lambda Insights utilizzando la CLI AWS Command Line Interface (AWS CLI), AWS Serverless Application Model (SAM) o CloudFormation Kit di sviluppo per il cloud AWS (AWS

CDK) Quando abiliti Lambda Insights a livello di codice su una funzione per un runtime supportato, CloudWatch allega la [CloudWatchLambdaInsightsExecutionRolePolicy](#) policy al ruolo di esecuzione della funzione. <https://docs.aws.amazon.com/lambda/latest/dg/lambda-intro-execution-role.html>

Per ulteriori informazioni, consulta [Getting started with Lambda Insights](#) nella Amazon User Guide. CloudWatch

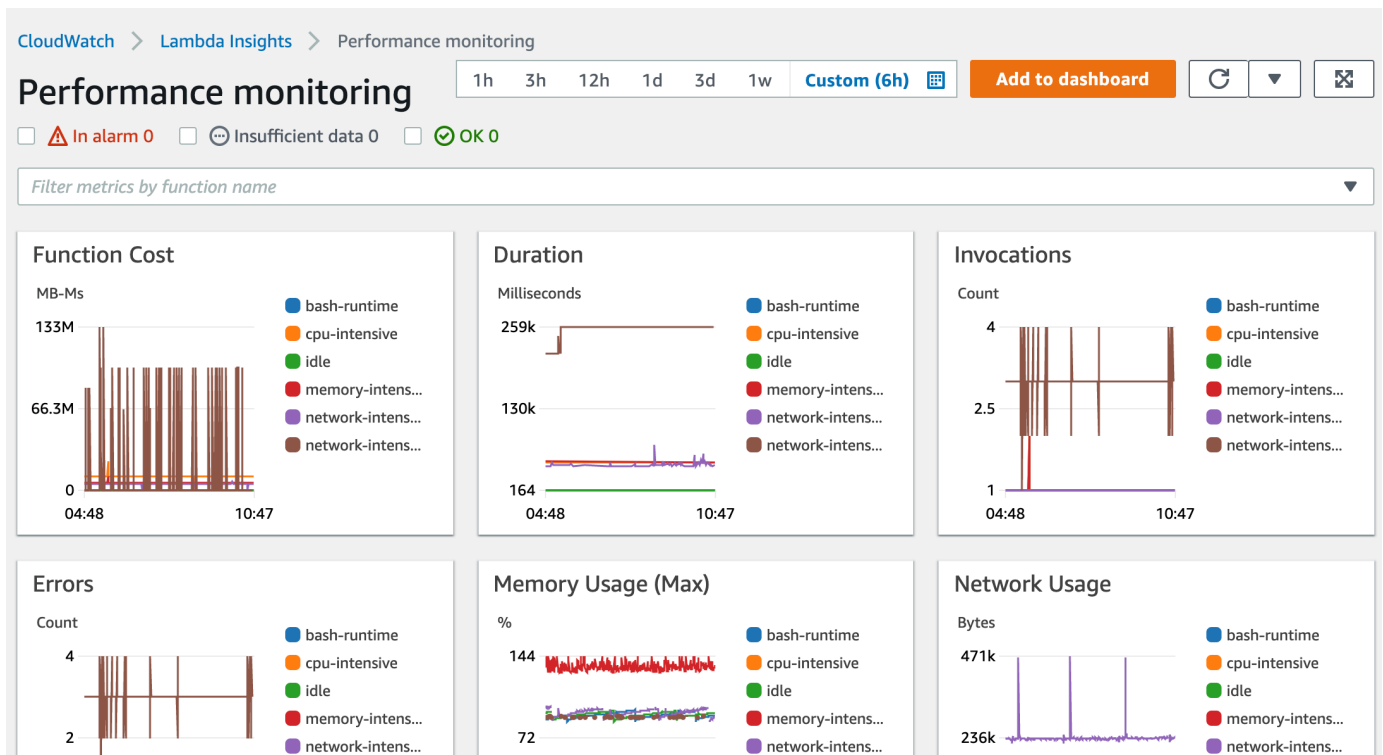
## Uso del pannello di controllo di Lambda Insights

La dashboard di Lambda Insights ha due visualizzazioni nella CloudWatch console: la panoramica multifunzione e la vista a funzione singola. La panoramica multifunzione aggrega le metriche di runtime per le funzioni Lambda nell'account corrente e nella regione. AWS La vista a funzione singola mostra i parametri di runtime disponibili per una singola funzione Lambda.

Puoi utilizzare la panoramica multifunzione della dashboard di Lambda Insights nella CloudWatch console per identificare le funzioni Lambda sovra e sottoutilizzate. Puoi utilizzare la visualizzazione a funzione singola del dashboard di Lambda Insights nella console per risolvere le singole richieste. CloudWatch

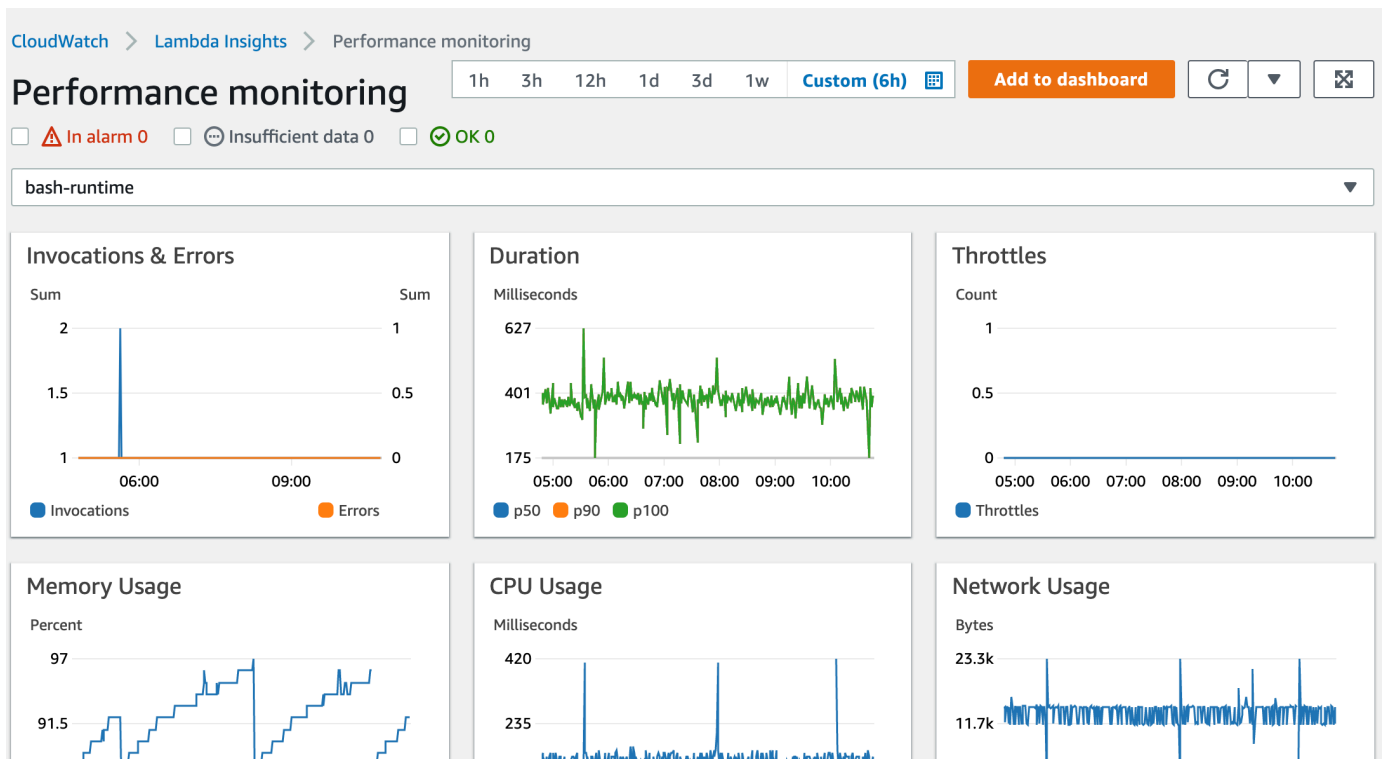
Per visualizzare i parametri di runtime per tutte le funzioni

1. Apri la [Multi-function](#) pagina nella console. CloudWatch
2. Scegli tra gli intervalli di tempo predefiniti oppure scegli un intervallo di tempo personalizzato.
3. (Facoltativo) Scegli **Aggiungi** alla dashboard per aggiungere i widget alla CloudWatch dashboard.



Per visualizzare le metriche di runtime di una singola funzione

1. Apri la [Single-function](#) pagina nella CloudWatch console.
2. Scegli tra gli intervalli di tempo predefiniti oppure scegli un intervallo di tempo personalizzato.
3. (Facoltativo) Scegli **Aggiungi** alla dashboard per aggiungere i widget alla CloudWatch dashboard.



Per ulteriori informazioni, consulta [Creazione e utilizzo dei widget nelle CloudWatch dashboard](#).

## Esempio di flusso di lavoro per rilevare anomalie delle funzioni

È possibile utilizzare la panoramica multifunzione sul pannello di controllo Lambda Insights per identificare e rilevare anomalie della memoria di calcolo con la funzione. Ad esempio, se la panoramica multifunzione indica che una funzione utilizza una grande quantità di memoria, è possibile visualizzare metriche dettagliate sull'utilizzo della memoria nel riquadro Uso memoria. Puoi quindi accedere al pannello di controllo Parametri per abilitare il rilevamento delle anomalie o creare un allarme.

Per abilitare il rilevamento delle anomalie per una funzione

1. Apri la [Multi-function](#) pagina nella console. CloudWatch
2. In Riepilogo funzione, scegliere il nome della funzione.

La vista a funzione singola si apre con le metriche di runtime della funzione.

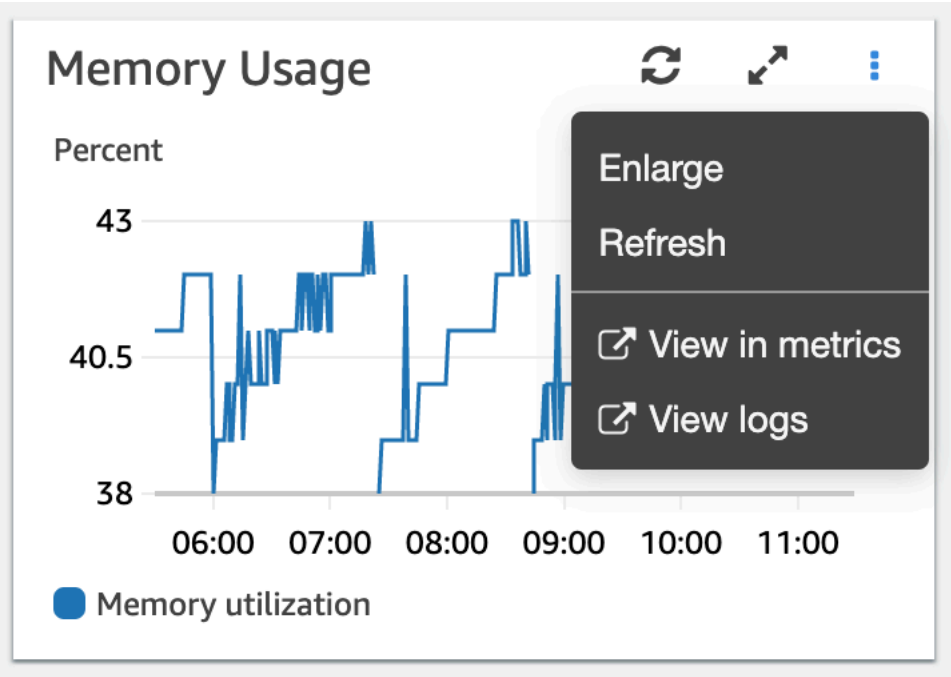
Function summary (6)

Actions

< 1 > ⚙

| <input type="checkbox"/> | Function name ▲                       | Invocations ▼ | CPU time ▼  | Network IO ▼ | Max. memory ▼              | Cold starts ▼ |
|--------------------------|---------------------------------------|---------------|-------------|--------------|----------------------------|---------------|
| <input type="checkbox"/> | <a href="#">bash-runtime</a>          | 360           | 132.9167ms  | 4770 kB      | <div><div></div></div> 97% | 3             |
| <input type="checkbox"/> | <a href="#">cpu-intensive</a>         | 359           | 6714.2897ms | 4780 kB      | <div><div></div></div> 43% | 4             |
| <input type="checkbox"/> | <a href="#">idle</a>                  | 359           | 120.2507ms  | 4746 kB      | <div><div></div></div> 96% | 3             |
| <input type="checkbox"/> | <a href="#">memory-intensive</a>      | 358           | 2385.9497ms | 4794 kB      | <div><div></div></div> 44% | 4             |
| <input type="checkbox"/> | <a href="#">network-intensive</a>     | 359           | 781.0585ms  | 82008 kB     | <div><div></div></div> 99% | 3             |
| <input type="checkbox"/> | <a href="#">network-intensive-vpc</a> | 43            | 2730.6977ms | 95 kB        | <div><div></div></div> 91% | 43            |

3. Nel riquadro Uso memoria scegliere i tre punti verticali, quindi scegliere Visualizza in metriche per aprire il pannello di controllo Parametri .



4. Nella scheda Graphed metrics (Parametri grafici), nella colonna Actions (Operazioni), scegli la prima icona per abilitare il rilevamento delle anomalie per la funzione.

All metrics

Graphed metrics (6)

Graph options

Source

Math expression

Dynamic labels

Statistic: Maximum

Period: 1 Minute

Remove all

| <input checked="" type="checkbox"/> |  | Label            | Details                                          | Statistic | Period   | Y Axis | Actions |
|-------------------------------------|--|------------------|--------------------------------------------------|-----------|----------|--------|---------|
| <input checked="" type="checkbox"/> |  | bash-runtime     | LambdaInsights * memory_utilization * functio... | Maximum   | 1 Minute |        |         |
| <input checked="" type="checkbox"/> |  | cpu-intensive    | LambdaInsights * memory_utilization * functio... | Maximum   | 1 Minute |        |         |
| <input checked="" type="checkbox"/> |  | idle             | LambdaInsights * memory_utilization * functio... | Maximum   | 1 Minute |        |         |
| <input checked="" type="checkbox"/> |  | memory-intensive | LambdaInsights * memory_utilization * functio... | Maximum   | 1 Minute |        |         |

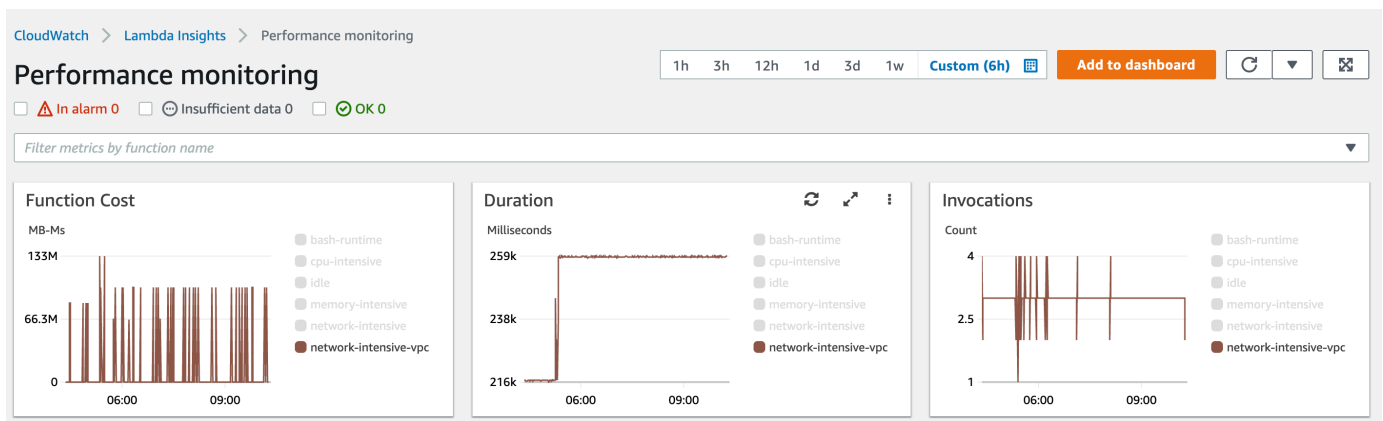
Per ulteriori informazioni, consulta [Utilizzo del rilevamento CloudWatch](#) delle anomalie.

## Esempio di flusso di lavoro utilizzando query per risolvere i problemi di una funzione

È possibile utilizzare la visualizzazione a funzione singola nel pannello di controllo Lambda Insights per identificare la causa principale di un picco nella durata della funzione. Ad esempio, se la panoramica multifunzione indica un notevole aumento della durata della funzione, è possibile sospendere o scegliere ciascuna funzione nel riquadro Durata per determinare quale funzione sta causando l'aumento. È quindi possibile accedere alla visualizzazione a funzione singola ed esaminare i registri applicazioni per determinare la causa principale.

Per eseguire query su una funzione

1. Apri la [Multi-function](#) pagina nella CloudWatch console.
2. Nel riquadro Durata scegliere la funzione per filtrare le metriche della durata.



3. Apri la [Single-function](#) pagina.
4. Scegli l'elenco a discesa Filtra metriche per nome funzione, quindi scegli la funzione.
5. Per visualizzare i 1000 registri applicazioni più recenti, scegliere la scheda Registri applicazioni.

- Esaminare il timestamp e il messaggio per identificare la richiesta di chiamata per cui si desidera risolvere i problemi.

Most recent 1000 application logs (1000)

View logs

< 1 2 3 4 5 6 7 ... 15 > ⚙

| Timestamp ▼                | Message ▼                                 |
|----------------------------|-------------------------------------------|
| 2020-09-30T16:24:36.121-06 | 0 0 0 0 0 0 0 --:--:-- 0:03:06 --:--:-- 0 |
| 2020-09-30T16:24:34.917-06 | 0 0 0 0 0 0 0 --:--:-- 0:04:15 --:--:-- 0 |
| 2020-09-30T16:24:34.120-06 | 0 0 0 0 0 0 0 --:--:-- 0:03:04 --:--:-- 0 |
| 2020-09-30T16:24:33.033-06 | 0 0 0 0 0 0 0 --:--:-- 0:01:26 --:--:-- 0 |

- Per visualizzare le 1000 invocazioni più recenti, scegliere la scheda Invocazioni .
- Selezionare il timestamp o il messaggio per la richiesta di invocazione per cui si desidera risolvere i problemi.

Most recent 1000 invocations (1/45)

View logs

< 1 > ⚙

| <input type="checkbox"/>            | Timestamp ▼                     | Request ID ▼       | Trace ▼ | Memory % ▼                 | Network IO ▼ | CPU time ▼ | Cold start ▼ |
|-------------------------------------|---------------------------------|--------------------|---------|----------------------------|--------------|------------|--------------|
| <input checked="" type="checkbox"/> | 2020-09-30 16:22:34 (UTC-06:00) | 247e6369-3a2b-...  | -       | <div><div></div></div> 91% | 2 kB         | 2550ms     | Yes          |
| <input type="checkbox"/>            | 2020-09-30 16:13:39 (UTC-06:00) | 311fb438-fa9d-4... | -       | <div><div></div></div> 90% | 2 kB         | 2340ms     | Yes          |

- Scegliere l'elenco a discesa Visualizza registri quindi scegliere Visualizza registri prestazioni.

Viene visualizzata una query generata automaticamente per la funzione nel pannello di controllo Logs Insights (Analisi registri) .

- Scegliere Esegui query per generare un messaggio Log per la richiesta di chiamata.

The screenshot shows the AWS Lambda Insights console. At the top, there's a search bar with "Select log group(s)" and a date range from 2020-09-30 (10:35:41) to 2020-09-30 (16:35:41). Below this, a query is entered in a text area: `fields @timestamp, @message | filter function_name = "network-intensive-vpc" | filter request_id = "247e6369-3a2b-4ccf-9e95-fb80c6ba711f" | sort @timestamp desc`. There are buttons for "Run query", "Save", and "History".

Below the query area, there are tabs for "Logs" and "Visualization". The "Logs" tab is selected. It shows a histogram with a single bar at 04 PM. The text above the histogram says "Showing 1 of 1 records matched" and "1,856 records (2.0 MB) scanned in 4.0s @ 467 records/s (521.7 kB/s)". There's a "Hide histogram" link.

Below the histogram, there's a table with columns: #, @timestamp, and @message. The first row shows a log entry with a timestamp of 2020-09-30T16:22:34.000Z and a message containing CPU and memory statistics.

| # | @timestamp               | @message                                                                                                |
|---|--------------------------|---------------------------------------------------------------------------------------------------------|
| 1 | 2020-09-30T16:22:34.000Z | {"cpu_system_time":1520,"shutdown":1,"cpu_user_time":1030,"agent_memory_avg":7487349,"used_memory":...} |

## Fasi successive

- Scopri come creare una dashboard di CloudWatch log in [Create a Dashboard](#) nella Amazon CloudWatch User Guide.
- Scopri come aggiungere query a una dashboard di CloudWatch log in [Aggiungi query alla dashboard o Esporta i risultati delle query](#) nella Amazon CloudWatch User Guide.

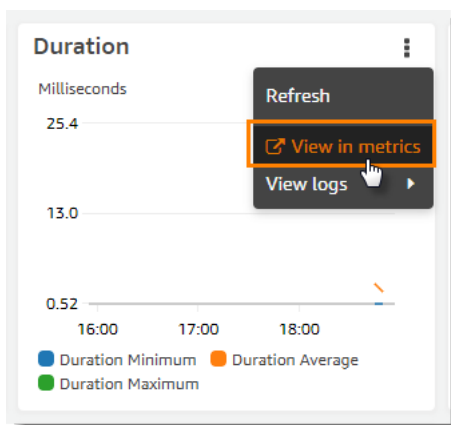


## Monitoraggio delle applicazioni Lambda

La sezione Applicazioni della console Lambda include una scheda Monitoraggio in cui puoi esaminare una CloudWatch dashboard Amazon con metriche aggregate per le risorse della tua applicazione.

Per monitorare un'applicazione Lambda

1. Aprire la [pagina Applicazioni](#) della console Lambda.
2. Selezionare Monitoring (Monitoraggio).
3. Per maggiori dettagli sui parametri in qualsiasi grafico, scegli Visualizza nei parametri dal menu a discesa.



Il grafico viene visualizzato in una nuova scheda, con i parametri pertinenti elencati sotto il grafico. Puoi personalizzare la visualizzazione di questo grafico, modificando i parametri e le risorse mostrate, le statistiche, il periodo e altri fattori per comprendere meglio la situazione attuale.

Per impostazione predefinita, la console Lambda mostra un pannello di controllo di base. Puoi personalizzare questa pagina aggiungendo uno o più CloudWatch dashboard Amazon al modello di applicazione con il tipo di risorsa [AWS::CloudWatch::Dashboard](#). Quando il modello include uno o più pannelli di controllo, la pagina mostra i pannelli di controllo anziché il pannello di controllo predefinito. Puoi passare da un pannello di controllo all'altro con il menu a discesa in alto a destra della pagina. Nell'esempio seguente viene creato un pannello di controllo con un singolo widget che rappresenta graficamente il numero di chiamate di una funzione denominata my-function.

## Example Modello di pannello di controllo delle funzioni

```
Resources:
  MyDashboard:
    Type: AWS::CloudWatch::Dashboard
    Properties:
      DashboardName: my-dashboard
      DashboardBody: |
        {
          "widgets": [
            {
              "type": "metric",
              "width": 12,
              "height": 6,
              "properties": {
                "metrics": [
                  [
                    "AWS/Lambda",
                    "Invocations",
                    "FunctionName",
                    "my-function",
                    {
                      "stat": "Sum",
                      "label": "MyFunction"
                    }
                  ],
                  [
                    {
                      "expression": "SUM(METRICS())",
                      "label": "Total Invocations"
                    }
                  ]
                ],
                "region": "us-east-1",
                "title": "Invocations",
                "view": "timeSeries",
                "stacked": false
              }
            }
          ]
        }
    }
```

Per ulteriori informazioni sulla creazione di CloudWatch dashboard e widget, consulta la [struttura del corpo e la sintassi della dashboard in](#) Amazon API Reference. CloudWatch

# Monitora le prestazioni delle applicazioni con Amazon CloudWatch Application Signals

Amazon CloudWatch Application Signals è una soluzione di monitoraggio delle prestazioni delle applicazioni (APM) che consente a sviluppatori e operatori di monitorare lo stato e le prestazioni delle loro applicazioni serverless create con Lambda. Puoi abilitare Application Signals con un clic dalla console Lambda e non è necessario aggiungere alcun codice di strumentazione o dipendenze esterne alla funzione Lambda. Dopo aver abilitato Application Signals, puoi visualizzare tutte le metriche e le tracce raccolte nella console. CloudWatch In questa pagina viene descritto come abilitare e visualizzare i dati di telemetria di Application Signals per le applicazioni.

## Argomenti

- [In che modo Application Signals si integra con Lambda](#)
- [Prezzi](#)
- [Runtime supportati](#)
- [Abilitazione di Application Signals nella console Lambda](#)
- [Utilizzo del pannello di controllo di Application Signals](#)

## In che modo Application Signals si integra con Lambda

Application Signals strumenta automaticamente le funzioni Lambda utilizzando [librerie avanzate di](#) AWS Distro for OpenTelemetry (ADOT), fornite tramite un layer Lambda. <https://docs.aws.amazon.com/lambda/latest/dg/chapter-layers.html> Application Signals legge i dati raccolti dal livello e genera pannelli di controllo con parametri delle prestazioni chiave per le applicazioni.

Puoi collegare questo livello con un clic [abilitando Application Signals](#) nella console Lambda. Quando attivi Application Signals dalla console, Lambda esegue le seguenti operazioni per conto tuo:

- Aggiorna il ruolo di esecuzione della funzione per includere `CloudWatchLambdaApplicationSignalsExecutionRolePolicy`. [Questa policy](#) fornisce l'accesso in scrittura AWS X-Ray e i gruppi di CloudWatch log utilizzati per Application Signals.
- Aggiunge un livello alla funzione che la strumenta automaticamente per acquisire dati di telemetria come richieste, disponibilità, latenza, errori e guasti. Per garantire che Application Signals funzioni correttamente, rimuovi qualsiasi codice di strumentazione X-Ray SDK esistente dalla tua funzione. Il codice di strumentazione X-Ray SDK personalizzato può interferire con la strumentazione fornita dal livello.

- Aggiunge la variabile di ambiente `AWS_LAMBDA_EXEC_WRAPPER` alla funzione e ne imposta il valore su `/opt/otel-instrument`. Questa variabile di ambiente modifica il comportamento di avvio della funzione per utilizzare il livello Application Signals ed è necessaria per una corretta strumentazione. Se questa variabile di ambiente esiste già, assicurati che sia impostata sul valore richiesto.

## Prezzi

L'utilizzo di Application Signals per le funzioni Lambda comporta dei costi. Per informazioni sui prezzi, consulta [i prezzi di Amazon. CloudWatch](#)

## Runtime supportati

L'integrazione di Application Signals con Lambda funziona con i seguenti runtime:

- .NET 8
- Java 11
- Java 17
- Java 21
- Python 3.10
- Python 3.11
- Python 3.12
- Python 3.13
- Node.js 18.x
- Node.js 20.x
- Node.js 22.x

## Abilitazione di Application Signals nella console Lambda

Puoi abilitare Application Signals su qualsiasi funzione Lambda esistente utilizzando un [runtime supportato](#). La procedura seguente descrive come abilitare Application Signals nella console Lambda.

Per abilitare Application Signals nella console Lambda

1. Aprire la pagina [Funzioni](#) della console Lambda.

2. Scegliere la funzione.
3. Scegli la scheda Configurazione.
4. Dal menu a sinistra, scegli Strumenti di monitoraggio e operazioni.
5. Nel riquadro Strumenti di monitoraggio aggiuntivi scegli Modifica.
6. In Segnali CloudWatch applicativi e AWS X-Ray, e in Segnali applicativi, scegli Abilita.
7. Scegli Save (Salva).

Se è la prima volta che abiliti Application Signals per la tua funzione, devi anche configurare una tantum il service discovery per Application Signals nella CloudWatch console. Dopo aver completato questa configurazione una tantum del rilevamento servizi, Application Signals rileva automaticamente tutte le funzioni Lambda aggiuntive per le quali abiliti Application Signals, in tutte le regioni.

#### Note

Dopo aver richiamato la funzione aggiornata, possono essere necessari fino a 10 minuti prima che i dati del servizio inizino a comparire nella dashboard di Application Signals nella CloudWatch console.

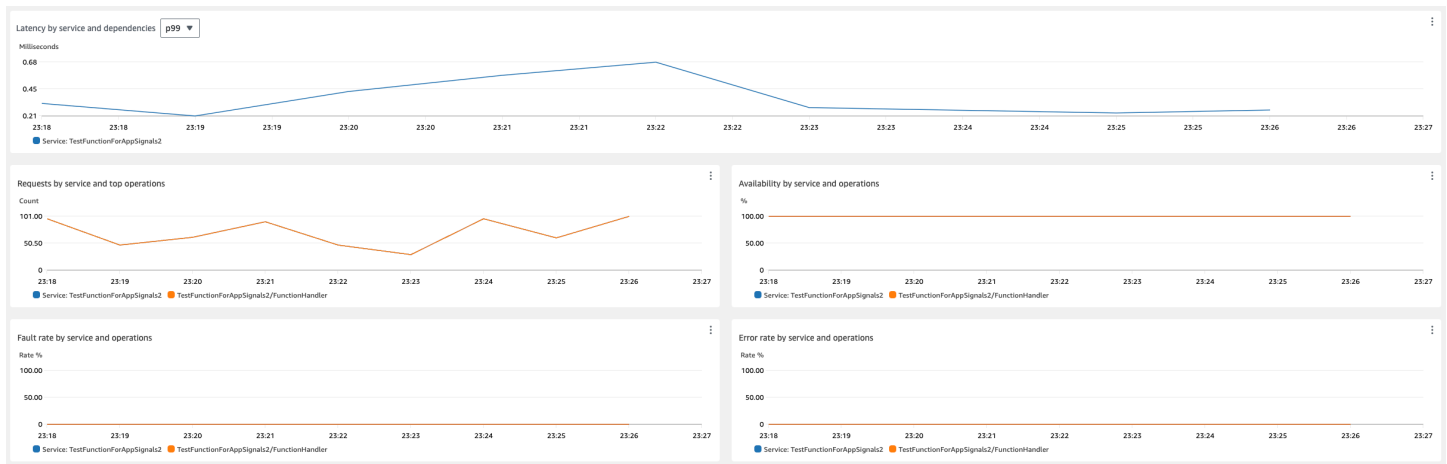
## Utilizzo del pannello di controllo di Application Signals

Dopo aver abilitato Application Signals per la tua funzione, puoi visualizzare le metriche dell'applicazione nella console. CloudWatch Puoi visualizzare rapidamente il pannello di controllo di Application Signals associato dalla console Lambda con i seguenti passaggi:

Per visualizzare il pannello di controllo di Application Signals relativo alla tua funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere la funzione.
3. Selezionare la scheda Monitor (Monitora).
4. Scegli il pulsante Visualizza Application Signals. Questo ti porta direttamente alla panoramica di Application Signals per il tuo servizio nella CloudWatch console.

Ad esempio, la schermata seguente mostra i parametri relativi a latenza, numero di richieste, disponibilità, tasso di guasti e tasso di errore per una funzione in una finestra temporale di 10 minuti.



Per sfruttare al meglio l'integrazione con Application Signals, puoi creare obiettivi a livello di servizio (SLO) per la tua applicazione. Ad esempio, puoi creare SLO di latenza per garantire che l'applicazione risponda rapidamente alle richieste degli utenti e SLO di disponibilità per monitorare il tempo di attività. Gli SLO possono aiutarti a rilevare il peggioramento delle prestazioni o le interruzioni prima che abbiano un impatto sugli utenti. Per ulteriori informazioni, consulta gli obiettivi [del livello di servizio \(SLO\)](#) nella Amazon User Guide. CloudWatch

# Esecuzione da remoto del debug delle funzioni Lambda con Visual Studio Code

Con la funzionalità di debug remoto in [AWS Toolkit for Visual Studio Code](#), puoi eseguire il debug delle tue funzioni Lambda in esecuzione direttamente nel cloud. AWS Ciò è utile quando si esaminano problemi difficili da replicare localmente o da diagnosticare solo con i log.

Con il debug da remoto, puoi:

- Impostare punti di interruzione nel codice della funzione Lambda.
- Eseguire il codice in tempo reale.
- Ispezionare le variabili e lo stato durante il runtime.
- Esegui il debug delle funzioni Lambda distribuite AWS, incluse quelle in VPC o con autorizzazioni IAM specifiche.

## Runtime supportati

Il debug da remoto è supportato per i seguenti runtime:

- Python (AL2023)
- Java
- JavaScript/Node.js (AL 2023)

### Note

Il debug da remoto è supportato per le architetture sia x86\_64 sia arm64.

## Sicurezza e debug da remoto

Il debug da remoto opera entro i limiti di sicurezza Lambda esistenti. Gli utenti possono collegare livelli a una funzione utilizzando l'autorizzazione `UpdateFunctionConfiguration`, che ha già la possibilità di accedere alle variabili di ambiente e alla configurazione della funzione. Il debug da remoto non va oltre queste autorizzazioni esistenti. Aggiunge invece ulteriori controlli di sicurezza attraverso il tunneling sicuro e la gestione automatica delle sessioni. Inoltre, il debug da remoto è interamente una funzionalità controllata dal cliente che richiede autorizzazioni e azioni esplicite:



- Creazione di un tunnel sicuro per l'IoT: il AWS Toolkit deve creare un tunnel sicuro per l'IoT, che si verifica solo con l'autorizzazione esplicita dell'utente. `iot:OpenTunnel`
- Gestione di allegati e token del livello di debug: il processo di debug mantiene la sicurezza attraverso questi controlli:
  - Il livello di debug deve essere collegato alla funzione Lambda e questo processo richiede le seguenti autorizzazioni: `lambda:UpdateFunctionConfiguration` e `lambda:GetLayerVersion`.
  - Aggiorna il token di sicurezza (generato tramite `iot:OpenTunnel`) nella variabile di ambiente della funzione prima di ogni sessione di debug. Questo passaggio richiede `lambda:UpdateFunctionConfiguration` anche.
  - Per motivi di sicurezza, questo token viene ruotato automaticamente e il livello di debug viene rimosso automaticamente alla fine di ogni sessione di debug e non può essere riutilizzato.

#### Note

Il debug da remoto è supportato per le architetture sia `x86_64` sia `arm64`.

## Prerequisiti

Prima di iniziare il debug da remoto, assicurati di disporre di quanto riportato di seguito:

1. Una funzione Lambda implementata nel tuo AWS account.
2. AWS Toolkit for Visual Studio Code. Consulta [Configurazione del AWS Toolkit for Visual Studio Code](#) per le istruzioni di installazione.
3. La versione del AWS Toolkit che hai installato è 3.69.0 o successiva.
4. AWS credenziali configurate in. AWS Toolkit for Visual Studio Code Per ulteriori informazioni, consulta [Autenticazione e controllo degli accessi](#).

## Debug da remoto delle funzioni Lambda

Esegui questi passaggi per avviare una sessione di debug da remoto:

1. Apri AWS Explorer in VS Code selezionando l' AWS icona nella barra laterale sinistra.
2. Espandi la sezione Lambda per vedere le tue funzioni.

3. Apri il menu contestuale della funzione di cui desideri eseguire il debug.
4. Dal menu contestuale, seleziona Richiama da remoto.
5. Nella finestra di invocazione che si apre, seleziona la casella Abilita debug.
6. Fai clic su Richiama per avviare la sessione di debug remota.

#### Note

Le funzioni Lambda hanno un limite combinato di 250 MB per il codice della funzione e tutti i livelli collegati. Il livello di debug da remoto aggiunge circa 40 MB alle dimensioni della funzione.

Una sessione di debug da remoto termina quando:

- Scegliere Rimuovi configurazione debug dalla schermata di configurazione Richiama da remoto.
- Selezionare l'icona di disconnessione nei controlli di debug di VS Code.
- Selezionare il file del gestore nell'editor VS Code.

#### Note

Il livello di debug viene rimosso automaticamente dopo 60 secondi di inattività dopo l'ultima invocazione.

## Disabilitazione de debug da remoto

Puoi disabilitare questa funzionalità in tre modi:

- Nega aggiornamenti delle funzioni: imposta `lambda:UpdateFunctionConfiguration` su deny.
- Limita le autorizzazioni IoT: nega le autorizzazioni IoT-related
- Blocca livelli di debug: nega `lambda:GetLayerVersion` per i seguenti ARN:
  - `arn:aws:lambda:*:*:layer:LDKLayerX86:*`
  - `arn:aws:lambda:*:*:layer:LDKLayerArm64:*`

 Note

La disabilitazione di questa funzionalità impedisce l'aggiunta del livello di debug durante gli aggiornamenti della configurazione delle funzioni.

## Informazioni aggiuntive

Per ulteriori informazioni sull'utilizzo di Lambda in VS Code, consulta [Sviluppo locale di funzioni Lambda con VS Code](#).

Per istruzioni dettagliate sulla risoluzione dei problemi, sui casi d'uso avanzati e sulla disponibilità geografica, consulta le funzioni Lambda per il debug [remoto nella Guida per l'utente](#). AWS Toolkit for Visual Studio Code

## Richiamare Lambda con eventi provenienti da altri AWS services

Alcuni Servizi AWS possono richiamare direttamente le funzioni Lambda utilizzando i trigger. Questi servizi inviano eventi a Lambda e la funzione viene richiamata immediatamente quando si verifica l'evento specificato. I trigger sono adatti per eventi discreti ed elaborazione in tempo reale. Quando si [crea un trigger utilizzando la console Lambda](#), la console interagisce con il AWS servizio corrispondente per configurare la notifica degli eventi su quel servizio. Il trigger viene effettivamente archiviato e gestito dal servizio che genera gli eventi, non da Lambda.

Gli eventi sono dati strutturati nel formato JSON. La struttura di JSON varia a seconda del servizio che la genera e del tipo di evento, ma contengono tutte i dati necessari alla funzione per elaborare l'evento.

Oltre alle chiamate basate sugli eventi, puoi utilizzare Lambda con [Amazon EC2](#), Apache Kafka autogestito e Kubernetes. [??? ???](#)

Una funzione può avere più trigger. Ogni trigger agisce come un client che invoca la funzione in modo indipendente e ogni evento che Lambda invia alla funzione contiene dati solo da un trigger. Lambda converte il documento di evento in un oggetto e lo passa al gestore funzione.

A seconda del servizio, l'invocazione basata sugli eventi può essere [sincrona](#) o [asincrona](#).

- Per la chiamata sincrona, il servizio che genera l'evento attende la risposta della funzione. Tale servizio definisce i dati che la funzione deve restituire nella risposta. Il servizio controlla la strategia di errore, ad esempio se riprovare in caso di errori.
- Per la chiamata asincrona, Lambda inserisce l'evento in una coda prima di passarlo alla funzione. Quando Lambda accoda l'evento, invia immediatamente una risposta riuscita al servizio che ha generato l'evento. Dopo che la funzione ha elaborato l'evento, Lambda non restituisce una risposta al servizio che genera l'evento.

## Creazione di un trigger

Il modo più semplice per creare un trigger consiste nell'utilizzare la console Lambda. Quando crei un trigger utilizzando la console, Lambda aggiunge automaticamente le autorizzazioni richieste alla [policy basata sulle risorse](#) della funzione.

Per creare un trigger utilizzando la console Lambda

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Seleziona la funzione per cui desideri creare un trigger.
3. Nel riquadro Panoramica della funzione, scegli Aggiungi trigger.
4. Seleziona il AWS servizio per il quale desideri richiamare la tua funzione.
5. Compila le opzioni nel riquadro Configurazione trigger e scegli Aggiungi. A seconda del tipo di funzione Servizio AWS che scegli di richiamare, le opzioni di configurazione del trigger sono diverse.

## Servizi che possono richiamare le funzioni Lambda

La tabella seguente elenca i servizi che possono richiamare le funzioni Lambda.

| Servizio                                                  | Metodo di chiamata                                               |
|-----------------------------------------------------------|------------------------------------------------------------------|
| <a href="#">Amazon Managed Streaming per Apache Kafka</a> | <a href="#">Strumento di mappatura dell'origine degli eventi</a> |
| <a href="#">Self-managed Apache Kafka</a>                 | <a href="#">Strumento di mappatura dell'origine degli eventi</a> |
| <a href="#">Gateway Amazon API</a>                        | Event-driven; invocazione sincrona                               |

| Servizio                                                              | Metodo di chiamata                                                                                 |
|-----------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| <a href="#">AWS CloudFormation</a>                                    | Event-driven; invocazione asincrona                                                                |
| <a href="#">CloudWatch Amazon Logs</a>                                | Event-driven; invocazione asincrona                                                                |
| <a href="#">AWS CodeCommit</a>                                        | Event-driven; invocazione asincrona                                                                |
| <a href="#">AWS CodePipeline</a>                                      | Event-driven; invocazione asincrona                                                                |
| <a href="#">Amazon Cognito</a>                                        | Event-driven; invocazione sincrona                                                                 |
| <a href="#">AWS Config</a>                                            | Event-driven; invocazione asincrona                                                                |
| <a href="#">Connetti il cliente</a>                                   | Event-driven; invocazione sincrona                                                                 |
| <a href="#">Amazon DocumentDB</a>                                     | <a href="#">Strumento di mappatura dell'origine degli eventi</a>                                   |
| <a href="#">Amazon DynamoDB</a>                                       | <a href="#">Strumento di mappatura dell'origine degli eventi</a>                                   |
| <a href="#">Elastic Load Balancer<br/>(Application Load Balancer)</a> | Event-driven; invocazione sincrona                                                                 |
| <a href="#">EventBridge</a>                                           | Event-driven; invocazione asincrona (bus e regole di eventi), chiamata sincrona o asincrona (pipe) |
| <a href="#">EventBridge Amazon Scheduler</a>                          | Event-driven (basato sul tempo); chiamata asincrona                                                |
| <a href="#">AWS IoT</a>                                               | Event-driven; invocazione asincrona                                                                |
| <a href="#">Amazon Kinesis</a>                                        | <a href="#">Strumento di mappatura dell'origine degli eventi</a>                                   |
| <a href="#">Amazon Data Firehose</a>                                  | Event-driven; invocazione sincrona                                                                 |
| <a href="#">Amazon Lex</a>                                            | Event-driven; invocazione sincrona                                                                 |
| <a href="#">Amazon MQ</a>                                             | <a href="#">Strumento di mappatura dell'origine degli eventi</a>                                   |
| <a href="#">Amazon Simple Email Service</a>                           | Event-driven; invocazione asincrona                                                                |

| Servizio                                                  | Metodo di chiamata                                               |
|-----------------------------------------------------------|------------------------------------------------------------------|
| <a href="#">Amazon Simple Notification Service</a>        | Event-driven; invocazione asincrona                              |
| <a href="#">Amazon Simple Queue Service</a>               | <a href="#">Strumento di mappatura dell'origine degli eventi</a> |
| <a href="#">Amazon Simple Storage Service (Amazon S3)</a> | Event-driven; invocazione asincrona                              |
| <a href="#">Batch di Amazon Simple Storage Service</a>    | Event-driven; invocazione sincrona                               |
| <a href="#">Secrets Manager</a>                           | Rotazione del segreto                                            |
| <a href="#">AWS Step Functions</a>                        | Event-driven; invocazione sincrona o asincrona                   |
| <a href="#">Amazon VPC Lattice</a>                        | Event-driven; invocazione sincrona                               |

## Usare Lambda con Apache Kafka

Lambda supporta [Apache Kafka](#) come [origine eventi](#). Apache Kafka è una piattaforma di streaming di eventi open source progettata per gestire pipeline di dati e applicazioni di streaming ad alto rendimento e in tempo reale. Esistono due modi principali per utilizzare Lambda con Apache Kafka:

- [the section called “MSK”](#)— Amazon Managed Streaming for Apache Kafka (Amazon MSK) è un servizio completamente gestito da AWS. Amazon MSK aiuta ad automatizzare la gestione dell'infrastruttura Kafka, inclusi il provisioning, l'applicazione di patch e la scalabilità.
- [the section called “Apache Kafka gestito dal cliente”](#)— In AWS terminologia, un cluster autogestito include cluster Kafka non ospitati AWS. [Ad esempio, puoi ancora utilizzare Lambda con un cluster Kafka ospitato presso un provider non cloud come Confluent AWS Cloud o Redpanda.](#)

Quando decidi tra Amazon MSK e Apache Kafka autogestito, considera le tue esigenze operative e i requisiti di controllo. Amazon MSK è la scelta migliore se desideri aiutarti AWS a gestire rapidamente una configurazione Kafka scalabile e pronta per la produzione con un sovraccarico operativo minimo. Semplifica la sicurezza, il monitoraggio e l'elevata disponibilità, aiutandoti a concentrarti sullo sviluppo di applicazioni piuttosto che sulla gestione dell'infrastruttura. D'altra parte, Apache Kafka autogestito è più adatto per i casi d'uso eseguiti su ambienti non AWS ospitati, inclusi i cluster locali.

### Argomenti

- [Uso di Lambda con Amazon MSK](#)
- [Utilizzo di Lambda con Apache Kafka gestito dal cliente](#)
- [Modalità di ridimensionamento del poller di eventi di Apache Kafka in Lambda](#)
- [Sondaggi di Apache Kafka e posizioni di partenza per lo streaming in Lambda](#)
- [ID gruppo di consumer personalizzabile](#)
- [Filtraggio eventi da origini eventi Amazon MSK e Apache Kafka autogestito](#)
- [Utilizzo di registri di schema con sorgenti di eventi Kafka in Lambda](#)
- [Elaborazione a bassa latenza per sorgenti di eventi Kafka](#)
- [Configurazione dei controlli di gestione degli errori per le sorgenti di eventi Kafka](#)
- [Acquisizione di batch scartati per un'origine eventi di Amazon MSK e Apache Kafka autogestito](#)
- [Usare un argomento di Kafka come destinazione in caso di errore](#)
- [Registrazione della mappatura delle sorgenti degli eventi Kafka](#)
- [Risoluzione degli errori di mappatura delle sorgenti degli eventi Kafka](#)





## Uso di Lambda con Amazon MSK

[Amazon Managed Streaming for Apache Kafka \(Amazon MSK\)](#) è un servizio completamente gestito [che puoi utilizzare per](#) creare ed eseguire applicazioni che utilizzano Apache Kafka per elaborare dati di streaming. Amazon MSK semplifica la configurazione, la scalabilità e la gestione dei cluster Kafka. Amazon MSK semplifica inoltre la configurazione dell'applicazione per più zone di disponibilità e per la sicurezza con AWS Identity and Access Management (IAM).

Questo capitolo spiega come utilizzare un cluster Amazon MSK come origine di eventi per la funzione Lambda. Il processo generale per l'integrazione di Amazon MSK con Lambda prevede i seguenti passaggi:

1. [Configurazione del cluster e della rete](#): per prima cosa, configura il tuo [cluster Amazon MSK](#). Ciò include la configurazione di rete corretta per consentire a Lambda di accedere al cluster.
2. [Configurazione della mappatura delle sorgenti degli eventi](#): quindi, crea la risorsa di [mappatura delle sorgenti degli eventi](#) di cui Lambda ha bisogno per connettere in modo sicuro il tuo cluster Amazon MSK alla tua funzione.
3. [Configurazione delle funzioni e delle autorizzazioni](#): infine, assicurati che la funzione sia configurata correttamente e disponga delle autorizzazioni necessarie per il suo ruolo di esecuzione.

### Note

Ora puoi creare e gestire le mappature delle sorgenti degli eventi Amazon MSK direttamente dalla console Lambda o Amazon MSK. Entrambe le console offrono la possibilità di gestire automaticamente la configurazione delle autorizzazioni necessarie per il ruolo di esecuzione Lambda per un processo di configurazione più semplificato.

Per esempi su come configurare un'integrazione Lambda con un cluster Amazon MSK, consulta [the section called “Tutorial”](#) Utilizzo di Amazon MSK [come origine di eventi per AWS Lambda](#) sul AWS Compute Blog e Integrazione di Amazon MSK [Lambda negli Amazon MSK](#) Labs.

### Argomenti

- [Esempio di evento](#)
- [Configurazione del cluster Amazon MSK e della rete Amazon VPC per Lambda](#)

- [Configurazione delle autorizzazioni Lambda per le mappature delle sorgenti degli eventi di Amazon MSK](#)
- [Configurazione delle origini eventi di Amazon MSK per Lambda](#)
- [Tutorial: Utilizzo di uno strumento di mappatura dell'origine degli eventi Amazon MSK per richiamare una funzione Lambda](#)

## Esempio di evento

Lambda invia il batch di messaggi nel parametro evento quando richiama la funzione. Il payload evento contiene un array di messaggi. Ogni elemento dell'array contiene i dettagli dell'argomento e dell'identificatore dello shard Amazon MSK, insieme a una data/ora e a un messaggio con codifica base64.

```
{
  "eventSource": "aws:kafka",
  "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
  "bootstrapServers": "b-2.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092,b-1.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092",
  "records": {
    "mytopic-0": [
      {
        "topic": "mytopic",
        "partition": 0,
        "offset": 15,
        "timestamp": 1545084650987,
        "timestampType": "CREATE_TIME",
        "key": "abcDEFghiJKLmnoPQRstuVWXYZ1234==",
        "value": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
        "headers": [
          {
            "headerKey": [
              104,
              101,
              97,
              100,
              101,
              114,
              86,
              97,
```

```

    108,
    117,
    101
  ]
}
]
}
]
}
}
}

```

## Configurazione del cluster Amazon MSK e della rete Amazon VPC per Lambda

Per connettere la tua AWS Lambda funzione al tuo cluster Amazon MSK, devi configurare correttamente il cluster e l'[Amazon Virtual Private Cloud \(VPC\) in cui risiede](#). Questa pagina descrive come configurare il cluster e il VPC. Se il cluster e il VPC sono già configurati correttamente, consulta [the section called “Configurare un'origine eventi”](#) per configurare la mappatura dell'origine degli eventi.

### Argomenti

- [Panoramica dei requisiti di configurazione di rete per le integrazioni Lambda e MSK](#)
- [Configurazione di un gateway NAT per una sorgente di eventi MSK](#)
- [Configurazione AWS PrivateLink endpoint per una sorgente di eventi MSK](#)

## Panoramica dei requisiti di configurazione di rete per le integrazioni Lambda e MSK

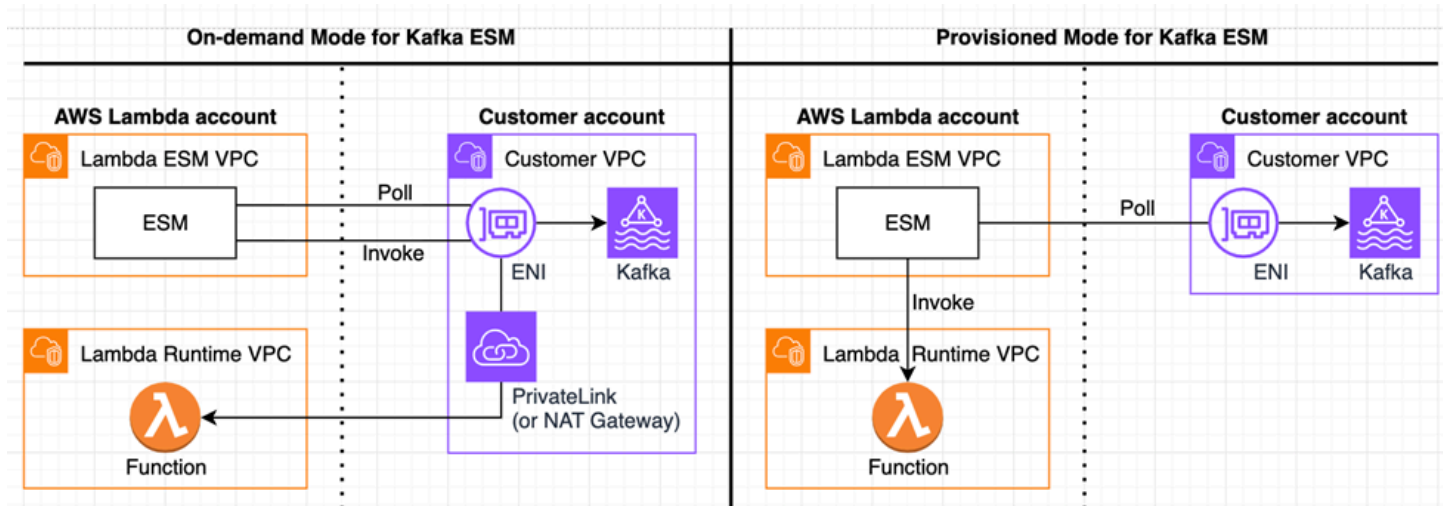
La configurazione di rete richiesta per un'integrazione tra Lambda e MSK dipende dall'architettura di rete dell'applicazione. Sono tre le risorse principali coinvolte in questa integrazione: il cluster Amazon MSK, la funzione Lambda e la mappatura delle sorgenti di eventi Lambda. Ognuna di queste risorse risiede in un VPC diverso:

- Il cluster Amazon MSK si trova in genere in una sottorete privata di un VPC che gestisci.
- La tua funzione Lambda risiede in un AWS VPC gestito di proprietà di Lambda.
- La mappatura dell'origine degli eventi Lambda risiede in un altro VPC AWS gestito di proprietà di Lambda, separato dal VPC che contiene la tua funzione.

La mappatura [dell'origine degli eventi](#) è la risorsa intermedia tra il cluster MSK e la funzione Lambda. La mappatura dell'origine degli eventi ha due funzioni principali. Innanzitutto, esegue il

polling del cluster MSK per verificare la presenza di nuovi messaggi. Quindi, richiama la funzione Lambda con quei messaggi. Poiché queste tre risorse si trovano in VPC diversi, sia le operazioni di polling che di invoke richiedono chiamate di rete tra VPC.

I requisiti di configurazione di rete per la mappatura dell'origine degli eventi dipendono dal fatto che utilizzi la modalità [provisioned](#) o la modalità on-demand, come mostrato nel diagramma seguente:



Il modo in cui la mappatura delle sorgenti di eventi Lambda esegue il polling del cluster MSK per individuare nuovi messaggi è lo stesso in entrambe le modalità. Per stabilire una connessione tra la mappatura dell'origine degli eventi e il cluster MSK, Lambda crea un ENI [iperpiano](#) (o ne riutilizza uno esistente, se disponibile) nella sottorete privata per stabilire una connessione sicura. Come illustrato nel diagramma, questo iperpiano ENI utilizza la configurazione della sottorete e del gruppo di sicurezza del cluster MSK, non della funzione Lambda.

Dopo aver analizzato il messaggio dal cluster, il modo in cui Lambda richiama la funzione è diverso in ciascuna modalità:

- In modalità provisioned, Lambda gestisce automaticamente la connessione tra il VPC di mappatura della sorgente dell'evento e la funzione VPC. Quindi, non è necessario alcun componente di rete aggiuntivo per richiamare correttamente la funzione.
- In modalità on-demand, la mappatura dell'origine degli eventi Lambda richiama la funzione utilizzando un percorso attraverso il VPC gestito dal cliente. Per questo motivo, è necessario configurare un gateway [NAT nella sottorete pubblica del VPC o gli AWS PrivateLink endpoint](#) nella sottorete privata del VPC che forniscono l'accesso a Lambda, (STS) e, facoltativamente, [Servizio di token di sicurezza AWS AWS Secrets Manager](#). La corretta configurazione di una di queste opzioni consente una connessione tra il VPC e il VPC di Lambda-managed runtime, necessaria per richiamare la funzione.

Un gateway NAT consente alle risorse della sottorete privata di accedere alla rete Internet pubblica. L'utilizzo di questa configurazione significa che il traffico attraversa Internet prima di richiamare la funzione Lambda. AWS PrivateLink gli endpoint consentono alle sottoreti private di connettersi in modo sicuro a AWS servizi o altre risorse VPC private senza attraversare la rete Internet pubblica. Consulta [the section called “Configurazione di un gateway NAT per una sorgente di eventi MSK”](#) o [the section called “Configurazione AWS PrivateLink endpoint per una sorgente di eventi MSK”](#) per i dettagli su come configurare queste risorse.

Finora, abbiamo ipotizzato che il cluster MSK risieda in una sottorete privata all'interno del VPC, che è il caso più comune. Tuttavia, anche se il cluster MSK si trova in una sottorete pubblica all'interno del VPC, è necessario configurare AWS PrivateLink gli endpoint per abilitare una connessione sicura. La tabella seguente riassume i requisiti di configurazione di rete in base alla configurazione del cluster MSK e alla mappatura delle sorgenti di eventi Lambda:

| Posizione del cluster MSK (in un VPC gestito dal cliente) | modalità di ridimensionamento della mappatura della sorgente degli eventi Lambda | Configurazione di rete richiesta                                                                                                                                                              |
|-----------------------------------------------------------|----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Sottorete privata                                         | On-demand modalità                                                               | Gateway NAT (nella sottorete pubblica del VPC) o AWS PrivateLink endpoint (nella sottorete privata del VPC) per consentire l'accesso a Lambda e, facoltativamente, AWS STS a Secrets Manager. |
| Sottorete pubblica                                        | On-demand modalità                                                               | AWS PrivateLink endpoint (nella sottorete pubblica del tuo VPC) per consentire l'accesso a Lambda e, facoltativamente AWS STS, a Secrets Manager.                                             |
| Sottorete privata                                         | Modalità provisioning                                                            | Nessuno                                                                                                                                                                                       |
| Sottorete pubblica                                        | Modalità provisioning                                                            | Nessuno                                                                                                                                                                                       |

Inoltre, i gruppi di sicurezza associati al cluster MSK devono consentire il traffico sulle porte corrette. Assicurati di aver configurato le seguenti regole del gruppo di sicurezza:

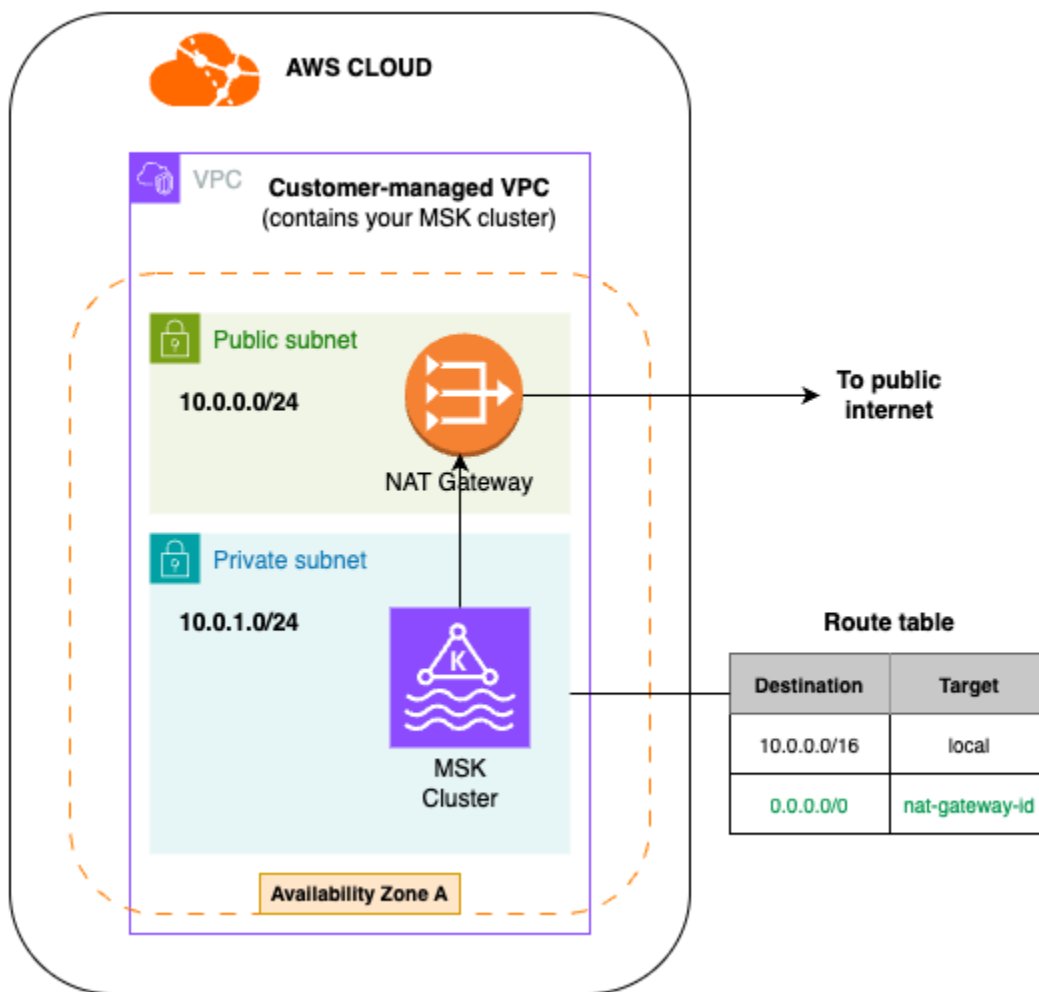
- Regole in entrata: consenti tutto il traffico sulla porta del broker predefinita. La porta utilizzata da MSK dipende dal tipo di autenticazione sul cluster: 9098 per l'autenticazione IAM, per e 9096 9094 per SASL/SCRAM TLS. In alternativa, puoi utilizzare una regola del gruppo di sicurezza autoreferenziale per consentire l'accesso da istanze all'interno dello stesso gruppo di sicurezza.
- Regole in uscita: consenti tutto il traffico sulla porta 443 per destinazioni esterne se la tua funzione deve comunicare con altri servizi. AWS In alternativa, puoi utilizzare una regola di gruppo di sicurezza autoreferenziale per limitare l'accesso al broker se non hai bisogno di comunicare con altri servizi. AWS
- Regole in entrata per gli endpoint Amazon VPC: se utilizzi un endpoint Amazon VPC, il gruppo di sicurezza associato all'endpoint deve consentire il traffico in entrata sulla porta dal gruppo di sicurezza del cluster. 443

### Configurazione di un gateway NAT per una sorgente di eventi MSK

È possibile configurare un gateway NAT per consentire alla mappatura della sorgente degli eventi di interrogare i messaggi dal cluster e richiamare la funzione utilizzando un percorso attraverso il VPC. Ciò è necessario solo se la mappatura dell'origine degli eventi utilizza la modalità su richiesta e il cluster risiede all'interno di una sottorete privata del VPC. Se il cluster risiede in una sottorete pubblica del VPC o la mappatura dell'origine degli eventi utilizza la modalità provisioned, non è necessario configurare un gateway NAT.

Le risorse in una sottorete privata richiedono un [gateway NAT](#) per accedere alla rete Internet pubblica. Se hai bisogno di una connettività privata a Lambda, consulta invece. [the section called "Configurazione AWS PrivateLink endpoint per una sorgente di eventi MSK"](#)

Dopo aver configurato il gateway NAT, è necessario configurare le tabelle di routing appropriate. Ciò consente al traffico proveniente dalla sottorete privata di essere indirizzato alla rete Internet pubblica tramite il gateway NAT.



I passaggi seguenti guidano l'utente nella configurazione di un gateway NAT tramite la console. Ripetere questi passaggi se necessario per ciascuna zona di disponibilità (AZ).

Per configurare un gateway NAT e il routing corretto (console)

1. Segui i passaggi descritti in [Creazione di un gateway NAT](#), tenendo presente quanto segue:
  - I gateway NAT devono sempre risiedere in una sottorete pubblica. Crea gateway NAT con connettività pubblica. <https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html>
  - Se il tuo cluster MSK è replicato su più AZ, crea un gateway NAT per AZ. Ad esempio, in ogni AZ, il VPC dovrebbe avere una sottorete privata contenente il cluster e una sottorete pubblica contenente il gateway NAT. Per una configurazione con tre AZ, avrai tre sottoreti private, tre sottoreti pubbliche e tre gateway NAT.
2. Dopo aver creato il gateway NAT, apri la console <https://console.aws.amazon.com/vpc/> Amazon VPC e scegli Route tables nel menu a sinistra.

3. Selezionare Create route table (Crea tabella di instradamento).
4. Associa questa tabella di routing al VPC che contiene il cluster MSK. Facoltativamente, inserisci un nome per la tabella delle rotte.
5. Selezionare Create route table (Crea tabella di instradamento).
6. Scegli la tabella delle rotte che hai appena creato.
7. Nella scheda Associazioni di sottorete, scegli Modifica associazioni di sottorete.
  - Associa questa tabella di routing alla sottorete privata che contiene il cluster MSK.
8. Selezionare Modifica route.
9. Scegli Aggiungi percorso:
  1. Per Destinazione scegliere 0.0.0.0/0.
  2. Per Target, scegli NAT gateway.
  3. Nella casella di ricerca, scegli il gateway NAT che hai creato nel passaggio 1. Dovrebbe essere il gateway NAT nella stessa AZ della sottorete privata che contiene il cluster MSK (la sottorete privata associata a questa tabella di routing nel passaggio 6).
10. Scegli Save changes (Salva modifiche).

### Configurazione AWS PrivateLink endpoint per una sorgente di eventi MSK

Puoi configurare gli AWS PrivateLink endpoint per interrogare i messaggi dal tuo cluster e richiamare la funzione utilizzando un percorso attraverso il tuo VPC. Questi endpoint dovrebbero consentire al cluster MSK di accedere a quanto segue:

- Il servizio Lambda
- Il [Servizio di token di sicurezza AWS \(STS\)](#)
- Facoltativamente, il [AWS Secrets Manager](#) servizio. Ciò è necessario se il segreto richiesto per l'autenticazione del cluster è archiviato in Secrets Manager.

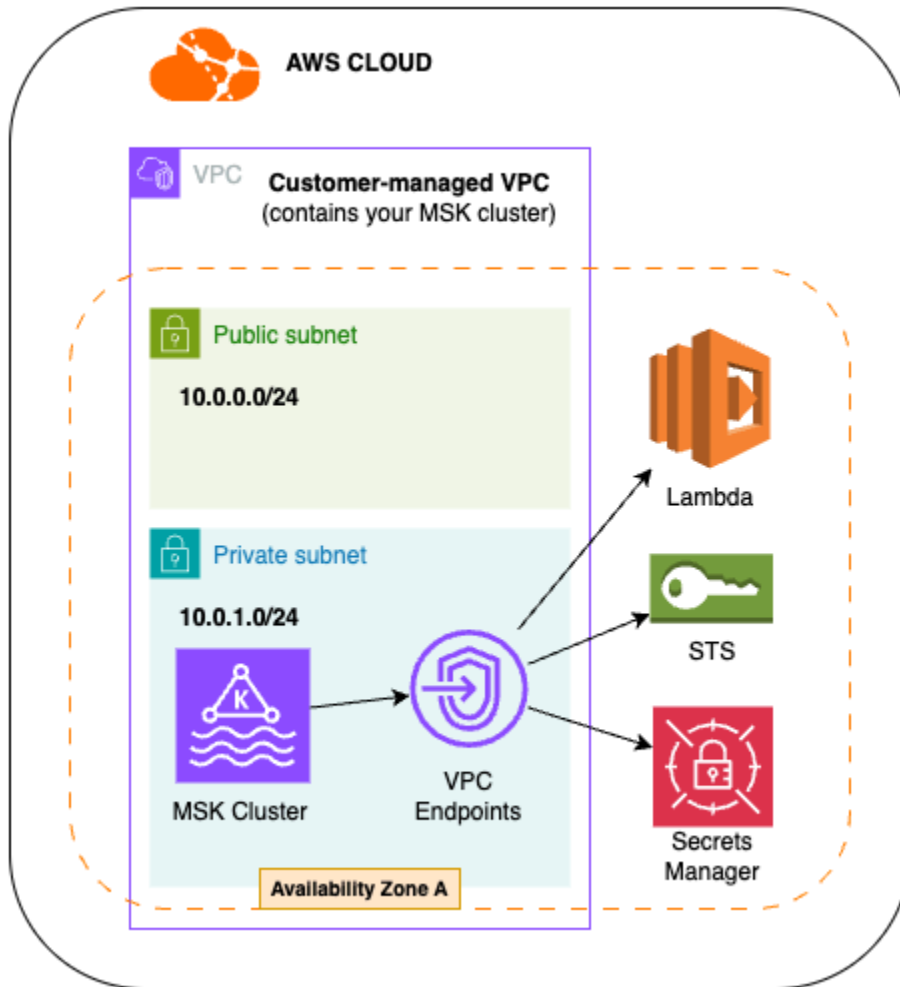
La configurazione PrivateLink degli endpoint è richiesta solo se la mappatura dell'origine degli eventi utilizza la modalità su richiesta. Se la mappatura dell'origine degli eventi utilizza la modalità provisioned, Lambda stabilisce le connessioni richieste per te.

PrivateLink gli endpoint consentono un accesso sicuro e privato ai servizi tramite AWS AWS PrivateLink. In alternativa, per configurare un gateway NAT per consentire al cluster MSK di accedere



alla rete Internet pubblica, consulta. [the section called “Configurazione di un gateway NAT per una sorgente di eventi MSK”](#)

Dopo aver configurato gli endpoint VPC, il cluster MSK dovrebbe avere accesso diretto e privato a Lambda, STS e, facoltativamente, a Secrets Manager.



I passaggi seguenti guidano l'utente nella configurazione di un PrivateLink endpoint tramite la console. Ripeti questi passaggi se necessario per ciascun endpoint (Lambda, STS, Secrets Manager).

Per configurare gli PrivateLink endpoint VPC (console)

1. Apri la console [Amazon VPC](#) e scegli **Endpoints** nel menu a sinistra.
2. Seleziona **Crea endpoint**.
3. Facoltativamente, inserisci un nome per il tuo endpoint.
4. Per **Tipo**, scegli **AWS servizi**.

5. In Servizi, inizia a digitare il nome del servizio. Ad esempio, per creare un endpoint per connettersi a Lambda, digita `lambda` nella casella di ricerca.
6. Nei risultati, dovresti vedere l'endpoint del servizio nella regione corrente. Ad esempio, nella regione Stati Uniti orientali (Virginia settentrionale), dovresti vedere `com.amazonaws.us-east-2.lambda`. Seleziona questo servizio.
7. In Impostazioni di rete, seleziona il VPC che contiene il tuo cluster MSK.
8. In Subnet, seleziona l'AZ in cui si trova il tuo cluster MSK.
  - Per ogni AZ, in Subnet ID, scegli la sottorete privata che contiene il tuo cluster MSK.
9. In Gruppo di sicurezza, seleziona i gruppi di sicurezza associati al cluster MSK.
10. Seleziona Crea endpoint.

Per impostazione predefinita, gli endpoint Amazon VPC dispongono di policy IAM aperte che consentono un ampio accesso alle risorse. La best practice consiste nel limitare queste policy per eseguire le azioni necessarie utilizzando quell'endpoint. Ad esempio, per il tuo endpoint Secrets Manager, puoi modificarne la politica in modo che consenta solo il ruolo di esecuzione della tua funzione di accedere al segreto.

Example Politica sugli endpoint VPC: endpoint Secrets Manager

```
{
  "Statement": [
    {
      "Action": "secretsmanager:GetSecretValue",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws::iam::123456789012:role/my-role"
        ]
      },
      "Resource": "arn:aws::secretsmanager:us-west-2:123456789012:secret:my-secret"
    }
  ]
}
```

Per gli endpoint Lambda AWS STS e Lambda, è possibile limitare il principale chiamante al principale del servizio Lambda. Tuttavia, assicurati di utilizzarlo `"Resource": "*" in queste politiche.`

## Example Politica sugli endpoint VPC — AWS STS endpoint

```
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

## Example Politica sugli endpoint VPC: endpoint Lambda

```
{
  "Statement": [
    {
      "Action": "lambda:InvokeFunction",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

## Configurazione delle autorizzazioni Lambda per le mappature delle sorgenti degli eventi di Amazon MSK

Per accedere al cluster Amazon MSK, la mappatura delle funzioni e delle sorgenti degli eventi richiede le autorizzazioni per eseguire varie azioni dell'API Amazon MSK. Aggiungi queste autorizzazioni al ruolo di esecuzione della funzione. [???](#) Se i tuoi utenti hanno bisogno di accedere, aggiungi le autorizzazioni richieste alla politica di identità per l'utente o il ruolo.

La policy [AWSLambdaMSKExecutionRole](#) gestita contiene le autorizzazioni minime richieste per le mappature delle sorgenti degli eventi Amazon MSK Lambda. Per semplificare il processo di autorizzazione, puoi:

- Allega la policy [AWSLambdaMSKExecutionRole](#) gestita al tuo ruolo di esecuzione.
- Lascia che la console Lambda generi le autorizzazioni per te. Quando [crei una mappatura dell'origine degli eventi di Amazon MSK nella console](#), Lambda valuta il tuo ruolo di esecuzione e ti avvisa in caso di mancanza di autorizzazioni. Scegli [Genera autorizzazioni](#) per aggiornare automaticamente il tuo ruolo di esecuzione. Questo non funziona se hai creato o modificato manualmente le politiche del ruolo di esecuzione o se le politiche sono collegate a più ruoli. Tieni presente che potrebbero essere comunque necessarie autorizzazioni aggiuntive nel tuo ruolo di esecuzione quando utilizzi funzionalità avanzate come [On-Failure Destination](#) o [AWS Glue Schema Registry](#).

## Argomenti

- [Autorizzazioni richieste](#)
- [Autorizzazioni facoltative](#)

## Autorizzazioni richieste

Il tuo ruolo di esecuzione della funzione Lambda deve disporre delle seguenti autorizzazioni richieste per le mappature delle sorgenti degli eventi di Amazon MSK. Queste autorizzazioni sono incluse nella policy gestita. [AWSLambdaMSKExecutionRole](#)

## CloudWatch Registra le autorizzazioni

Le seguenti autorizzazioni consentono a Lambda di creare e archiviare log in Amazon Logs. CloudWatch

- [log: CreateLogGroup](#)
- [registri: CreateLogStream](#)
- [registri: PutLogEvents](#)

## autorizzazioni del cluster MSK

Le seguenti autorizzazioni consentono a Lambda di accedere al tuo cluster Amazon MSK per tuo conto:

- [kafka: DescribeCluster](#)
- [kafka: V2 DescribeCluster](#)
- [kafka: GetBootstrapBrokers](#)

Ti consigliamo di usare [kafka: DescribeCluster V2](#) invece di kafka:. [DescribeCluster](#)

L'autorizzazione v2 funziona con cluster Amazon MSK con provisioning e serverless. È necessaria solo una di queste autorizzazioni nella tua policy.

### Autorizzazioni VPC

Le seguenti autorizzazioni consentono a Lambda di creare e gestire interfacce di rete durante la connessione al cluster Amazon MSK:

- [ec2: CreateNetworkInterface](#)
- [ec2: DescribeNetworkInterfaces](#)
- [ec2: DescribeVpcs](#)
- [ec2: DeleteNetworkInterface](#)
- [ec2: DescribeSubnets](#)
- [ec2: DescribeSecurityGroups](#)

### Autorizzazioni facoltative

La funzione Lambda potrebbe richiedere autorizzazioni per:

- Accedi ai cluster Amazon MSK con più account. Per le mappature delle sorgenti degli eventi tra più account, hai bisogno di [kafka: nel ruolo di esecuzione. DescribeVpcConnection](#). Un responsabile IAM che crea una mappatura dell'origine degli eventi tra più account necessita di kafka:. [ListVpcConnections](#)
- Accedi al tuo segreto SCRAM, se stai utilizzando l'autenticazione. [SASL/SCRAM](#). In questo modo, la tua funzione può utilizzare un nome utente e una password per connettersi a Kafka.
- Descrivi il tuo segreto di Secrets Manager, se stai utilizzando l'autenticazione SASL/SCRAM [MTLS](#). Con questa autorizzazione, la tua funzione può recuperare le credenziali o i certificati necessari per connessioni sicure.
- Accedi alla tua chiave gestita AWS KMS dal cliente, se la tua chiave AWS Secrets Manager segreta è crittografata con una chiave gestita AWS KMS dal cliente.

- Accedi ai segreti del tuo registro di schema, se utilizzi un registro di schemi con autenticazione:
  - Per AWS Glue Schema Registry: le esigenze `glue:GetRegistry` e le `glue:GetSchemaVersion` autorizzazioni della tua funzione. Questi consentono alla funzione di cercare e utilizzare le regole di formato dei messaggi memorizzate in AWS Glue.
  - Per [Confluent Schema Registry](#) con BASIC\_AUTH o CLIENT\_CERTIFICATE\_TLS\_AUTH: la funzione richiede l'`secretsmanager:GetSecretValue` autorizzazione per il segreto contenente le credenziali di autenticazione. Ciò consente alla funzione di recuperare i certificati username/password o necessari per accedere al Confluent Schema Registry.
  - Per i certificati CA privati: la tua funzione richiede `secretsmanager:GetSecretValue` autorizzazione per il segreto contenente il certificato. Con questa autorizzazione, la funzione può verificare l'identità dei registri di schema che utilizzano certificati personalizzati.
- Accedi ai gruppi di consumatori del cluster Kafka e ai messaggi di sondaggio dall'argomento, se utilizzi l'autenticazione IAM per la mappatura dell'origine degli eventi.

Queste corrispondono alle seguenti autorizzazioni richieste:

- [kafka: ListScramSecrets](#) - Consente l'elenco dei segreti SCRAM per l'autenticazione Kafka
- [secretsmanager: GetSecretValue](#) - Consente il recupero dei segreti da Secrets Manager
- [KMS:Decrypt](#) - Permette la decrittografia dei dati crittografati utilizzando AWS KMS
- [glue: - Consente l'accesso allo Schema Registry GetRegistry](#) AWS Glue
- [glue: GetSchemaVersion](#) - Consente il recupero di versioni specifiche dello schema dal AWS Glue registro degli schemi
- [Kafka-cluster:connect](#) - Concede il permesso di connettersi e autenticarsi al cluster
- [kafka-cluster: AlterGroup](#) - Concede il permesso di unirsi a gruppi su un cluster, equivalente all'ACL READ GROUP di Apache Kafka
- [kafka-cluster: DescribeGroup](#) - Concede il permesso di descrivere i gruppi su un cluster, equivalente all'ACL DESCRIBE GROUP di Apache Kafka
- [kafka-cluster: DescribeTopic](#) - Concede il permesso di descrivere argomenti su un cluster, equivalente all'ACL DESCRIBE TOPIC di Apache Kafka
- [kafka-cluster: ReadData](#) - Concede il permesso di leggere i dati dagli argomenti su un cluster, equivalente all'ACL READ TOPIC di Apache Kafka

Inoltre, se desideri inviare record di chiamate non riuscite a una destinazione in caso di errore, avrai bisogno delle seguenti autorizzazioni a seconda del tipo di destinazione:

- Per le destinazioni Amazon SQS: [sqs: SendMessage](#) - Consente l'invio di messaggi a una coda Amazon SQS
- Per le destinazioni Amazon SNS: [SNS:Publish: consente di pubblicare messaggi su un argomento Amazon SNS](#)
- Per le destinazioni bucket Amazon S3: e [s3: ListBucket - Consente di scrivere PutObject ed elencare oggetti in un bucket Amazon S3](#)

Per la risoluzione degli errori di autenticazione e autorizzazione, consulta. [the section called “Risoluzione dei problemi”](#)

## Configurazione delle origini eventi di Amazon MSK per Lambda

Per utilizzare un cluster Amazon MSK come origine di eventi per la tua funzione Lambda, crei una mappatura [dell'origine degli eventi](#) che connette le due risorse. Questa pagina descrive come creare una mappatura dell'origine degli eventi per Amazon MSK.

Questa pagina presuppone che tu abbia già configurato correttamente il cluster MSK e l'[Amazon Virtual Private Cloud \(VPC\) in cui risiede](#). Se devi configurare il tuo cluster o VPC, consulta. [the section called “Configurazione del cluster e della rete”](#) Per configurare il comportamento dei nuovi tentativi per la gestione degli errori, consulta. [the section called “Riprova le configurazioni”](#)

## Argomenti

- [Utilizzo di un cluster Amazon MSK come origine eventi](#)
- [Configurazione dei metodi di autenticazione del cluster Amazon MSK in Lambda](#)
- [Creazione di una mappatura della sorgente di eventi Lambda per una sorgente di eventi Amazon MSK](#)
- [Creazione di mappature delle sorgenti di eventi tra più account in Lambda](#)
- [Tutti i parametri di configurazione dell'origine degli eventi Amazon MSK in Lambda](#)

## Utilizzo di un cluster Amazon MSK come origine eventi

Quando aggiungi il cluster Apache Kafka o Amazon MSK come trigger per la funzione Lambda, il cluster viene utilizzato come [origine eventi](#).

Lambda legge i dati degli eventi dagli argomenti di Kafka specificati Topics in una [CreateEventSourceMapping](#) richiesta, in base alla posizione [???](#) iniziale specificata. Dopo che l'elaborazione è avvenuta con successo, l'argomento Kafka viene salvato nel cluster Kafka.

Lambda legge i messaggi in sequenza per ogni partizione dell'argomento Kafka. Un singolo payload Lambda può contenere messaggi provenienti da più partizioni. Quando sono disponibili più record, Lambda continua a elaborare i record in batch, in base al BatchSize valore specificato in una [CreateEventSourceMapping](#) richiesta, finché la funzione non raggiunge l'argomento.

Dopo che Lambda ha elaborato ogni batch, esegue il commit degli offset dei messaggi in quel batch. Se la funzione restituisce un errore per uno qualsiasi dei messaggi di un batch, Lambda ritenta l'intero batch di messaggi fino a quando l'elaborazione non riesce o i messaggi scadono. È possibile inviare i record per i quali tutti i nuovi tentativi falliscono a una destinazione in errore per un'elaborazione successiva.

#### Note

Anche se le funzioni Lambda generalmente prevedono un timeout massimo di 15 minuti, gli strumenti di mappatura dell'origine degli eventi per Amazon MSK, Apache Kafka autogestito, Amazon DocumentDB e Amazon MQ per ActiveMQ e RabbitMQ supportano solo funzioni con timeout massimi di 14 minuti.

## Configurazione dei metodi di autenticazione del cluster Amazon MSK in Lambda

Lambda necessita dell'autorizzazione per accedere al cluster Amazon MSK, recuperare i record ed eseguire altre attività. Amazon MSK supporta diversi metodi per l'autenticazione con il cluster MSK.

### Metodi di autenticazione del cluster

- [Accesso non autenticato](#)
- [SASL/SCRAM authentication](#)
- [Autenticazione TLS reciproca](#)
- [Autenticazione IAM](#)
- [Come Lambda sceglie un broker bootstrap](#)

### Accesso non autenticato

Se nessun client accede al cluster tramite Internet, è possibile utilizzare l'accesso non autenticato.



## SASL/SCRAM authentication

Lambda supporta l'autenticazione [Simple Authentication e Security Layer/Salted Challenge Response](#) Authentication Mechanism (SASL/SCRAM), con funzione SHA-512 hash e crittografia Transport Layer Security (TLS). Affinché Lambda si connetta al cluster, memorizza le credenziali di autenticazione (nome utente e password) in un segreto di Secrets Manager e fai riferimento a questo segreto durante la configurazione della mappatura della sorgente dell'evento.

Per ulteriori informazioni sull'utilizzo di Secrets Manager, consulta l'autenticazione [Sign-in delle credenziali con Secrets Manager](#) nella Amazon Managed Streaming for Apache Kafka Developer Guide.

### Note

Amazon MSK non supporta l'autenticazione. SASL/PLAIN

## Autenticazione TLS reciproca

Mutual TLS (MTLS) fornisce l'autenticazione bidirezionale tra il client e il server. Il client invia un certificato al server affinché il server verifichi il client. Il server invia anche un certificato al client per consentire al client di verificare il server.

Per le integrazioni Amazon MSK con Lambda, il cluster MSK funge da server e Lambda funge da client.

- Affinché Lambda verifichi il cluster MSK, devi configurare un certificato client come segreto in Secrets Manager e fare riferimento a questo certificato nella configurazione di mappatura dell'origine degli eventi. Il certificato client deve essere firmato da un'autorità di certificazione (CA) presente nel trust store del server.
- Il cluster MSK invia anche un certificato server a Lambda. Il certificato del server deve essere firmato da un'autorità di certificazione (CA) nel AWS trust store.

Amazon MSK non supporta i certificati server autofirmati. Tutti i broker di Amazon MSK utilizzano certificati [pubblici](#) firmati dalle CA di [Amazon Trust Services](#), di cui Lambda si fida per impostazione predefinita.

## Configurazione del segreto mTLS

Il segreto `CLIENT_CERTIFICATE_TLS_AUTH` richiede un campo certificato e un campo chiave privata. Per una chiave privata crittografata, il segreto richiede una password per chiave privata. Il certificato e la chiave privata devono essere in formato PEM.

### Note

Lambda supporta il [PBES1](#) (ma non PBES2) come algoritmi di crittografia a chiave privata.

Il campo certificato deve contenere un elenco di certificati, a partire dal certificato client, seguito da qualsiasi certificato intermedio, per finire con il certificato root. Ogni certificato deve iniziare su una nuova riga con la struttura seguente:

```
-----BEGIN CERTIFICATE-----
    <certificate contents>
-----END CERTIFICATE-----
```

Secrets Manager supporta segreti fino a 65.536 byte, che è uno spazio sufficiente per lunghe catene di certificati.

La chiave privata deve essere in formato [PKCS #8](#), con la struttura seguente:

```
-----BEGIN PRIVATE KEY-----
    <private key contents>
-----END PRIVATE KEY-----
```

Per una chiave privata crittografata, utilizza la struttura seguente:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
    <private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

Nell'esempio seguente viene mostrato il contenuto di un segreto per l'autenticazione mTLS utilizzando una chiave privata crittografata. Per una chiave privata crittografata, includi una password per chiave privata nel segreto.

```
{
```

```

    "privateKeyPassword": "testpassword",
    "certificate": "-----BEGIN CERTIFICATE-----
MIIE5DCCAsygAwIBAgIRAPJdwaFaNRrytHBto0j5BA0wDQYJKoZIhvcNAQELBQAw
...
j0Lh4/+1HfgyE2KlmII36dg4IMzNjAFEBZiCRoPim040s1cRqtFHxoa10QQbIlxk
cmUuiAii9R0=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFgjCCA2qgAwIBAgIQdJNZd6uFf9hbNC5RdfmHrzANBgkqhkiG9w0BAQsFADBb
...
rQoiowbbk5wXCheYSANQIfTZ6weQTgiCHCCbuuMKNVS95FkXm0vqVD/YpXKwA/no
c8PH3PSOAaRwMMg0SA2ALJvbRz8mpg==
-----END CERTIFICATE-----",
    "privateKey": "-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFKzBVBgkqhkiG9w0BBQ0wSDANBgkqhkiG9w0BBQwwGgQUiAFcK5hT/X7Kjmgp
...
QrSekqF+kWzmB6nAfSzg09IaoAaytLvNgGTckWeUkwn/V0Ck+LdGUXzAC4RxZnoQ
zp2mwJn2NYB7AZ7+imp0azDZb+8YG2aUCiyqb6PnnA==
-----END ENCRYPTED PRIVATE KEY-----"
}

```

Per ulteriori informazioni sugli MTL per Amazon MSK e le istruzioni su come generare un certificato client, consulta [Mutual TLS client authentication for Amazon MSK](#) nella Amazon Managed Streaming for Apache Kafka Developer Guide.

## Autenticazione IAM

Puoi utilizzare AWS Identity and Access Management (IAM) per autenticare l'identità dei client che si connettono al cluster MSK. Con l'autenticazione IAM, Lambda si basa sulle autorizzazioni nel [ruolo di esecuzione](#) della funzione per connettersi al cluster, recuperare i record ed eseguire altre azioni necessarie. Per un esempio di policy che contiene le autorizzazioni necessarie, consulta [Create authorization policies for the IAM role](#) nella Amazon Managed Streaming for Apache Kafka Developer Guide.

Se l'autenticazione IAM è attiva sul tuo cluster MSK e non fornisci un segreto, Lambda utilizza automaticamente l'autenticazione IAM per impostazione predefinita.

Per ulteriori informazioni sull'autenticazione IAM in Amazon MSK, consulta IAM access control. <https://docs.aws.amazon.com/msk/latest/developerguide/iam-access-control.html>

## Come Lambda sceglie un broker bootstrap

Lambda sceglie un [broker bootstrap](#) in base ai metodi di autenticazione disponibili nel tuo cluster e se fornisci un segreto per l'autenticazione. Se fornisci un segreto per MTL o SASL/SCRAM, Lambda sceglie automaticamente quel metodo di autenticazione. Se non fornisci un segreto, Lambda seleziona il metodo di autenticazione più forte attivo sul tuo cluster. Di seguito è riportato l'ordine di priorità in cui Lambda seleziona un broker, dall'autenticazione più forte a quella più debole:

- mTLS (segreto fornito per mTLS)
- SASL/SCRAM (segreto previsto) SASL/SCRAM
- IAM SASL (nessun segreto fornito e autenticazione IAM attiva)
- TLS non autenticato (nessun segreto fornito e autenticazione IAM non attiva)
- Testo semplice (nessun segreto fornito e autenticazione IAM e TLS non autenticato non attivi)

### Note

Se Lambda non riesce a connettersi al tipo di broker più sicuro, non proverà a connettersi a un tipo di broker diverso (più debole). Se vuoi che Lambda scelga un tipo di broker più debole, disattiva tutti i metodi di autenticazione più forti sul tuo cluster.

## Creazione di una mappatura della sorgente di eventi Lambda per una sorgente di eventi Amazon MSK

Per creare una mappatura dell'origine degli eventi, puoi utilizzare la console Lambda, [AWS Command Line Interface \(CLI\)](#) o un SDK. [AWS](#)

### Note

Quando crei la mappatura dell'origine degli eventi, Lambda crea un ENI [iperpiano](#) nella sottorete privata che contiene il cluster MSK, consentendo a Lambda di stabilire una connessione sicura. Questo iperpiano consentito da ENI utilizza la configurazione della sottorete e del gruppo di sicurezza del cluster MSK, non della funzione Lambda.

I seguenti passaggi della console aggiungono un cluster Amazon MSK come trigger per la funzione Lambda. Sotto il cofano, questo crea una risorsa di mappatura dell'origine degli eventi.

## Per aggiungere un trigger Amazon MSK alla funzione Lambda (console)

1. Apri la [pagina Funzione](#) della console Lambda.
2. Scegli il nome della funzione Lambda per la quale si desidera aggiungere un trigger Amazon MSK.
3. In Panoramica delle funzioni, scegliere Aggiungi trigger.
4. In Trigger configuration, scegli MSK.
5. Per specificare i dettagli del cluster Kafka, procedi come segue:
  1. Per MSK cluster (Cluster MSK) seleziona il cluster.
  2. Per Nome argomento, inserisci il nome dell'argomento Kafka da cui consumare i messaggi.
  3. Per ID del gruppo di consumatori, inserisci l'ID di un gruppo di consumatori Kafka a cui aderire, se applicabile. Per ulteriori informazioni, consulta [the section called "Gruppo di consumer"](#).
6. Per l'autenticazione del cluster, effettua le configurazioni necessarie. Per ulteriori informazioni sull'autenticazione cluster, consulta [the section called "Autenticazione cluster"](#).
  - Attiva Usa l'autenticazione se desideri che Lambda esegua l'autenticazione con il tuo cluster MSK quando stabilisce una connessione. L'autenticazione è consigliata.
  - Se utilizzi l'autenticazione, per Metodo di autenticazione, scegli il metodo di autenticazione da utilizzare.
  - Se usi l'autenticazione, per la chiave Secrets Manager, scegli la chiave Secrets Manager che contiene le credenziali di autenticazione necessarie per accedere al tuo cluster.
7. In Event poller configuration, effettua le configurazioni necessarie.
  - Scegliete Attiva il trigger per abilitare il trigger subito dopo la creazione.
  - Scegliete se volete configurare la modalità predisposta per la mappatura della sorgente dell'evento. Per ulteriori informazioni, consulta [the section called "Ridimensionamento del poller di eventi"](#).
    - Se configuri la modalità con provisioning, inserisci un valore per Minimum event pollers, un valore per Maximum event pollers e un valore opzionale per specificare il raggruppamento di più ESM PollerGroupName all'interno dello stesso VPC di origine degli eventi.
  - Per Posizione iniziale, scegli come vuoi che Lambda inizi a leggere dal tuo stream. Per ulteriori informazioni, consulta [the section called "Posizioni nei sondaggi e nello streaming"](#).

8. In **Batching**, effettua le configurazioni necessarie. Per ulteriori informazioni sul batch, vedere [the section called “Comportamento di batching”](#).
  1. Per **Batch Size** (Dimensione batch), immettere il numero massimo di messaggi da recuperare in un singolo batch.
  2. Nella finestra **Batch**, inserisci il numero massimo di secondi che Lambda impiega a raccogliere i record prima di richiamare la funzione.
9. In **Filtraggio**, effettua le configurazioni necessarie. Per ulteriori informazioni sul filtro, consulta [the section called “Filtro eventi”](#).
  - Per i criteri di filtro, aggiungete le definizioni dei criteri di filtro per determinare se elaborare o meno un evento.
10. In **Gestione degli errori**, effettuate le configurazioni necessarie. Per ulteriori informazioni sulla gestione degli errori, vedere [the section called “Conserva le chiamate non riuscite”](#).
  - Per la **On-failure destinazione**, specifica l'ARN della destinazione in caso di errore.
11. Per **Tag**, inserisci i tag da associare a questa mappatura della sorgente dell'evento.
12. Per creare il trigger, scegli **Aggiungi**.

Puoi anche creare la mappatura della sorgente dell'evento utilizzando la AWS CLI con il comando `create-event-source-mapping`. <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/create-event-source-mapping.html> L'esempio seguente crea una mappatura della sorgente di eventi per mappare la funzione Lambda all'argomento, a partire dal messaggio. `my-msk-function` `AWSKafkaTopic` `LATEST` Questo comando utilizza anche l'[SourceAccessConfiguration](#) oggetto per indicare a Lambda di utilizzare l'[SASL/SCRAM](#) autenticazione durante la connessione al cluster.

```
aws lambda create-event-source-mapping \  
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-  
fd1b-45ad-85dd-15b4a5a6247e-2 \  
  --topics AWSKafkaTopic \  
  --starting-position LATEST \  
  --function-name my-kafka-function \  
  --source-access-configurations '[{"Type": "SASL_SCRAM_512_AUTH", "URI":  
"arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret"}]'
```

Se il cluster utilizza l'autenticazione [MTLS](#), includi un [SourceAccessConfiguration](#) oggetto che specifichi `CLIENT_CERTIFICATE_TLS_AUTH` e una chiave ARN di Secrets Manager. Ciò è mostrato nel comando seguente:

```
aws lambda create-event-source-mapping \  
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-  
fd1b-45ad-85dd-15b4a5a6247e-2 \  
  --topics AWSKafkaTopic \  
  --starting-position LATEST \  
  --function-name my-kafka-function  
  --source-access-configurations '[{"Type": "CLIENT_CERTIFICATE_TLS_AUTH", "URI":  
"arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret"}]'
```

Quando il cluster utilizza l'autenticazione [IAM](#), non è necessario un [SourceAccessConfiguration](#) oggetto. Questo è mostrato nel comando seguente:

```
aws lambda create-event-source-mapping \  
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-  
fd1b-45ad-85dd-15b4a5a6247e-2 \  
  --topics AWSKafkaTopic \  
  --starting-position LATEST \  
  --function-name my-kafka-function
```

## Creazione di mappature delle sorgenti di eventi tra più account in Lambda

È possibile utilizzare la connettività privata [multi-VPC](#) per connettere una funzione Lambda a un cluster MSK di cui è stato eseguito il provisioning in un altro cluster. Account AWS Multi-VPC usi della connettività AWS PrivateLink, che mantiene tutto il traffico all'interno della rete. AWS

### Note

Non è possibile creare strumenti di mappatura dell'origine degli eventi multi-account per i cluster MSK serverless.

Per creare uno strumento di mappatura dell'origine degli eventi multi-account, è necessario innanzitutto [configurare la connettività multi-VPC per il cluster MSK](#). Quando crei lo strumento di mappatura dell'origine degli eventi, utilizza l'ARN della connessione VPC gestita anziché l'ARN del cluster, come illustrato negli esempi seguenti. L'[CreateEventSourceMapping](#) operazione è inoltre diversa a seconda del tipo di autenticazione utilizzato dal cluster MSK.

Example— Creazione di uno strumento di mappatura dell'origine degli eventi multi-account per cluster che utilizzano l'autenticazione IAM

Quando il cluster utilizza l'autenticazione [IAM basata sui ruoli](#), non è necessario un oggetto.

[SourceAccessConfiguration](#) Esempio:

```
aws lambda create-event-source-mapping \  
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:vpc-connection/444455556666/  
my-cluster-name/51jn98b4-0a61-46cc-b0a6-61g9a3d797d5-7 \  
  --topics AWSKafkaTopic \  
  --starting-position LATEST \  
  --function-name my-kafka-function
```

Example— Crea una mappatura dell'origine degli eventi tra più account per il cluster che utilizza l'autenticazione SASL/SCRAM

Se il cluster utilizza [SASL/SCRAM l'autenticazione](#), è necessario includere un [SourceAccessConfiguration](#) oggetto che specifichi SASL\_SCRAM\_512\_AUTH e un ARN segreto di Secrets Manager.

Esistono due modi per utilizzare i segreti per la mappatura delle sorgenti di eventi Amazon MSK tra più account con autenticazione: SASL/SCRAM

- Crea un segreto nell'account della funzione Lambda e sincronizzalo con il segreto del cluster. [Crea una rotazione](#) per mantenere sincronizzati i due segreti. Questa opzione ti consente di controllare il segreto dall'account della funzione.
- Utilizza il segreto associato al cluster MSK. Questo segreto deve consentire l'accesso multi-account all'account della funzione Lambda. Per ulteriori informazioni, consulta la pagina [Permissions to AWS Secrets Manager secrets for users in a different account](#).

```
aws lambda create-event-source-mapping \  
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:vpc-connection/444455556666/  
my-cluster-name/51jn98b4-0a61-46cc-b0a6-61g9a3d797d5-7 \  
  --topics AWSKafkaTopic \  
  --starting-position LATEST \  
  --function-name my-kafka-function \  
  --source-access-configurations '[{"Type": "SASL_SCRAM_512_AUTH", "URI":  
"arn:aws:secretsmanager:us-east-1:444455556666:secret:my-secret"}]'
```



Example— Creazione di uno strumento di mappatura dell'origine degli eventi multi-account per cluster che utilizzano l'autenticazione mTLS

Se il cluster utilizza l'autenticazione [MTLS](#), è necessario includere un [SourceAccessConfiguration](#) oggetto che specifichi CLIENT\_CERTIFICATE\_TLS\_AUTH e un ARN segreto di Secrets Manager. Il segreto può essere archiviato nell'account del cluster o nell'account della funzione Lambda.

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:vpc-connection/444455556666/
my-cluster-name/51jn98b4-0a61-46cc-b0a6-61g9a3d797d5-7 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function \
  --source-access-configurations '[{"Type": "CLIENT_CERTIFICATE_TLS_AUTH", "URI":
"arn:aws:secretsmanager:us-east-1:444455556666:secret:my-secret"}]'
```

Tutti i parametri di configurazione dell'origine degli eventi Amazon MSK in Lambda

Tutti i tipi di sorgenti di eventi Lambda condividono le stesse operazioni [CreateEventSourceMapping](#) e [UpdateEventSourceMapping](#) le API. Tuttavia, solo alcuni dei parametri si applicano ad Amazon MSK, come mostrato nella tabella seguente.

| Parametro                           | Obbligatorio | Predefinito                                                                              | Note                                                                   |
|-------------------------------------|--------------|------------------------------------------------------------------------------------------|------------------------------------------------------------------------|
| AmazonManagedKafkaEventSourceConfig | N            | Contiene il ConsumerGroupID campo, che per impostazione predefinita è un valore univoco. | Può essere impostato solo su Create                                    |
| BatchSize                           | N            | 100                                                                                      | Massimo: 10.000.                                                       |
| DestinationConfig                   | N            | N/A                                                                                      | <a href="#">the section called “Conserva le chiamate non riuscite”</a> |
| Abilitato                           | N            | True                                                                                     |                                                                        |

| Parametro                      | Obbligatorio | Predefinito   | Note                                                                    |
|--------------------------------|--------------|---------------|-------------------------------------------------------------------------|
| BisectBatchOnFunctionError     | N            | False         | <a href="#">the section called “Riprova le configurazioni”</a>          |
| FunctionResponseTypes          | N            | N/A           | <a href="#">the section called “Riprova le configurazioni”</a>          |
| MaximumRecordAgeInSeconds      | N            | -1 (infinito) | <a href="#">the section called “Riprova le configurazioni”</a>          |
| MaximumRetryAttempts           | N            | -1 (infinito) | <a href="#">the section called “Riprova le configurazioni”</a>          |
| EventSourceArn                 | Y            | N/A           | Può essere impostato solo su Create                                     |
| FilterCriteria                 | N            | N/A           | <a href="#">Controllare gli eventi che Lambda invia alla funzione</a>   |
| FunctionName                   | Y            | N/A           |                                                                         |
| KMSKeyArn                      | N            | N/A           | <a href="#">the section called “Crittografia dei criteri di filtro”</a> |
| MaximumBatchingWindowInSeconds | N            | 500 ms        | <a href="#">Comportamento di batching</a>                               |

| Parametro                      | Obbligatorio | Predefinito                                                                                                                                                                                        | Note                                                                                                                              |
|--------------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| ProvisionedPollers<br>Config   | N            | <p>MinimumPollers :<br/>se non specificato, il<br/>valore predefinito è 1</p> <p>MaximumPollers :<br/>se non specificato, il<br/>valore predefinito è<br/>200</p> <p>PollerGroupName :<br/>N/A</p> | <a href="#">the section called<br/>“Modalità provision<br/>ing”</a>                                                               |
| SourceAccessConfig<br>urations | N            | Nessuna credenziale                                                                                                                                                                                | SASL/SCRAM<br>o credenziali di<br>autenticazione<br>CLIENT_C<br>RTIFICATE<br>_TLS_AUTH<br>(MutualTLS) per la<br>fonte dell'evento |
| StartingPosition               | Y            | N/A                                                                                                                                                                                                | <p>AT_TIMESTAMP,<br/>TRIM_HORIZON o<br/>LATEST</p> <p>Può essere impostato<br/>solo su Create</p>                                 |
| StartingPositionTi<br>mestamp  | N            | N/A                                                                                                                                                                                                | Obbligatorio se<br>è impostato su<br>StartingPosition<br>AT_TIMESTAMP                                                             |

| Parametro | Obbligatorio | Predefinito | Note                                                                                            |
|-----------|--------------|-------------|-------------------------------------------------------------------------------------------------|
| Tag       | N            | N/A         | <a href="#">the section called “Tag dello strumento di mappatura dell'origine degli eventi”</a> |
| Argomenti | Y            | N/A         | Nome argomento<br>Kafka<br><br>Può essere impostato solo su Create                              |

### Note

Quando si specifica un `PollerGroupName`, più ESM all'interno dello stesso Amazon VPC possono condividere la capacità di Event Poller Unit (EPU). È possibile utilizzare questa opzione per ottimizzare i costi in modalità Provisioned per i propri ESM. Requisiti per il raggruppamento ESM:

- Gli ESM devono trovarsi all'interno dello stesso Amazon VPC
- Massimo 100 ESM per gruppo di poller
- Il numero massimo aggregato di sondaggi per tutti gli ESM di un gruppo non può superare 2000

È possibile aggiornarlo `PollerGroupName` per spostare un ESM in un gruppo diverso o rimuovere un ESM da un gruppo `PollerGroupName` impostandolo su una stringa vuota (`«»`).

**Tutorial:** Utilizzo di uno strumento di mappatura dell'origine degli eventi Amazon MSK per richiamare una funzione Lambda

In questo tutorial, esegui quanto segue:

- Crea una funzione Lambda nello stesso AWS account di un cluster Amazon MSK esistente.
- Configura la rete e l'autenticazione per consentire a Lambda di comunicare con Amazon MSK.

- Configura uno strumento di mappatura dell'origine degli eventi Lambda Amazon MSK, che esegue la funzione Lambda quando gli eventi vengono visualizzati nell'argomento.

Dopo aver completato questi passaggi, quando gli eventi vengono inviati ad Amazon MSK, puoi configurare una funzione Lambda per elaborare tali eventi automaticamente con il tuo codice Lambda personalizzato.

Cosa puoi fare con questa funzionalità?

Soluzione di esempio: utilizza uno strumento di mappatura dell'origine degli eventi MSK per fornire risultati in tempo reale ai tuoi clienti.

Considera lo scenario seguente: la tua azienda ospita un'applicazione Web in cui i clienti possono visualizzare informazioni sugli eventi dal vivo, come le partite sportive. Gli aggiornamenti delle informazioni dal gioco vengono forniti al tuo team tramite un argomento Kafka su Amazon MSK. Vuoi progettare una soluzione che utilizzi gli aggiornamenti dell'argomento MSK per fornire una visione aggiornata dell'evento dal vivo ai clienti all'interno di un'applicazione da te sviluppata. Hai scelto il seguente approccio di progettazione: le tue applicazioni client comunicano con un backend serverless ospitato in AWS. I client si connettono tramite sessioni websocket utilizzando l'API Amazon API Gateway. WebSocket

In questa soluzione, è necessario un componente che legga gli eventi MSK, esegua una logica personalizzata per preparare tali eventi per il livello dell'applicazione e quindi inoltri tali informazioni all'API Gateway. Puoi implementare questo componente fornendo la tua logica personalizzata in una funzione Lambda, quindi chiamandola con una mappatura dell'origine degli eventi di AWS Lambda Amazon MSK. AWS Lambda

Per ulteriori informazioni sull'implementazione di soluzioni utilizzando l'API Amazon API Gateway WebSocket, consulta i tutorial sulle [WebSocket API](#) nella documentazione di API Gateway.

## Prerequisiti

Un AWS account con le seguenti risorse preconfigurate:

Per soddisfare questi prerequisiti, ti consigliamo di consultare la sezione [Guida introduttiva all'uso di Amazon MSK](#) nella documentazione di Amazon MSK.

- UN cluster Amazon MSK. Consulta [Creare un cluster Amazon MSK](#) in Guida introduttiva all'uso di Amazon MSK.

- La seguente configurazione:
  - Assicurati che l'autenticazione basata sui ruoli IAM sia abilitata nelle impostazioni di sicurezza del cluster. Ciò aumenta la sicurezza limitando la funzione Lambda al solo accesso alle risorse Amazon MSK necessarie. Questa funzionalità è abilitata per impostazione predefinita per i nuovi cluster Amazon MSK.
  - Assicurati che l'accesso pubblico sia disattivato nelle impostazioni di rete del cluster. Limitare l'accesso a Internet del cluster Amazon MSK migliora la sicurezza limitando il numero di intermediari che gestiscono i dati. Questa funzionalità è abilitata per impostazione predefinita per i nuovi cluster Amazon MSK.
- Un argomento Kafka nel tuo cluster Amazon MSK da utilizzare per questa soluzione. Consulta [Creare un argomento](#) in Guida introduttiva all'uso di Amazon MSK.
- Un host di amministrazione Kafka configurato per recuperare informazioni dal tuo cluster Kafka e inviare eventi Kafka al tuo argomento per i test, ad esempio un'istanza Amazon EC2 con la CLI di amministrazione Kafka e la libreria IAM Amazon MSK installate. Consulta [Creare una macchina client](#) in Guida introduttiva all'uso di Amazon MSK.

Dopo aver impostato queste risorse, raccogli le seguenti informazioni dal tuo AWS account per confermare che sei pronto a continuare.

- Il nome del cluster Amazon MSK. È possibile trovare queste informazioni nella console Amazon MSK.
- L'UUID del cluster, parte dell'ARN per il tuo cluster Amazon MSK, che puoi trovare nella console Amazon MSK. Segui le procedure in [Elencare i cluster](#) nella documentazione di Amazon MSK per trovare queste informazioni.
- I gruppi di sicurezza associati al cluster Amazon MSK. È possibile trovare queste informazioni nella console Amazon MSK. Nei passaggi seguenti, chiamale tue *clusterSecurityGroups*.
- L'ID Amazon VPC contenente il cluster Amazon MSK. Puoi trovare queste informazioni identificando le sottoreti associate al tuo cluster Amazon MSK nella console Amazon MSK, quindi identificando l'Amazon VPC associato alla sottorete nella console Amazon VPC.
- Il nome dell'argomento Kafka usato nella tua soluzione. Puoi trovare queste informazioni chiamando il tuo cluster Amazon MSK con l'`topics` CLI di Kafka dal tuo host di amministrazione Kafka. Per ulteriori informazioni sugli argomenti della CLI, consulta [Aggiungere e rimuovere argomenti](#) nella documentazione di Kafka.
- Il nome di un gruppo di consumer per l'argomento Kafka, adatto all'uso con la funzione Lambda. Questo gruppo può essere creato automaticamente da Lambda, quindi non devi crearlo con la

CLI Kafka. Se devi gestire i tuoi gruppi di consumer, per saperne di più sulla CLI dei gruppi di consumer, consulta [Gestione dei gruppi di consumer](#) nella documentazione di Kafka.

Le seguenti autorizzazioni nel tuo AWS account:

- Autorizzazione per creare e gestire una funzione Lambda.
- Autorizzazione per creare policy IAM e associarle alla funzione Lambda.
- Autorizzazione a creare endpoint Amazon VPC e modificare la configurazione di rete nell'Amazon VPC che ospita il cluster Amazon MSK.

Installa il AWS Command Line Interface

Se non hai ancora installato il AWS Command Line Interface, segui i passaggi in [Installazione o aggiornamento della versione più recente AWS CLI](#) per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

#### Note

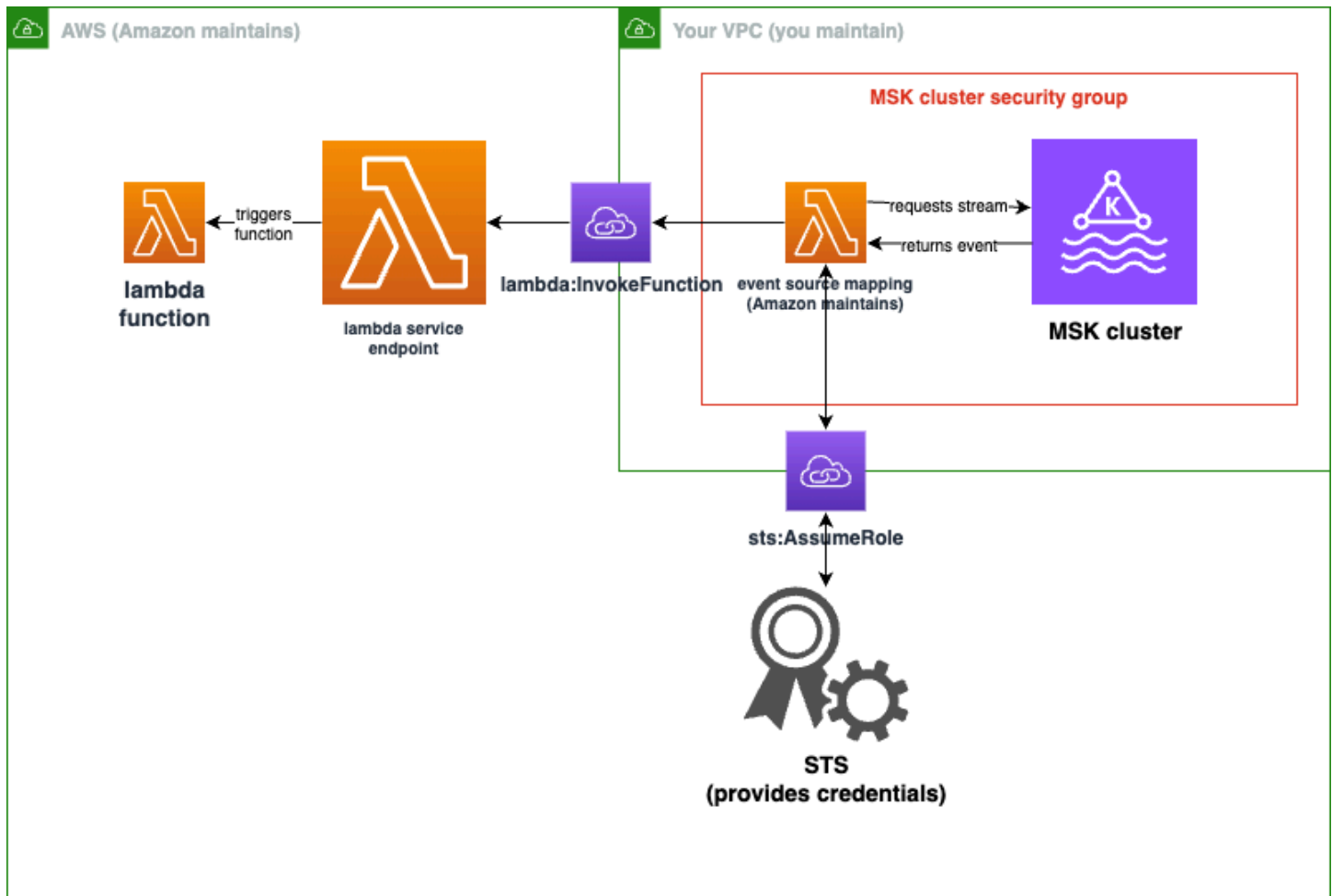
Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere una Windows-integrated versione di Ubuntu e Bash, [installa il sottosistema Windows per Linux](#).

Configurare la connettività di rete per consentire a Lambda di comunicare con Amazon MSK

AWS PrivateLink Usalo per connettere Lambda e Amazon MSK. È possibile farlo creando endpoint Amazon VPC di interfaccia nella console Amazon VPC. Per ulteriori informazioni sui problemi di configurazioni di rete, consulta [the section called “Configurazione del cluster e della rete”](#).

Quando una mappatura dell'origine degli eventi di Amazon MSK viene eseguita per conto di una funzione Lambda, assume il ruolo di esecuzione della funzione Lambda. Questo ruolo IAM autorizza la mappatura per accedere a risorse protette da IAM, come il cluster Amazon MSK. Sebbene i componenti condividano un ruolo di esecuzione, la mappatura Amazon MSK e la funzione Lambda

hanno requisiti di connettività separati per le rispettive attività, come illustrato nel diagramma seguente.



Lo strumento di mappatura dell'origine degli eventi appartiene al gruppo di sicurezza del cluster Amazon MSK. In questa fase di networking, crea endpoint Amazon VPC dal tuo VPC del cluster Amazon MSK per connettere lo strumento di mappatura dell'origine degli eventi ai servizi Lambda e STS. Proteggi questi endpoint per accettare il traffico proveniente dal tuo gruppo di sicurezza del cluster Amazon MSK. Quindi, modifica i gruppi di sicurezza del cluster Amazon MSK per consentire allo strumento di mappatura dell'origine degli eventi di comunicare con il cluster Amazon MSK.

Puoi configurare la procedura seguente utilizzando la Console di gestione AWS.

Per configurare gli endpoint Amazon VPC di interfaccia per connettere Lambda e Amazon MSK

1. Crea un gruppo di sicurezza per gli endpoint Amazon VPC della tua interfaccia **endpointSecurityGroup**, che consenta il traffico TCP in entrata su 443 da **clusterSecurityGroups**. Segui la procedura in [Creare un gruppo di sicurezza](#) nella



documentazione di Amazon EC2 per creare un gruppo di sicurezza. Quindi, segui la procedura in [Aggiungere regole a un gruppo di sicurezza](#) nella documentazione di Amazon EC2 per aggiungere le regole appropriate.

Crea un gruppo di sicurezza con le seguenti informazioni:

Quando aggiungi le regole in entrata, crea una regola per ogni gruppo di sicurezza in. *clusterSecurityGroups* Per ogni regola:

- Per Tipo, seleziona HTTPS.
  - Per Source, seleziona uno dei. *clusterSecurityGroups*
2. Crea un endpoint che connette il servizio Lambda all'Amazon VPC contenente il cluster Amazon MSK. Segui la procedura riportata in [Creare un endpoint di interfaccia](#).

Crea un endpoint di interfaccia con le seguenti informazioni:

- Per Nome del servizio, seleziona `com.amazonaws.regionName.lambda`, dove *regionName* ospita la tua funzione Lambda.
- Per VPC, seleziona l'Amazon VPC contenente il cluster Amazon MSK.
- Per i gruppi di sicurezza, seleziona *endpointSecurityGroup*, che hai creato in precedenza.
- Per Sottoreti, seleziona le sottoreti che ospitano il tuo cluster Amazon MSK.
- Per Policy, fornisci il seguente documento di policy, che protegge l'endpoint per l'utilizzo da parte del responsabile del servizio Lambda per l'azione `lambda:InvokeFunction`.

```
{
  "Statement": [
    {
      "Action": "lambda:InvokeFunction",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

- Assicurati Abilita nome DNS rimanga impostato.

3. Crea un endpoint che connette il AWS STS servizio all'Amazon VPC contenente il tuo cluster Amazon MSK. Segui la procedura riportata in [Creare un endpoint di interfaccia](#).

Crea un endpoint di interfaccia con le seguenti informazioni:

- Per Nome del servizio, seleziona. AWS STS
- Per VPC, seleziona l'Amazon VPC contenente il cluster Amazon MSK.
- Per i gruppi di sicurezza, seleziona *endpointSecurityGroup*.
- Per Sottoreti, seleziona le sottoreti che ospitano il tuo cluster Amazon MSK.
- Per Policy, fornisci il seguente documento di policy, che protegge l'endpoint per l'utilizzo da parte del responsabile del servizio Lambda per l'azione `sts:AssumeRole`.

```
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

- Assicurati Abilita nome DNS rimanga impostato.
4. Per ogni gruppo di sicurezza associato al tuo cluster Amazon MSK, ovvero in *clusterSecurityGroups*, consenti quanto segue:
    - Consenti tutto il traffico TCP in entrata e in uscita su 9098 a tutti, anche al suo interno. *clusterSecurityGroups*
    - Consenti tutto il traffico TCP in uscita su 443.

Parte di questo traffico è consentito dalle regole predefinite del gruppo di sicurezza, quindi se il cluster è collegato a un singolo gruppo di sicurezza e tale gruppo ha regole predefinite, non sono necessarie regole aggiuntive. Per ottimizzare le regole del gruppo di sicurezza, segui la procedura in [Aggiungere regole a un gruppo di sicurezza](#) nella documentazione di Amazon EC2.

Aggiungi le regole ai tuoi gruppi di sicurezza con le seguenti informazioni:

- Per ogni regola in entrata o in uscita per la porta 9098, fornisci
  - Per Type (Tipo) seleziona Custom TCP (TCP personalizzato).
  - Per Intervallo di porte, specifica 9098.
  - Per Source, fornisci uno dei. *clusterSecurityGroups*
- Per ogni regola in entrata per la porta 443, per Tipo, seleziona HTTPS.

Crea un ruolo IAM perché Lambda legga dal tuo argomento Amazon MSK

Identifica i requisiti di autenticazione perché Lambda legga dall'argomento di Amazon MSK, quindi definiscili in una policy. Crea un ruolo che autorizzi Lambda a utilizzare tali autorizzazioni. *LambdaAuthRole* Autorizza le azioni sul tuo cluster Amazon MSK utilizzando azioni kafka-cluster IAM. Quindi, autorizza Lambda a eseguire le azioni di Amazon MSK kafka e Amazon EC2 necessarie per scoprire e connettersi al tuo cluster Amazon MSK, nonché le azioni in modo che Lambda possa registrare ciò che ha CloudWatch fatto.

Per descrivere i requisiti di autenticazione perché Lambda legga da Amazon MSK

1. Scrivi un documento di policy IAM (un documento JSON) *clusterAuthPolicy*, che consenta a Lambda di leggere l'argomento di Kafka nel tuo cluster Amazon MSK utilizzando il tuo gruppo di consumatori Kafka. Lambda richiede che durante la lettura sia impostato un gruppo di consumer Kafka.

Modifica il seguente modello per allinearli ai tuoi prerequisiti:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kafka-cluster:Connect",
        "kafka-cluster:DescribeGroup",
        "kafka-cluster:AlterGroup",
        "kafka-cluster:DescribeTopic",
```

```

        "kafka-cluster:ReadData",
        "kafka-cluster:DescribeClusterDynamicConfiguration"
    ],
    "Resource": [
        "arn:aws:kafka:us-  
east-1:111122223333:cluster/mskClusterName/cluster-uuid",
        "arn:aws:kafka:us-  
east-1:111122223333:topic/mskClusterName/cluster-uuid/mskTopicName",
        "arn:aws:kafka:us-  
east-1:111122223333:group/mskClusterName/cluster-uuid/mskGroupName"
    ]
}

```

Per ulteriori informazioni, consulta [the section called “Configurazione delle autorizzazioni”](#).

Quando scrivi la tua policy:

- Sostituisci *us-east-1* e *111122223333* con la fine del Regione AWS tuo cluster Amazon MSK. Account AWS
  - Per *mskClusterName*, fornisci il nome del tuo cluster Amazon MSK.
  - Per *cluster-uuid*, fornisci l'UUID nell'ARN per il tuo cluster Amazon MSK.
  - Per *mskTopicName*, fornisci il nome del tuo argomento di Kafka.
  - Per *mskGroupName*, fornisci il nome del tuo gruppo di consumatori Kafka.
2. Identifica Amazon MSK, Amazon EC2 e CloudWatch le autorizzazioni necessarie a Lambda per rilevare e connettere il tuo cluster Amazon MSK e registrare tali eventi.

La policy gestita da `AWSLambdaMSKExecutionRole` definisce in modo permissivo le autorizzazioni richieste. Utilizzalo nelle fasi seguenti.

In un ambiente di produzione, valuta `AWSLambdaMSKExecutionRole` per limitare la policy del ruolo di esecuzione in base al principio del privilegio minimo, quindi scrivi una policy per il tuo ruolo che sostituisca questa policy gestita.

Per i dettagli sul linguaggio della policy IAM, consulta la [documentazione IAM](#).

Ora che hai scritto il tuo documento di policy, crea una policy IAM in modo da poterla collegare al tuo ruolo. È possibile effettuare tale operazione mediante la console utilizzando la seguente procedura.

Per creare una policy IAM dal documento della policy

1. Accedi Console di gestione AWS e apri la console IAM all'indirizzo. <https://console.aws.amazon.com/iam/>
2. Nel riquadro di navigazione a sinistra, seleziona Policies (Policy).
3. Scegli Crea policy.
4. Nella sezione Editor di policy, scegli l'opzione JSON.
5. Incolla *clusterAuthPolicy*.
6. Una volta terminata l'aggiunta delle autorizzazioni alla policy, scegli Successivo.
7. Nella pagina Verifica e crea, digita i valori per Nome policy e Descrizione (facoltativa) per la policy che si sta creando. Rivedi Autorizzazioni definite in questa policy per visualizzare le autorizzazioni concesse dalla policy.
8. Seleziona Crea policy per salvare la nuova policy.

Per ulteriori informazioni, consulta [Creazione di policy IAM](#) nella documentazione di IAM.

Ora che disponi delle policy IAM appropriate, crea un ruolo e associale ad esso. È possibile effettuare tale operazione mediante la console utilizzando la seguente procedura.

Per creare un ruolo di esecuzione nella console IAM

1. Aprire la pagina [Roles \(Ruoli\)](#) nella console IAM.
2. Scegliere Crea ruolo.
3. In Tipo di entità attendibile, scegli Servizio AWS .
4. In Use case (Caso d'uso), scegli Lambda.
5. Scegli Avanti.
6. Selezionare le seguenti policy:
  - *clusterAuthPolicy*
  - AWSLambdaMSKExecutionRole
7. Scegli Avanti.
8. Per il nome del ruolo, inserisci *lambdaAuthRole* e scegli Crea ruolo.

Per ulteriori informazioni, consulta [the section called “Ruolo di esecuzione \(autorizzazioni per le funzioni per accedere ad altre risorse\)”](#).

## Creare una funzione Lambda per leggere dal tuo argomento Amazon MSK

Crea una funzione Lambda configurata per utilizzare il tuo ruolo IAM. È possibile creare la funzione Lambda utilizzando la console.

Per creare una funzione Lambda utilizzando la tua configurazione di autenticazione

1. Apri la console Lambda e seleziona Crea funzione dall'interfaccia.
2. Scegli Crea da zero.
3. Per Nome della funzione, fornisci un nome appropriato a tua scelta.
4. Per Runtime, scegli l'ultima versione supportata di Node . js per utilizzare il codice fornito in questo tutorial.
5. Scegli Cambia ruolo di esecuzione predefinito.
6. Seleziona Utilizza un ruolo esistente.
7. Per Ruolo esistente, seleziona *lambdaAuthRole*.

In un ambiente di produzione, in genere è necessario aggiungere ulteriori policy al ruolo di esecuzione per la funzione Lambda per elaborare in modo significativo gli eventi Amazon MSK. Per ulteriori informazioni sull'aggiunta di policy al tuo ruolo, consulta [Aggiungere o rimuovere le autorizzazioni di identità](#) nella documentazione IAM.

## Creare uno strumento di mappatura dell'origine degli eventi sulla funzione Lambda

Lo strumento di mappatura dell'origine degli eventi Amazon MSK fornisce al servizio Lambda le informazioni necessarie per richiamare Lambda quando si verificano gli eventi Amazon MSK appropriati. Puoi creare una mappatura Amazon MSK utilizzando la console. Crea un trigger Lambda, quindi lo strumento di mappatura dell'origine degli eventi viene impostato automaticamente.

Per creare un trigger Lambda (e uno strumento di mappatura dell'origine degli eventi)

1. Vai alla pagina della panoramica della tua funzione Lambda.
2. Nella sezione della panoramica della funzione, scegli Aggiungi trigger in basso a sinistra.
3. Nel menu a discesa Seleziona un'origine, seleziona Amazon MSK.
4. Non impostare l'autenticazione.
5. Per Cluster MSK seleziona il nome del cluster.
6. Per Dimensioni del batch, immetti 1. Questo passaggio semplifica il test di questa funzionalità e non rappresenta un valore ideale nella produzione.

7. Per Nome argomento, fornisci il nome del tuo argomento Kafka.
8. Per l'ID del gruppo di consumer, fornisci l'ID del tuo gruppo di consumer Kafka.

Aggiornare la funzione Lambda per leggere i dati di streaming

Lambda fornisce informazioni sugli eventi di Kafka tramite il parametro event method. Per una struttura di esempio di un evento Amazon MSK, consulta [the section called “Esempio di evento”](#). Dopo aver capito come interpretare gli eventi Amazon MSK inoltrati da Lambda, puoi modificare il codice della funzione Lambda per utilizzare le informazioni fornite.

Fornisci il seguente codice alla tua funzione Lambda per registrare il contenuto di un evento Lambda Amazon MSK a scopo di test:

.NET

SDK per .NET

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite .NET.

```
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KafkaEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace MSKLambda;

public class Function
{

```

```

    /// <param name="input">The event for the Lambda function handler to
    process.</param>
    /// <param name="context">The ILambdaContext that provides methods for
    logging and describing the Lambda environment.</param>
    /// <returns></returns>
    public void FunctionHandler(KafkaEvent evnt, ILambdaContext context)
    {

        foreach (var record in evnt.Records)
        {
            Console.WriteLine("Key:" + record.Key);
            foreach (var eventRecord in record.Value)
            {
                var valueBytes = eventRecord.Value.ToArray();
                var valueText = Encoding.UTF8.GetString(valueBytes);

                Console.WriteLine("Message:" + valueText);
            }
        }
    }
}

```

## Go

### SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite Go.

```

package main

import (
    "encoding/base64"

```



```
"fmt"

"github.com/aws/aws-lambda-go/events"
"github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.KafkaEvent) {
    for key, records := range event.Records {
        fmt.Println("Key:", key)

        for _, record := range records {
            fmt.Println("Record:", record)

            decodedValue, _ := base64.StdEncoding.DecodeString(record.Value)
            message := string(decodedValue)
            fmt.Println("Message:", message)
        }
    }
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent.KafkaEventRecord;
```

```
import java.util.Base64;
import java.util.Map;

public class Example implements RequestHandler<KafkaEvent, Void> {

    @Override
    public Void handleRequest(KafkaEvent event, Context context) {
        for (Map.Entry<String, java.util.List<KafkaEventRecord>> entry :
event.getRecords().entrySet()) {
            String key = entry.getKey();
            System.out.println("Key: " + key);

            for (KafkaEventRecord record : entry.getValue()) {
                System.out.println("Record: " + record);

                byte[] value = Base64.getDecoder().decode(record.getValue());
                String message = new String(value);
                System.out.println("Message: " + message);
            }
        }

        return null;
    }
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Consumare un evento Amazon MSK utilizzando Lambda. JavaScript

```
exports.handler = async (event) => {
    // Iterate through keys
    for (let key in event.records) {
```

```
    console.log('Key: ', key)
    // Iterate through records
    event.records[key].map((record) => {
      console.log('Record: ', record)
      // Decode base64
      const msg = Buffer.from(record.value, 'base64').toString()
      console.log('Message:', msg)
    })
  }
}
```

## Consumare un evento Amazon MSK con Lambda utilizzando TypeScript

```
import { MSKEvent, Context } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "msk-handler-sample",
});

export const handler = async (
  event: MSKEvent,
  context: Context
): Promise<void> => {
  for (const [topic, topicRecords] of Object.entries(event.records)) {
    logger.info(`Processing key: ${topic}`);

    // Process each record in the partition
    for (const record of topicRecords) {
      try {
        // Decode the message value from base64
        const decodedMessage = Buffer.from(record.value, 'base64').toString();

        logger.info({
          message: decodedMessage
        });
      }
      catch (error) {
        logger.error('Error processing event', { error });
        throw error;
      }
    }
  }
}
```

```
};  
}  
}
```

## PHP

### SDK per PHP

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite PHP.

```
<?php  
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
  
// using bref/bref and bref/logger for simplicity  
  
use Bref\Context\Context;  
use Bref\Event\Kafka\KafkaEvent;  
use Bref\Event\Handler as StdHandler;  
use Bref\Logger\StderrLogger;  
  
require __DIR__ . '/vendor/autoload.php';  
  
class Handler implements StdHandler  
{  
    private StderrLogger $logger;  
    public function __construct(StderrLogger $logger)  
    {  
        $this->logger = $logger;  
    }  
  
    /**  
     * @throws JsonException  
     * @throws \Bref\Event\InvalidLambdaEvent  
     */
```

```
public function handle(mixed $event, Context $context): void
{
    $kafkaEvent = new KafkaEvent($event);
    $this->logger->info("Processing records");
    $records = $kafkaEvent->getRecords();

    foreach ($records as $record) {
        try {
            $key = $record->getKey();
            $this->logger->info("Key: $key");

            $values = $record->getValue();
            $this->logger->info(json_encode($values));

            foreach ($values as $value) {
                $this->logger->info("Value: $value");
            }

        } catch (Exception $e) {
            $this->logger->error($e->getMessage());
        }
    }
    $totalRecords = count($records);
    $this->logger->info("Successfully processed $totalRecords records");
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite Python.

```
import base64

def lambda_handler(event, context):
    # Iterate through keys
    for key in event['records']:
        print('Key:', key)
        # Iterate through records
        for record in event['records'][key]:
            print('Record:', record)
            # Decode base64
            msg = base64.b64decode(record['value']).decode('utf-8')
            print('Message:', msg)
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite Ruby.

```
require 'base64'

def lambda_handler(event:, context:)
    # Iterate through keys
    event['records'].each do |key, records|
        puts "Key: #{key}"

        # Iterate through records
        records.each do |record|
            puts "Record: #{record}"

            # Decode base64
            msg = Base64.decode64(record['value'])
            puts "Message: #{msg}"
```

```

    end
  end
end

```

## Rust

### SDK per Rust

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite Rust.

```

use aws_lambda_events::event::kafka::KafkaEvent;
use lambda_runtime::{run, service_fn, tracing, Error, LambdaEvent};
use base64::prelude::*;
use serde_json::{Value};
use tracing::{info};

/// Pre-Requisites:
/// 1. Install Cargo Lambda - see https://www.cargo-lambda.info/guide/getting-started.html
/// 2. Add packages tracing, tracing-subscriber, serde_json, base64
///
/// This is the main body for the function.
/// Write your code inside it.
/// There are some code example in the following URLs:
/// - https://github.com/aws-labs/aws-lambda-rust-runtime/tree/main/examples
/// - https://github.com/aws-samples/serverless-rust-demo/

async fn function_handler(event: LambdaEvent<KafkaEvent>) -> Result<Value, Error>
{
    let payload = event.payload.records;

    for (_name, records) in payload.iter() {

        for record in records {

```

```

        let record_text = record.value.as_ref().ok_or("Value is None")?;
        info!("Record: {}", &record_text);

        // perform Base64 decoding
        let record_bytes = BASE64_STANDARD.decode(record_text)?;
        let message = std::str::from_utf8(&record_bytes)?;

        info!("Message: {}", message);
    }

}

Ok(()).into())
}

#[tokio::main]
async fn main() -> Result<(), Error> {

    // required to enable CloudWatch error logging by the runtime
    tracing::init_default_subscriber();
    info!("Setup CW subscriber!");

    run(service_fn(function_handler)).await
}

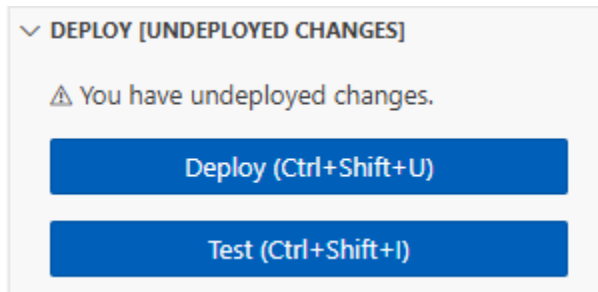
```

È possibile fornire il codice della funzione a Lambda utilizzando la console.

Aggiornamento del codice della funzione utilizzando l'editor di codice della console

1. Apri la [pagina Funzioni](#) della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, seleziona il tuo file di codice sorgente e modificalo nell'editor di codice integrato.
4. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:





Testa la tua funzione Lambda per verificare che sia connessa al tuo argomento Amazon MSK

Ora puoi verificare se la tua Lambda viene richiamata o meno dall'origine dell'evento CloudWatch ispezionando i registri degli eventi.

Per verificare se la funzione Lambda viene richiamata

1. Usa il tuo host di amministrazione Kafka per generare eventi Kafka utilizzando la CLI `kafka-console-producer`. Per ulteriori informazioni, consulta [Scrivere alcuni eventi nell'argomento](#) della documentazione di Kafka. Invia un numero sufficiente di eventi per riempire il batch definito in base alla dimensione del batch per la mappatura dell'origine degli eventi definita nel passaggio precedente, oppure Lambda attende che vengano richiamate ulteriori informazioni.
2. Se la funzione viene eseguita, Lambda scrive cosa è successo a. CloudWatch Nella console, passa alla pagina dei dettagli della funzione Lambda.
3. Selezionare la scheda Configurazione.
4. Dalla barra laterale, seleziona Strumenti di monitoraggio e operazioni.
5. Identifica il gruppo di CloudWatch log in Logging configuration. Il gruppo di log dovrebbe iniziare con `/aws/lambda`. Scegli il link del gruppo di log.
6. Nella CloudWatch console, controlla gli eventi di registro per individuare gli eventi di registro che Lambda ha inviato al flusso di log. Identifica se ci sono eventi del log contenenti il messaggio del tuo evento Kafka, come nell'immagine seguente. In tal caso, hai collegato correttamente una funzione Lambda ad Amazon MSK con uno strumento di mappatura dell'origine degli eventi Lambda.

|                               |                                                                                                                                                                                                                                   |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2020-08-06T15:06:18.861-04:00 | START RequestId: 88ebae59-be0c-4e22-9db7-4154b437e43a Version: \$LATEST                                                                                                                                                           |
| 2020-08-06T15:06:18.866-04:00 | 2020-08-06T19:06:18.866Z 88ebae59-be0c-4e22-9db7-4154b437e43a INFO Key: mytopic-0                                                                                                                                                 |
| 2020-08-06T15:06:18.866-04:00 | 2020-08-06T19:06:18.866Z 88ebae59-be0c-4e22-9db7-4154b437e43a INFO Record: {<br>topic: 'mytopic',<br>partition: 0,<br>offset: 38,<br>timestamp: 1596740777633,<br>timestampType: 'CREATE_TIME',<br>value: 'TwVzc2FnZSAjMQ=='<br>} |
| 2020-08-06T15:06:18.866-04:00 | 2020-08-06T19:06:18.866Z 88ebae59-be0c-4e22-9db7-4154b437e43a INFO Message: Message #1                                                                                                                                            |
| 2020-08-06T15:06:18.890-04:00 | END RequestId: 88ebae59-be0c-4e22-9db7-4154b437e43a                                                                                                                                                                               |

## Utilizzo di Lambda con Apache Kafka gestito dal cliente

Questo argomento descrive come utilizzare Lambda con un cluster Kafka autogestito. In AWS terminologia, un cluster autogestito include cluster Kafka non AWS ospitati. Ad esempio, è possibile ospitare il proprio cluster Kafka con un provider cloud come [Confluent Cloud](#) o [Redpanda](#).

Questo capitolo spiega come utilizzare un cluster Apache Kafka autogestito come origine di eventi per la funzione Lambda. Il processo generale per l'integrazione di Apache Kafka autogestito con Lambda prevede i seguenti passaggi:

1. [Configurazione del cluster e della rete](#): per prima cosa, configura il cluster Apache Kafka autogestito con la configurazione di rete corretta per consentire a Lambda di accedere al cluster.
2. [Configurazione della mappatura delle sorgenti degli eventi](#): quindi, crea la risorsa di [mappatura delle sorgenti degli eventi](#) di cui Lambda ha bisogno per connettere in modo sicuro il cluster Apache Kafka alla tua funzione.
3. [Configurazione delle funzioni e delle autorizzazioni](#): infine, assicurati che la funzione sia [configurata correttamente e disponga delle autorizzazioni necessarie nel suo ruolo di esecuzione](#).

Apache Kafka come origine eventi funziona in modo simile all'utilizzo di Amazon Simple Queue Service (Amazon SQS) o Amazon Kinesis. Lambda interroga internamente i nuovi messaggi dell'origine eventi, quindi richiama in modo sincrono la funzione Lambda di destinazione. Lambda legge i messaggi in batch e li fornisce alla funzione come payload di evento. La dimensione massima del batch è configurabile (l'impostazione predefinita è 100 messaggi). Per ulteriori informazioni, consulta [the section called “Comportamento di batching”](#).

Per ottimizzare il throughput dello strumento di mappatura dell'origine degli eventi di Apache Kafka autogestito, configura la modalità provisioning. In modalità provisioning, puoi definire il numero minimo e massimo di poller di eventi assegnati allo strumento di mappatura dell'origine degli eventi. Ciò può migliorare la capacità dello strumento di mappatura dell'origine degli eventi per gestire picchi di messaggi imprevisti. Per ulteriori informazioni, consulta [Modalità provisioning](#).

### Warning

Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per

ulteriori informazioni, consulta [Come posso rendere idempotente la mia funzione Lambda](#) nel Knowledge Center. AWS

Per le origini eventi basate su KAFKA, Lambda supporta i parametri di controllo dell'elaborazione, come le finestre di batch e le dimensioni del batch. Per ulteriori informazioni, consulta [Comportamento di batching](#).

Per un esempio di come utilizzare Kafka autogestito come fonte di eventi, vedi [Utilizzo di Apache Kafka ospitato autonomamente come fonte di eventi per sul blog di Compute](#). AWS Lambda AWS

## Argomenti

- [Esempio di evento](#)
- [Configurazione del cluster e della rete Apache Kafka autogestiti per Lambda](#)
- [Configurazione delle autorizzazioni del ruolo di esecuzione Lambda](#)
- [Configurazione di origini eventi Apache Kafka autogestito per Lambda](#)

## Esempio di evento

Lambda invia il batch di messaggi nel parametro evento quando richiama la funzione Lambda. Il payload evento contiene un array di messaggi. Ogni elemento dell'array contiene i dettagli dell'argomento Kafka e dell'identificatore dello shard Kafka, insieme a una data/ora e a un messaggio con codifica base64.

```
{
  "eventSource": "SelfManagedKafka",
  "bootstrapServers": "b-2.demo-cluster-1.a1bcde.c1.kafka.us-east-1.amazonaws.com:9092,b-1.demo-cluster-1.a1bcde.c1.kafka.us-east-1.amazonaws.com:9092",
  "records": {
    "mytopic-0": [
      {
        "topic": "mytopic",
        "partition": 0,
        "offset": 15,
        "timestamp": 1545084650987,
        "timestampType": "CREATE_TIME",
        "key": "abcDEFghiJKLmnoPQRstuVWXYZ1234==",
        "value": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg=="
      }
    ]
  }
}
```

```
"headers": [
  {
    "headerKey": [
      104,
      101,
      97,
      100,
      101,
      114,
      86,
      97,
      108,
      117,
      101
    ]
  }
]
```

## Configurazione del cluster e della rete Apache Kafka autogestiti per Lambda

Per connettere la funzione Lambda al cluster Apache Kafka autogestito, devi configurare correttamente il cluster e la rete in cui risiede. Questa pagina descrive come configurare il cluster e la rete. Se il cluster e la rete sono già configurati correttamente, consulta [the section called “Configurare un'origine eventi”](#) per configurare la mappatura dell'origine degli eventi.

### Argomenti

- [Self-managed Configurazione del cluster Apache Kafka](#)
- [Configurare la sicurezza della rete](#)

### Self-managed Configurazione del cluster Apache Kafka

Puoi ospitare il tuo cluster Apache Kafka autogestito con provider cloud come [Confluent Cloud](#) o <https://www.redpanda.com/> Redpanda oppure eseguirlo sulla tua infrastruttura. Assicurati che il cluster sia configurato correttamente e accessibile dalla rete a cui si connette la mappatura delle sorgenti degli eventi Lambda.

## Configurare la sicurezza della rete

Per concedere a Lambda l'accesso completo ad Apache Kafka autogestito tramite lo strumento di mappatura dell'origine degli eventi, il cluster deve utilizzare un endpoint pubblico (indirizzo IP pubblico) oppure devi fornire l'accesso all'Amazon VPC in cui hai creato il cluster.

Quando usi Apache Kafka autogestito con Lambda, crea [endpoint VPC AWS PrivateLink](#) che forniscono alla funzione l'accesso alle risorse del tuo Amazon VPC.

### Note

**AWS PrivateLink** Gli endpoint VPC sono necessari per le funzioni con mappature delle sorgenti di eventi che utilizzano la modalità predefinita (su richiesta) per i poller di eventi. Se la mappatura dell'origine degli eventi utilizza la modalità [provisioned](#), non è necessario configurare gli endpoint VPC. AWS PrivateLink

Crea un endpoint per fornire l'accesso alle seguenti risorse:

- Lambda: crea un endpoint per il principale del servizio Lambda.
- AWS STS — Crea un endpoint per consentire AWS STS a un responsabile del servizio di assumere un ruolo per tuo conto.
- Secrets Manager: se il tuo cluster utilizza Secrets Manager per archiviare le credenziali, crea un endpoint per Secrets Manager.

In alternativa, configura un gateway NAT su ogni sottorete pubblica in Amazon VPC. Per ulteriori informazioni, consulta [the section called “Accesso Internet per funzioni del VPC”](#).

Quando crei uno strumento di mappatura dell'origine degli eventi per Apache Kafka autogestito, Lambda verifica se le interfacce di rete elastiche (ENI) sono già presenti per le sottoreti e i gruppi di sicurezza configurati per il tuo Amazon VPC. Se Lambda trova ENI esistenti, prova a riutilizzarle. Altrimenti, Lambda crea nuove ENI per connettersi all'origine dell'evento e richiamare la tua funzione.

### Note

Le funzioni Lambda vengono sempre eseguite all'interno di VPC di proprietà del servizio Lambda. La configurazione VPC della funzione non influisce sullo strumento di mappatura

dell'origine degli eventi. Solo la configurazione di rete dell'origine dell'evento determina il modo in cui Lambda si connette all'origine dell'evento.

Configura i gruppi di sicurezza per l'Amazon VPC contenente il tuo cluster. Per impostazione predefinita, Apache Kafka autogestito utilizza le seguenti porte: 9092.

- Regole in ingresso: consenti tutto il traffico sulla porta del broker predefinita per il gruppo di sicurezza associato all'origine eventi. In alternativa, puoi utilizzare una regola autoreferenziale del gruppo di sicurezza per consentire l'accesso da istanze all'interno dello stesso gruppo di sicurezza.
- Regole in uscita: consenti tutto il traffico sulla porta 443 per destinazioni esterne se la tua funzione deve comunicare con i servizi. AWS In alternativa, puoi anche utilizzare una regola di gruppo di sicurezza autoreferenziale per limitare l'accesso al broker se non hai bisogno di comunicare con altri servizi. AWS
- Regole di ingresso degli endpoint Amazon VPC: se utilizzi un endpoint Amazon VPC, il gruppo di sicurezza associato all'endpoint Amazon VPC deve consentire il traffico in entrata sulla porta 443 dal gruppo di sicurezza del cluster.

Se il cluster utilizza l'autenticazione, puoi anche limitare la policy degli endpoint per l'endpoint Secrets Manager. Per chiamare l'API Secrets Manager, Lambda utilizza il ruolo della funzione, non il principale del servizio Lambda.

Example Policy dell'endpoint VPC: endpoint Secrets Manager

```
{
  "Statement": [
    {
      "Action": "secretsmanager:GetSecretValue",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws::iam::123456789012:role/my-role"
        ]
      },
      "Resource": "arn:aws::secretsmanager:us-west-2:123456789012:secret:my-secret"
    }
  ]
}
```

Quando utilizzi gli endpoint Amazon VPC, AWS indirizza le tue chiamate API per richiamare la tua funzione utilizzando l'Elastic Network Interface (ENI) dell'endpoint. Il principale del servizio Lambda deve chiamare `lambda:InvokeFunction` su tutti i ruoli e le funzioni che utilizzano tali ENI.

Per impostazione predefinita, gli endpoint Amazon VPC dispongono di policy IAM aperte che consentono un ampio accesso alle risorse. La best practice consiste nel limitare queste policy per eseguire le azioni necessarie utilizzando quell'endpoint. Per garantire che lo strumento di mappatura dell'origine degli eventi sia in grado di invocare la funzione Lambda, la policy degli endpoint VPC deve consentire al principale del servizio Lambda di chiamare `sts:AssumeRole` e `lambda:InvokeFunction`. Limitare le policy degli endpoint VPC per consentire solo le chiamate API provenienti dall'organizzazione impedisce il corretto funzionamento dello strumento di mappatura dell'origine degli eventi, pertanto `"Resource": "*"` è richiesto in queste policy.

Il seguente esempio di policy degli endpoint VPC mostra come concedere l'accesso richiesto al principale del servizio Lambda per gli endpoint AWS STS e Lambda.

#### Example Politica VPC Endpoint — AWS STS endpoint

```
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

#### Example Policy dell'endpoint VPC: endpoint Lambda

```
{
  "Statement": [
    {
      "Action": "lambda:InvokeFunction",
      "Effect": "Allow",
      "Principal": {
        "Service": [
```



```
        "lambda.amazonaws.com"
    ],
    },
    "Resource": "*"
}
]
```

## Configurazione delle autorizzazioni del ruolo di esecuzione Lambda

Oltre ad [accedere al cluster Kafka autogestito](#), la funzione Lambda necessita delle autorizzazioni per eseguire varie azioni API. Aggiungere queste autorizzazioni al [ruolo di esecuzione](#) della funzione. Se i tuoi utenti devono accedere a qualsiasi azione API, aggiungi le autorizzazioni richieste alla politica di identità per l'utente o il ruolo AWS Identity and Access Management (IAM).

### Argomenti

- [Autorizzazioni necessarie per la funzione Lambda](#)
- [Autorizzazioni facoltative per la funzione Lambda](#)
- [Aggiunta di autorizzazioni al ruolo di esecuzione](#)
- [Concessione di accesso agli utenti con una policy IAM](#)

## Autorizzazioni necessarie per la funzione Lambda

Per creare e archiviare i log in un gruppo di log in Amazon CloudWatch Logs, la funzione Lambda deve disporre delle seguenti autorizzazioni nel ruolo di esecuzione:

- [registri: CreateLogGroup](#)
- [registri: CreateLogStream](#)
- [registri: PutLogEvents](#)

## Autorizzazioni facoltative per la funzione Lambda

La funzione Lambda potrebbe richiedere autorizzazioni per:

- Descrivere il segreto di Secrets Manager.
- Accedi alla tua chiave AWS Key Management Service (AWS KMS) gestita dal cliente.
- Accedere ad Amazon VPC.
- Invia i record delle chiamate non riuscite a una destinazione.

## Secrets Manager e AWS KMS autorizzazioni

A seconda del tipo di controllo degli accessi che stai configurando per i tuoi broker Kafka, la tua funzione Lambda potrebbe richiedere l'autorizzazione per accedere al tuo segreto di Secrets Manager o per decrittografare la tua chiave gestita dal cliente. AWS KMS Per accedere a queste risorse, il ruolo di esecuzione della funzione deve disporre delle seguenti autorizzazioni:

- [gestore dei segreti: GetSecretValue](#)
- [kms:Decrypt](#)

## Autorizzazioni VPC

Se soltanto gli utenti all'interno di un VPC possono accedere al cluster Apache Kafka autogestito, la funzione Lambda deve disporre dell'autorizzazione per accedere alle risorse di Amazon VPC. Queste risorse includono la VPC, le sottoreti, i gruppi di sicurezza e le interfacce di rete. Per accedere a queste risorse, il ruolo di esecuzione della funzione deve disporre delle seguenti autorizzazioni:

- [ec2: CreateNetworkInterface](#)
- [ec2: DescribeNetworkInterfaces](#)
- [ec2: DescribeVpcs](#)
- [ec2: DeleteNetworkInterface](#)
- [ec2: DescribeSubnets](#)
- [ec2: DescribeSecurityGroups](#)

## Aggiunta di autorizzazioni al ruolo di esecuzione

[Per accedere ad altri AWS servizi utilizzati dal cluster Apache Kafka autogestito, Lambda utilizza le politiche di autorizzazione definite nel ruolo di esecuzione della funzione Lambda.](#)

Per impostazione predefinita, Lambda non è autorizzato a eseguire le operazioni richieste o facoltative per un cluster Apache Kafka autogestito. Dovrai creare e definire queste operazioni in una [policy di attendibilità IAM](#) per il ruolo di esecuzione. Questo esempio mostra come creare una policy che consente a Lambda di accedere alle risorse Amazon VPC.

## JSON

```
{
```

```
"Version":"2012-10-17",
"Statement":[
  {
    "Effect":"Allow",
    "Action":[
      "ec2:CreateNetworkInterface",
      "ec2:DescribeNetworkInterfaces",
      "ec2:DescribeVpcs",
      "ec2>DeleteNetworkInterface",
      "ec2:DescribeSubnets",
      "ec2:DescribeSecurityGroups"
    ],
    "Resource":"*"
  }
]
```

## Concessione di accesso agli utenti con una policy IAM

Per impostazione predefinita, gli utenti e i ruoli non dispongono dell'autorizzazione per eseguire [operazioni API di origine eventi](#). Per concedere l'accesso agli utenti dell'organizzazione o dell'account, è possibile creare o aggiornare una policy basata sull'identità. Per ulteriori informazioni, consulta [Controlling access to AWS resources using](#) policies nella IAM User Guide.

Per la risoluzione degli errori di autenticazione e autorizzazione, consulta [the section called “Risoluzione dei problemi”](#).

## Configurazione di origini eventi Apache Kafka autogestito per Lambda

Per utilizzare un cluster Apache Kafka autogestito come origine di eventi per la funzione Lambda, è necessario creare una [mappatura delle sorgenti degli eventi](#) che collega le due risorse. Questa pagina descrive come creare una mappatura delle sorgenti degli eventi per Apache Kafka autogestito.

Questa pagina presuppone che tu abbia già configurato correttamente il tuo cluster Kafka e la rete in cui risiede. Se devi configurare il tuo cluster o la tua rete, vedi. [the section called “Configurazione del cluster e della rete”](#)

## Argomenti

- [Utilizzo di un cluster Apache Kafka autogestito come fonte di eventi](#)
- [Configurazione dei metodi di autenticazione del cluster in Lambda](#)

- [Creazione di una mappatura della sorgente di eventi Lambda per un'origine di eventi Apache Kafka autogestita](#)
- [Tutti i parametri di configurazione dell'origine degli eventi Apache Kafka autogestiti in Lambda](#)

Utilizzo di un cluster Apache Kafka autogestito come fonte di eventi

Quando aggiungi il cluster Apache Kafka o Amazon MSK come trigger per la funzione Lambda, il cluster viene utilizzato come [origine eventi](#).

Lambda legge i dati degli eventi dagli argomenti di Kafka specificati Topics in una [CreateEventSourceMapping](#) richiesta, in base alla posizione [iniziale](#) specificata. Dopo che l'elaborazione è avvenuta con successo, l'argomento Kafka viene salvato nel cluster Kafka.

Lambda legge i messaggi in sequenza per ogni partizione dell'argomento Kafka. Un singolo payload Lambda può contenere messaggi provenienti da più partizioni. Quando sono disponibili più record, Lambda continua a elaborare i record in batch, in base al BatchSize valore specificato in una [CreateEventSourceMapping](#) richiesta, finché la funzione non raggiunge l'argomento.

Dopo che Lambda ha elaborato ogni batch, esegue il commit degli offset dei messaggi in quel batch. Se la funzione restituisce un errore per uno qualsiasi dei messaggi di un batch, Lambda ritenta l'intero batch di messaggi fino a quando l'elaborazione non riesce o i messaggi scadono. È possibile inviare i record per i quali tutti i nuovi tentativi falliscono a una destinazione in errore per un'elaborazione successiva.

#### Note

Anche se le funzioni Lambda generalmente prevedono un timeout massimo di 15 minuti, gli strumenti di mappatura dell'origine degli eventi per Amazon MSK, Apache Kafka autogestito, Amazon DocumentDB e Amazon MQ per ActiveMQ e RabbitMQ supportano solo funzioni con timeout massimi di 14 minuti.

Configurazione dei metodi di autenticazione del cluster in Lambda

Lambda supporta diversi metodi per l'autenticazione al cluster Apache Kafka autogestito. Assicurarsi di configurare il cluster Kafka in modo da utilizzare uno dei seguenti metodi di autenticazione supportati. Per ulteriori informazioni sulla sicurezza con Kafka, consultare la sezione [Sicurezza](#) della documentazione di Kafka.

## Autenticazione SASL/SCRAM

Lambda supporta l'autenticazione SASL/SCRAM (Simple Authentication and Security Layer/Salted Challenge Response Authentication Mechanism) con crittografia Transport Layer Security (TLS) (). SASL\_SSL Lambda invia le credenziali crittografate per l'autenticazione con il cluster. Lambda non supporta il testo in SASL/SCRAM chiaro (). SASL\_PLAINTEXT [Per ulteriori informazioni sull' SASL/SCRAM autenticazione, consulta RFC 5802.](#)

Lambda supporta SASL/PLAIN anche l'autenticazione. Poiché questo meccanismo utilizza credenziali in chiaro, la connessione al server deve utilizzare la crittografia TLS per garantire che le credenziali siano protette.

Per l'autenticazione SASL, è necessario archiviare le credenziali di accesso come segreto in AWS Secrets Manager. Per ulteriori informazioni sull'utilizzo di Secrets Manager, consulta [Create an AWS Secrets Manager secret](#) nella Guida AWS Secrets Manager per l'utente.

### Important

Per utilizzare Secrets Manager per l'autenticazione, i segreti devono essere archiviati nella stessa AWS area della funzione Lambda.

## Autenticazione TLS reciproca

MTLS (Mutual TLS) fornisce l'autenticazione bidirezionale tra client e server. Il client invia un certificato al server affinché il server verifichi il client e il server invia un certificato al client affinché il client verifichi il server.

In Apache Kafka autogestito Lambda agisce come client. È possibile configurare un certificato client (come segreto in Secrets Manager) per autenticare Lambda con i broker Kafka. Il certificato client deve essere firmato da una CA nell'archivio trust del server.

Il cluster Kafka invia un certificato server a Lambda per autenticare i broker con Lambda. Il certificato del server può essere un certificato CA pubblico o un certificato con CA/self-signed certificate. The public CA certificate must be signed by a certificate authority (CA) that's in the Lambda trust store. For a private CA/self firma privata, è possibile configurare il certificato CA root del server (come segreto in Secrets Manager). Lambda utilizza il certificato root per verificare i broker Kafka.

Per ulteriori informazioni su mTLS, consultare [Introduzione dell'autenticazione TLS reciproca per Amazon MSK come origine eventi.](#)

## Configurazione del segreto del certificato client

Il segreto `CLIENT_CERTIFICATE_TLS_AUTH` richiede un campo certificato e un campo chiave privata. Per una chiave privata crittografata, il segreto richiede una password per chiave privata. Il certificato e la chiave privata devono essere in formato PEM.

### Note

Lambda supporta gli algoritmi di crittografia a chiave privata [PBES1](#) (ma non PBES2).

Il campo certificato deve contenere un elenco di certificati, a partire dal certificato client, seguito da qualsiasi certificato intermedio, per finire con il certificato root. Ogni certificato deve iniziare su una nuova riga con la struttura seguente:

```
-----BEGIN CERTIFICATE-----
    <certificate contents>
-----END CERTIFICATE-----
```

Secrets Manager supporta segreti fino a 65.536 byte, che è uno spazio sufficiente per lunghe catene di certificati.

La chiave privata deve essere in formato [PKCS #8](#), con la struttura seguente:

```
-----BEGIN PRIVATE KEY-----
    <private key contents>
-----END PRIVATE KEY-----
```

Per una chiave privata crittografata, utilizza la struttura seguente:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
    <private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

Nell'esempio seguente viene mostrato il contenuto di un segreto per l'autenticazione mTLS utilizzando una chiave privata crittografata. Per una chiave privata crittografata, includere la password per chiave privata nel segreto.

```
{"privateKeyPassword":"testpassword",
```

```

"certificate": "-----BEGIN CERTIFICATE-----
MIIE5DCCAsygAwIBAgIRAPJdwaFaNRrytHBto0j5BA0wDQYJKoZIhvcNAQELBQAw
...
j0Lh4/+1HfgyE2KlmII36dg4IMzNjAFEBZiCRoPim040s1cRqtFHxoa10QQbIlxk
cmUuiAii9R0=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFGjCCA2qgAwIBAgIQdjNzd6uFf9hbNC5RdfmHrzANBgkqhkiG9w0BAQsFADBb
...
rQoiowbbk5wXCheYSANQIfTZ6weQTgiCHCCbuuMKNVS95FkXm0vqVD/YpXKwA/no
c8PH3PSOAaRwMMg0SA2ALJvbRz8mpg==
-----END CERTIFICATE-----",
"privateKey": "-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFKzBVBgkqhkiG9w0BBQ0wSDANBgkqhkiG9w0BBQwwGgQUiAFcK5hT/X7Kjmgp
...
QrSekqF+kWzmB6nAfSzg09IaoAaytLvNgGTckWeUkWn/V0Ck+LdGUXzAC4RxZnoQ
zp2mwJn2NYB7AZ7+imp0azDZb+8YG2aUCiyqb6PnnA==
-----END ENCRYPTED PRIVATE KEY-----"
}

```

## Configurazione del segreto del certificato CA root del server

Questo segreto viene creato se i broker Kafka utilizzano la crittografia TLS con certificati firmati da una CA privata. Puoi utilizzare la crittografia TLS per l'autenticazione VPC SASL/SCRAM, SASL/PLAIN o MTLS.

Il segreto del certificato CA root del server richiede un campo che contiene il certificato CA root del broker Kafka in formato PEM. Il seguente esempio illustra la struttura del segreto.

```

{"certificate": "-----BEGIN CERTIFICATE-----
MIID7zCCAtegAwIBAgIBADANBgkqhkiG9w0BAQsFADCBmDELMakGA1UEBhMCVVMx
EDA0BgNVBAgTB0FyaXpvbmExEzARBgNVBACTC1Njb3R0c2RhbGUxJTAjBgNVBAoT
HFN0YXJmaWVsZCBUZWNobm9sb2dpZXMxIEluYy4x0zA5BgNVBAMTMlN0YXJmaWVs
ZCBTZXJ2aWNlcyBSb290IENlcnRpZmljYXR1IEF1dG...
-----END CERTIFICATE-----"
}

```

Creazione di una mappatura della sorgente di eventi Lambda per un'origine di eventi Apache Kafka autogestita

[Per creare una mappatura delle sorgenti degli eventi, puoi utilizzare la console Lambda, la AWS Command Line Interface \(CLI\) o un SDK.](#)

I seguenti passaggi della console aggiungono un cluster Apache Kafka autogestito come trigger per la funzione Lambda. Sotto il cofano, questo crea una risorsa di mappatura delle sorgenti degli eventi.

## Prerequisiti

- Un cluster Apache Kafka autogestito. Lambda supporta la versione 0.10.1.0 e successive di Apache Kafka.
- Un [ruolo di esecuzione](#) con autorizzazione ad accedere alle AWS risorse utilizzate dal cluster Kafka autogestito.

## Aggiunta di un cluster Kafka autogestito (console)

Segui questi passaggi per aggiungere il cluster Apache Kafka autogestito e un argomento Kafka come trigger per la funzione Lambda.

Per aggiungere un trigger Apache Kafka alla funzione Lambda (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere il nome della funzione Lambda.
3. In Panoramica delle funzioni, scegliere Aggiungi trigger.
4. In Configurazione trigger, effettua le operazioni seguenti:
  - a. Scegliere il tipo di trigger Apache Kafka.
  - b. Per Server di bootstrap, inserisci l'indirizzo composto dalla coppia host e porta di un broker Kafka nel cluster, quindi scegli Aggiungi. Ripeti la procedura per ogni broker Kafka del cluster.
  - c. Per Nome argomento, inserisci il nome dell'argomento Kafka utilizzato per memorizzare i record nel cluster.
  - d. Se configuri la modalità provisioned, inserisci un valore per Minimum event poller, un valore per Maximum event poller e un valore opzionale per PollerGroupName specificare il raggruppamento di più eventi ESMs all'interno dello stesso VPC di origine eventi.
  - e. (Facoltativo) In Dimensioni batch, inserisci il numero massimo di record da ricevere in un singolo batch.
  - f. Per Finestra batch, immetti il tempo massimo in secondi per la raccolta dei registri da parte di Lambda prima di richiamare la funzione.



- g. (Facoltativo) Per ID gruppo di consumer, inserisci l'ID di un gruppo di consumer Kafka a cui aderire.
- h. (Facoltativo) Per Posizione di inizio, scegli Più recente per iniziare a leggere il flusso dal record più recente, Orizzonte di taglio per iniziare dal primo record disponibile o In corrispondenza del timestamp per specificare un timestamp da cui iniziare la lettura.
- i. (Facoltativo) Per VPC, scegli Amazon VPC per il cluster Kafka. Quindi, scegli Sottoreti VPC e Gruppi di sicurezza VPC.

Questa impostazione è necessaria soltanto se gli utenti del VPC accedono ai broker.

- j. (Facoltativo) Per Autenticazione, scegli Aggiungi e quindi esegui le seguenti operazioni:
  - i. Scegli il protocollo di accesso o di autenticazione dei broker Kafka del cluster.
    - Se il tuo broker Kafka utilizza l'autenticazione, scegli BASIC\_AUTH. SASL/PLAIN
    - Se il tuo broker utilizza SASL/SCRAM l'autenticazione, scegli uno dei protocolli. SASL\_SCRAM
    - Se stai configurando l'autenticazione mTLS, scegli il protocollo CLIENT\_CERTIFICATE\_TLS\_AUTH.
  - ii. Per SASL/SCRAM l'autenticazione MTLS, scegli la chiave segreta Secrets Manager che contiene le credenziali per il tuo cluster Kafka.
- k. (Facoltativo) Per Crittografia, scegli il segreto di Secrets Manager contenente il certificato CA root utilizzato dai broker Kafka per la crittografia TLS, se i broker Kafka utilizzano certificati firmati da una CA privata.

Questa impostazione si applica alla crittografia TLS per SASL/PLAIN e all' SASL/SCRAM autenticazione MTLS.

- l. Per creare il trigger in uno stato disabilitato per il test (scelta consigliata), deselezionare Abilita trigger. Oppure, per attivare immediatamente il trigger, selezionare Abilita trigger.

5. Per creare il trigger, scegli Aggiungi.

## Aggiunta di un cluster Kafka autogestito (AWS CLI)

Usa i seguenti AWS CLI comandi di esempio per creare e visualizzare un trigger Apache Kafka autogestito per la tua funzione Lambda.

## Utilizzo di SASL/SCRAM

Se gli utenti Kafka accedono ai tuoi broker Kafka tramite Internet, specifica il segreto Secrets Manager che hai creato per l'autenticazione. SASL/SCRAM L'esempio seguente utilizza il [create-event-source-mapping](#) AWS CLI comando per mappare una funzione Lambda denominata `my-kafka-function` a un argomento di Kafka denominato `AWSKafkaTopic`

```
aws lambda create-event-source-mapping \
  --topics AWSKafkaTopic \
  --source-access-configuration Type=SASL_SCRAM_512_AUTH,URI=arn:aws:secretsmanager:us-east-1:111122223333:secret:MyBrokerSecretName \
  --function-name arn:aws:lambda:us-east-1:111122223333:function:my-kafka-function \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":  
["abc3.xyz.com:9092", "abc2.xyz.com:9092"]}}'
```

## Utilizzo di un VPC

Se solo gli utenti Kafka all'interno del proprio VPC accedono ai broker Kafka, dovrai specificare VPC, sottorete e gruppo di sicurezza del VPC. L'esempio seguente utilizza il [create-event-source-mapping](#) AWS CLI comando per mappare una funzione Lambda denominata `my-kafka-function` a un argomento di Kafka denominato `AWSKafkaTopic`

```
aws lambda create-event-source-mapping \
  --topics AWSKafkaTopic \
  --source-access-configuration '[{"Type": "VPC_SUBNET", "URI":  
"subnet:subnet-0011001100"}, {"Type": "VPC_SUBNET", "URI":  
"subnet:subnet-0022002200"}, {"Type": "VPC_SECURITY_GROUP", "URI":  
"security_group:sg-0123456789"}]' \
  --function-name arn:aws:lambda:us-east-1:111122223333:function:my-kafka-function \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":  
["abc3.xyz.com:9092", "abc2.xyz.com:9092"]}}'
```

## Visualizzazione dello stato utilizzando il AWS CLI

L'esempio seguente utilizza il [get-event-source-mapping](#) AWS CLI comando per descrivere lo stato della mappatura dell'origine degli eventi creata.

```
aws lambda get-event-source-mapping
  --uuid dh38738e-992b-343a-1077-3478934hjkfd7
```

Tutti i parametri di configurazione dell'origine degli eventi Apache Kafka autogestiti in Lambda

Tutti i tipi di sorgenti di eventi Lambda condividono le stesse operazioni [CreateEventSourceMapping](#) quelle dell'[UpdateEventSourceMapping](#) API. Tuttavia, solo alcuni parametri si applicano ad Apache Kafka autogestito, come illustrato nella tabella seguente.

| Parametro                      | Obbligatorio | Predefinito                                                                                                                                                                  | Note                                                                    |
|--------------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| BatchSize                      | N            | 100                                                                                                                                                                          | Massimo: 10.000.                                                        |
| DestinationConfig              | N            | N/D                                                                                                                                                                          | <a href="#">the section called “Conserva le chiamate non riuscite”</a>  |
| Abilitato                      | N            | True                                                                                                                                                                         |                                                                         |
| FilterCriteria                 | N            | N/D                                                                                                                                                                          | <a href="#">Controllare gli eventi che Lambda invia alla funzione</a>   |
| FunctionName                   | Y            | N/D                                                                                                                                                                          |                                                                         |
| KMSKeyArn                      | N            | N/D                                                                                                                                                                          | <a href="#">the section called “Crittografia dei criteri di filtro”</a> |
| MaximumBatchingWindowInSeconds | N            | 500 ms                                                                                                                                                                       | <a href="#">Comportamento di batching</a>                               |
| ProvisionedPollersConfig       | N            | MinimumPollers :<br>se non specificato, il valore predefinito è 1<br><br>MaximumPollers :<br>se non specificato, il valore predefinito è 200<br><br>PollerGroupName :<br>N/A | <a href="#">the section called “Modalità provisioning”</a>              |

| Parametro                         | Obbligatorio | Predefinito                                                                             | Note                                                                                                             |
|-----------------------------------|--------------|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| SelfManagedEventSource            | Y            | N/D                                                                                     | Elenco dei broker Kafka. Può essere impostato solo su Create                                                     |
| SelfManagedKafkaEventSourceConfig | N            | Contiene il ConsumerGroupId campo che per impostazione predefinita è un valore univoco. | Può essere impostato solo su Create                                                                              |
| SourceAccessConfigurations        | N            | Nessuna credenziale                                                                     | Informazioni sul VPC o credenziali di autenticazione per il cluster<br><br>Per SASL_PLAIN, imposta su BASIC_AUTH |
| StartingPosition                  | Y            | N/D                                                                                     | AT_TIMESTAMP, TRIM_HORIZON o LATEST<br><br>Può essere impostato solo su Create                                   |
| StartingPositionTimestamp         | N            | N/D                                                                                     | Obbligatorio se StartingPosition è impostato su AT_TIMESTAMP                                                     |

| Parametro | Obbligatorio | Predefinito | Note                                                                                            |
|-----------|--------------|-------------|-------------------------------------------------------------------------------------------------|
| Tag       | N            | N/D         | <a href="#">the section called “Tag dello strumento di mappatura dell'origine degli eventi”</a> |
| Argomenti | Y            | N/D         | Nome dell'argomento<br><br>Può essere impostato solo su Create                                  |

#### Note

Quando si specifica una `PollerGroupName`, più unità ESMs all'interno dello stesso Amazon VPC possono condividere la capacità dell'Event Poller Unit (EPU). Puoi utilizzare questa opzione per ottimizzare i costi della modalità Provisioned per i tuoi ESMs. Requisiti per il raggruppamento ESM:

- ESMs deve trovarsi all'interno dello stesso Amazon VPC
- Massimo 100 ESMs per gruppo di sondaggi
- Il numero massimo aggregato di sondaggi di tutti ESMs i membri di un gruppo non può superare i 2000

È possibile aggiornare il file `PollerGroupName` per spostare un ESM in un gruppo diverso oppure rimuovere un ESM da un gruppo `PollerGroupName` impostandolo su una stringa vuota («»).

## Modalità di ridimensionamento del poller di eventi di Apache Kafka in Lambda

Puoi scegliere tra due modalità di scalabilità degli event poller per Amazon MSK e le mappature delle sorgenti di eventi Apache Kafka autogestite:

- [On-demand modalità \(predefinita\)](#)
- [Modalità provisioning](#)

### On-demand modalità (predefinita)

Quando si crea inizialmente la sorgente di eventi Kafka, Lambda assegna un numero predefinito di poller di eventi per elaborare tutte le partizioni nell'argomento Kafka. Lambda aumenta o diminuisce automaticamente il numero di [poller di eventi](#) in base al carico di messaggi.

In un intervallo di un minuto, Lambda valuta il ritardo dell'offset di tutte le partizioni dell'argomento. Se il ritardo dell'offset è troppo alto, lo shard sta ricevendo messaggi più velocemente di quanto Lambda possa elaborarli. Se necessario, Lambda aggiunge o rimuove i poller di eventi dall'argomento. Questo processo di dimensionamento automatico di aggiunta o rimozione dei poller degli eventi avviene entro tre minuti dalla valutazione.

Se la funzione Lambda di destinazione è limitata, Lambda riduce il numero di poller di eventi. Questa operazione riduce il carico di lavoro sulla funzione riducendo il numero di messaggi che i poller di eventi possono recuperare e inviare alla funzione.

### Modalità provisioning

Per i carichi di lavoro in cui è necessario ottimizzare il throughput dello strumento di mappatura dell'origine degli eventi, è possibile utilizzare la modalità provisioning. In modalità provisioning, vengono definiti i limiti minimi e massimi per la quantità di poller di eventi assegnati. Questi poller di eventi con provisioning sono dedicati allo strumento di mappatura dell'origine degli eventi e possono gestire picchi di messaggi imprevisti tramite un dimensionamento automatico reattivo. Ti consigliamo di utilizzare la modalità provisioning per i carichi di lavoro Kafka che hanno requisiti di prestazioni rigorosi.

In Lambda, un event poller è un'unità di calcolo con capacità di throughput che variano in base al tipo di origine dell'evento. Per Amazon MSK e Apache Kafka autogestito, ogni poller di eventi può gestire fino al 5% del throughput o fino a 5 chiamate simultanee. MB/sec Ad esempio, se la sorgente dell'evento produce un payload medio di 1 MB e la durata media della funzione è di 1 secondo, un singolo poller di eventi Kafka può supportare 5 throughput e 5 invocazioni Lambda simultanee

(supponendo che non vi sia alcuna trasformazione del payload). MB/sec Per Amazon SQS, ogni poller di eventi può gestire fino all'1% del throughput o fino a 10 chiamate simultanee. MB/sec L'utilizzo della modalità provisioned comporta costi aggiuntivi in base all'utilizzo del poller di eventi. Per i dettagli sui prezzi, vedere [Prezzi di AWS Lambda](#).

#### Note

Quando si utilizza la modalità provisioning, non è necessario creare endpoint AWS PrivateLink VPC o concedere le autorizzazioni associate come parte della configurazione di rete.

In modalità provisioning, l'intervallo di valori accettati per il numero minimo di poller di event (MinimumPollers) è compreso tra 1 e 200, inclusi. L'intervallo di valori accettati per il numero massimo di poller di eventi (MaximumPollers) è compreso tra 1 e 2.000, inclusi. MaximumPollers deve essere maggiore o uguale a MinimumPollers. Inoltre, per mantenere l'elaborazione ordinata all'interno delle partizioni, Lambda limita a MaximumPollers il numero di partizioni indicato nell'argomento.

Per ulteriori informazioni sulla scelta dei valori minimi e massimi appropriati di poller di eventi, consulta [Best practice](#).

Puoi configurare la modalità provisioned per la mappatura della sorgente degli eventi Kafka utilizzando la console o l'API Lambda.

Per configurare la modalità predisposta per una mappatura della sorgente di eventi esistente (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione con la mappatura dell'origine degli eventi per la quale desideri configurare la modalità provisioned.
3. Scegli la scheda Configurazione, quindi scegli Trigger.
4. Scegli la mappatura dell'origine degli eventi per la quale desideri configurare la modalità provisioning, quindi scegli Modifica.
5. In modalità provvisoria, seleziona Configura.
  - Per Numero minimo di poller di eventi, inserisci un valore compreso tra 1 e 200. Se non si specifica un valore, Lambda assegna il valore predefinito 1.

- Per Numero massimo di poller di eventi, inserisci un valore compreso tra 1 e 2.000. Questo valore deve essere maggiore o uguale al valore specificato in Numero minimo di poller di eventi. Se non si specifica un valore, Lambda assegna il valore predefinito 200.

6. Scegli Save (Salva).

È possibile configurare la modalità provisioning a livello di codice utilizzando l'oggetto nel [ProvisionedPollerConfig](#) file. [EventSourceMappingConfiguration](#) Ad esempio, il seguente comando [UpdateEventSourceMapping](#) CLI configura un MinimumPollers valore di 5 e un valore di 100. MaximumPollers

```
aws lambda update-event-source-mapping \  
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \  
  --provisioned-poller-config '{"MinimumPollers": 5, "MaximumPollers": 100}'
```

Dopo aver configurato la modalità provisioning, puoi osservare l'utilizzo dei poller di eventi per il tuo carico di lavoro monitorando il parametro ProvisionedPollers. Per ulteriori informazioni, consulta [the section called “Parametri dello strumento di mappatura dell'origine degli eventi”](#).

Per disabilitare la modalità provisioned e tornare alla modalità predefinita (su richiesta), è possibile utilizzare il seguente comando CLI: [UpdateEventSourceMapping](#)

```
aws lambda update-event-source-mapping \  
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \  
  --provisioned-poller-config '{}'
```

Funzionalità avanzate di gestione degli errori e prestazioni

Per le mappature delle sorgenti di eventi Kafka con la modalità provisioned abilitata, puoi configurare funzionalità aggiuntive per migliorare la gestione e le prestazioni degli errori:

- [Riprova le configurazioni](#): controlla il modo in cui Lambda gestisce i record non riusciti con il numero massimo di tentativi di ripetizione, i limiti di età dei record, la suddivisione dei batch e le risposte batch parziali.
- [Destinazioni Kafka in caso di errore](#): invia i record non riusciti a un argomento di Kafka per elaborarli o analizzarli successivamente.



## Best practice e considerazioni sull'utilizzo della modalità provisioning

La configurazione ottimale dei poller di eventi minimi e massimi per lo strumento di mappatura dell'origine degli eventi dipende dai requisiti delle prestazioni dell'applicazione. Ti consigliamo di iniziare con i poller di eventi minimi di default per definire il profilo delle prestazioni. Modifica la configurazione in base ai modelli di elaborazione dei messaggi osservati e al profilo delle prestazioni desiderato.

Per carichi di lavoro con picchi di traffico e requisiti delle prestazioni rigorosi, aumenta il numero minimo di poller di eventi per gestire picchi improvvisi di messaggi. Per determinare il numero minimo di poller di eventi richiesti, considera i messaggi al secondo del carico di lavoro e la dimensione media del payload e utilizza la capacità effettiva di throughput di un singolo poller di eventi (fino a 5 Mbps) come riferimento.

Per mantenere l'elaborazione ordinata all'interno di uno shard, Lambda limita il numero massimo di poller di eventi al numero di shard nell'argomento. Inoltre, il numero massimo di poller di eventi a cui lo strumento di mappatura dell'origine degli eventi può scalare dipende dalle impostazioni di simultaneità della funzione.

Quando attivi la modalità provisioned, aggiorna le impostazioni di rete per rimuovere gli endpoint VPC e le autorizzazioni associate. AWS PrivateLink

## Ottimizzazione dei costi per la modalità Provisioned

### Prezzi in modalità provvisoria

La modalità con provisioning viene addebitata in base ai poller di eventi minimi assegnati e ai poller di eventi utilizzati durante la scalabilità automatica. Gli addebiti vengono calcolati utilizzando un'unità di fatturazione denominata Event Poller Unit (EPU). Il pagamento viene calcolato in base al numero e alla durata delle EPU utilizzate, misurati in. Event-Poller-Unit-hours

È possibile utilizzare la modalità Provisioned con un singolo ESM per applicazioni sensibili alle prestazioni oppure è possibile raggruppare più ESM all'interno dello stesso VPC per condividere capacità e costi dell'EPU. Le sezioni seguenti descrivono due funzionalità che consentono di ottimizzare i costi della modalità Provisioned. Per i dettagli sui prezzi, consulta i prezzi di [AWS Lambda](#).

### Utilizzo migliorato dell'EPU

Ogni EPU supporta fino a 20 capacità di MB/s throughput per il polling degli eventi e supporta un valore predefinito di 10 poller di eventi. Quando si crea una modalità Provisioned per Kafka ESM

impostando i poller minimi e massimi, viene utilizzato un numero minimo di poller per il provisioning delle EPU in base all'impostazione predefinita di 10 poller di eventi per EPU. Tuttavia, ogni event poller può scalare in modo indipendente per supportare fino al 5% MB/s della capacità di throughput, il che potrebbe richiedere una minore densità di event poller su una specifica EPU e può innescare il ridimensionamento delle EPU. Il numero di event poller allocati su un'EPU dipende dalla capacità di calcolo consumata da ciascun event poller. Questo approccio di utilizzo avanzato dell'EPU consente ai poller di eventi con requisiti di throughput variabili di utilizzare la capacità EPU in modo efficace, riducendo i costi per tutti gli ESM.

## Raggruppamento ESM

Per ottimizzare ulteriormente i costi della modalità Provisioned, puoi raggruppare più ESM Kafka per condividere la capacità EPU. Con il raggruppamento ESM e l'utilizzo avanzato dell'EPU, puoi ridurre i costi della modalità Provisioned fino al 90% per i carichi di lavoro a bassa produttività rispetto all'esecuzione in modalità ESM singola. Tutti gli ESM che richiedono meno di una capacità EPU traggono vantaggio dal raggruppamento ESM. Questi ESM richiedono in genere un numero minimo di poller di eventi per supportare le loro esigenze di throughput. Grazie a questa funzionalità, puoi adottare la modalità Provisioned per tutti i tuoi carichi di lavoro Kafka e beneficiare di funzionalità come la convalida dello schema, il filtraggio Avro/Protobuf degli eventi, le chiamate a bassa latenza e una migliore gestione degli errori, disponibili solo in modalità Provisioned.

Quando configuri il `PollerGroupName` parametro con lo stesso valore per più ESM all'interno dello stesso Amazon VPC, tali ESM condividono le risorse EPU anziché richiedere ciascuna una capacità EPU dedicata. È possibile raggruppare fino a 100 ESM per gruppo di poller e aggregare il numero massimo di poller tra tutti gli ESM di un gruppo non può superare i 2000.

Per configurare il raggruppamento ESM (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere la funzione.
3. Scegli Configurazione, quindi scegli Trigger.
4. Quando si crea una nuova mappatura della sorgente di un evento Kafka o se ne modifica una esistente, selezionare Configura in modalità Provisioned.
5. Per Numero minimo di poller di eventi, inserisci un valore compreso tra 1 e 200.
6. Per Numero massimo di poller di eventi, inserisci un valore compreso tra 1 e 2.000.
7. Per il nome del gruppo Poller, inserisci un identificatore per il gruppo. Usa lo stesso nome per gli altri ESM che desideri raggruppare.

## 8. Scegli Save (Salva).

Per configurare il raggruppamento ESM (AWS CLI)

L'esempio seguente crea un ESM con un gruppo di poller denominato: `production-app-group`

```
aws lambda create-event-source-mapping \  
  --function-name myFunction1 \  
  --event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/MyCluster/abcd1234 \  
  --topics topic1 \  
  --starting-position LATEST \  
  --provisioned-poller-config '{  
    "MinimumPollers": 1,  
    "MaximumPollers": 10,  
    "PollerGroupName": "production-app-group"  
  }'
```

Per aggiungere un altro ESM allo stesso gruppo (condividendo la capacità EPU), usa lo stesso: `PollerGroupName`

```
aws lambda create-event-source-mapping \  
  --function-name myFunction2 \  
  --event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/MyCluster/abcd1234 \  
  --topics topic2 \  
  --starting-position LATEST \  
  --provisioned-poller-config '{  
    "MinimumPollers": 1,  
    "MaximumPollers": 10,  
    "PollerGroupName": "production-app-group"  
  }'
```

### Note

È possibile aggiornare l'ESM `PollerGroupName` per spostare un ESM in un gruppo diverso o rimuovere un ESM da un gruppo passando una stringa vuota («») per: `PollerGroupName`

```
# Move ESM to a different group  
aws lambda update-event-source-mapping \  
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \  
  --poller-group-name ''
```

```
--provisioned-poller-config '{
  "MinimumPollers": 1,
  "MaximumPollers": 10,
  "PollerGroupName": "new-group-name"
}'

# Remove ESM from group (use dedicated resources)
aws lambda update-event-source-mapping \
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
  --provisioned-poller-config '{
    "MinimumPollers": 1,
    "MaximumPollers": 10,
    "PollerGroupName": ""
  }'
```

## Considerazioni sulla strategia di raggruppamento

- **Limite delle applicazioni:** raggruppa gli ESM che appartengono alle stesse applicazioni o servizi per una migliore allocazione e gestione dei costi. Prendi in considerazione l'utilizzo di convenzioni di denominazione come `app-name-environment` (ad esempio, `order-processor-prod`).
- **Schema di traffico:** evita di raggruppare gli ESM con un throughput elevato e uno schema di traffico intenso, poiché ciò potrebbe portare a conflitti in termini di risorse.
- **Blast radius:** considera l'impatto di eventuali problemi nell'infrastruttura condivisa. Tutti gli ESM dello stesso gruppo sono interessati dalle limitazioni delle risorse condivise. Per i carichi di lavoro cruciali, potresti voler utilizzare gruppi separati o ESM dedicati.

## Esempio di ottimizzazione dei costi

Considerate uno scenario in cui avete 10 ESM, ciascuno configurato con un poller di eventi e un throughput inferiore a 2: MB/s

Senza raggruppamento:

- Ogni ESM richiede una propria EPU
- EPU totali necessarie: 10
- Costo per EPU: 0 USD. 185/hour negli Stati Uniti orientali (Virginia settentrionale)
- Costo mensile dell'EPU (720 ore):  $10 \times 720 \times 0,185 \text{ USD} = 1.332 \text{ USD}$

Con raggruppamento:

- Tutti e 10 gli ESM condividono la capacità EPU
- 10 sondaggi di eventi rientrano in 1 EPU (con un nuovo supporto per 10 poller per EPU)
- EPU totali necessarie: 1
- Costo mensile dell'EPU (720 ore):  $1 \times 720 \times 0,185 \text{ USD} = 133,20 \text{ USD}$
- Risparmio sui costi: 90% (risparmio di 1.198,80 USD al mese)

## Sondaggi di Apache Kafka e posizioni di partenza per lo streaming in Lambda

Il [StartingPosition parametro](#) indica a Lambda quando iniziare a leggere i messaggi dal tuo stream Amazon MSK o Apache Kafka autogestito. È possibile scegliere tra tre opzioni:

- Ultimo: Lambda inizia a leggere subito dopo il record più recente nell'argomento Kafka.
- Trim horizon: Lambda inizia a leggere dall'ultimo record non tagliato nell'argomento Kafka. Questo è anche il record più vecchio dell'argomento.
- Al timestamp: Lambda inizia la lettura da una posizione definita da un timestamp, in secondi temporali Unix. Usa il parametro per specificare il timestamp. [StartingPositionTimestamp](#)

Il polling dello stream durante la creazione o l'aggiornamento della mappatura della sorgente di un evento è alla fine coerente:

- Durante la creazione della mappatura della sorgente degli eventi, potrebbero essere necessari diversi minuti per avviare il polling degli eventi dallo stream.
- Durante gli aggiornamenti della mappatura dell'origine degli eventi, potrebbero essere necessari fino a 90 secondi per interrompere e riavviare gli eventi di polling dallo stream.

Questo comportamento significa che se si specifica LATEST come posizione iniziale per lo stream, nella mappatura dell'origine degli eventi potrebbero mancare gli eventi durante una creazione o un aggiornamento. Per garantire che nessun evento venga perso, specifica uno TRIM\_HORIZON o AT\_TIMESTAMP.

## ID gruppo di consumer personalizzabile

Quando configuri Amazon MSK o Apache Kafka autogestita come origine, puoi specificare un ID [gruppo di consumer](#). Questo ID gruppo di consumer è un identificatore esistente per il gruppo di consumer Kafka a cui desideri che la tua funzione Lambda aderisca. Puoi utilizzare questa funzione per migrare senza problemi qualsiasi configurazione di elaborazione dei record Kafka in corso da altri utenti a Lambda.

Kafka distribuisce messaggi a tutti i consumer di un gruppo di consumer. Se si specifica un ID di gruppo di consumatori con altri consumatori attivi, Lambda riceve solo una parte dei messaggi dall'argomento Kafka. Se desideri che Lambda gestisca tutti i messaggi dell'argomento, disattiva tutti gli altri sondaggi in quel gruppo di consumer.

Inoltre, se specifichi un ID gruppo di consumer e Kafka trova un gruppo di consumer esistente valido con lo stesso ID, Lambda ignora il parametro per la mappatura dell'origine eventi. Inizia invece ad elaborare i record in base alla compensazione impegnata del gruppo di consumer. Se specifichi un ID gruppo di consumer e Kafka non riesce a trovare un gruppo di consumer esistente, Lambda configura l'origine eventi con la specificata `StartingPosition`.

L'ID gruppo di consumer deve essere univoco tra tutte le origini eventi Kafka. Dopo aver creato una mappatura dell'origine eventi Kafka con l'ID gruppo di consumer specificato, non sarà più possibile aggiornare questo valore.

## Filtraggio eventi da origini eventi Amazon MSK e Apache Kafka autogestito

Puoi utilizzare il filtraggio degli eventi per controllare quali record di un flusso o di una coda Lambda invia alla funzione. Per informazioni generali sul funzionamento del filtraggio eventi, consulta [the section called “Filtro eventi”](#).

### Note

Amazon MSK e le mappature origine eventi Apache Kafka autogestito. `value`

### Argomenti

- [Nozioni di base del filtro evento](#)

### Nozioni di base del filtro evento

Supponiamo che un produttore stia scrivendo messaggi su un argomento nel tuo cluster Amazon MSK, in formato JSON valido o come stringhe semplici. Un record di esempio sarebbe simile al seguente, con il messaggio convertito in una stringa codificata Base64 nel campo `value`.

```
{
  "mytopic-0": [
    {
      "topic": "mytopic",
      "partition": 0,
      "offset": 15,
      "timestamp": 1545084650987,
      "timestampType": "CREATE_TIME",
      "value": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
      "headers": []
    }
  ]
}
```

Supponiamo che il produttore Apache Kafka stia scrivendo messaggi sul tuo argomento nel seguente formato JSON.

```
{
  "device_ID": "AB1234",
  "session": {
```



```

        "start_time": "yyyy-mm-ddThh:mm:ss",
        "duration": 162
    }
}

```

Puoi utilizzare la chiave `value` per filtrare i record. Supponiamo di voler filtrare solo i record in cui `device_ID` inizia con le lettere AB. L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```

{
  "Filters": [
    {
      "Pattern": "{ \"value\" : { \"device_ID\" : [ { \"prefix\": \"AB\" } ] } }"
    }
  ]
}

```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```

{
  "value": {
    "device_ID": [ { "prefix": "AB" } ]
  }
}

```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

### Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in [Collegamento dei criteri di filtro a una mappatura dell'origine evento \(console\)](#) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "value" : { "device_ID" : [ { "prefix": "AB" } ] } }
```

### AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
aws lambda create-event-source-mapping \
```

```
--function-name my-function \
--event-source-arn arn:aws:kafka:us-east-2:123456789012:cluster/my-cluster/
b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
--filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :
[ { \"prefix\": \"AB\" } ] } }"]}]'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
--uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
--filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :
[ { \"prefix\": \"AB\" } ] } }"]}]'
```

## AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : { "device_ID" : [ { "prefix": "AB" } ] } }'
```

Con Amazon MSK, puoi anche filtrare i record in cui il messaggio è una stringa semplice. Supponiamo di voler ignorare i messaggi la cui stringa è "errore". L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"value\" : [ { \"anything-but\": [ \"error\" ] } ] }"
    }
  ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
  "value": [
    {
```

```

    "anything-but": [ "error" ]
  }
]
}

```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

## Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in [Collegamento dei criteri di filtro a una mappatura dell'origine evento \(console\)](#) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "value" : [ { "anything-but": [ "error" ] } ] }
```

## AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:kafka:us-east-2:123456789012:cluster/my-cluster/  
b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":  
[ \"error\" ] } ] }"]}]'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":  
[ \"error\" ] } ] }"]}]'
```

## AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
```

```
- Pattern: '{ "value" : [ { "anything-but": [ "error" ] } ] }'
```

I messaggi Kafka e Amazon MQ devono essere stringhe codificate in UTF-8, stringhe semplici o in formato JSON. Questo perché Lambda decodifica gli array di byte Kafka in UTF-8 prima di applicare i criteri di filtro. Se i messaggi utilizzano un'altra codifica, ad esempio UTF-16 o ASCII o se il formato del messaggio non corrisponde al formato `FilterCriteria`, Lambda elabora solo i filtri di metadati. La tabella seguente riepiloga il comportamento specifico:

| Formato messaggio in arrivo  | Formato del modello di filtro per le proprietà di messaggi | Operazione risultante                                                                 |
|------------------------------|------------------------------------------------------------|---------------------------------------------------------------------------------------|
| Stringa normale              | Stringa normale                                            | Filtri Lambda in base ai criteri di filtro.                                           |
| Stringa normale              | Nessun modello di filtro per le proprietà dei dati         | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro. |
| Stringa normale              | JSON valido                                                | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro. |
| JSON valido                  | Stringa normale                                            | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro. |
| JSON valido                  | Nessun modello di filtro per le proprietà dei dati         | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro. |
| JSON valido                  | JSON valido                                                | Filtri Lambda in base ai criteri di filtro.                                           |
| Stringa codificata non UTF-8 | JSON, stringa semplice o nessun modello                    | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro. |

## Utilizzo di registri di schema con sorgenti di eventi Kafka in Lambda

I registri degli schemi ti aiutano a definire e gestire gli schemi dei flussi di dati. Uno schema definisce la struttura e il formato di un registro di dati. Nel contesto delle mappature delle sorgenti degli eventi Kafka, puoi configurare un registro degli schemi per convalidare la struttura e il formato dei messaggi Kafka rispetto a schemi predefiniti prima che raggiungano la tua funzione Lambda. Ciò aggiunge un livello di governance dei dati all'applicazione e consente di gestire in modo efficiente i formati di dati, garantire la conformità degli schemi e ottimizzare i costi tramite il filtro degli eventi.

Questa funzionalità funziona con tutti i linguaggi di programmazione, ma considera questi punti importanti:

- Powertools for Lambda fornisce un supporto specifico per Java, Python e TypeScript, mantenendo la coerenza con i modelli di sviluppo esistenti di Kafka e consentendo l'accesso diretto agli oggetti aziendali senza codice di deserializzazione personalizzato
- Questa funzionalità è disponibile solo per le mappature delle sorgenti degli eventi utilizzando la modalità provisioned. Il registro degli schemi non supporta le mappature delle origini degli eventi in modalità su richiesta. Se si utilizza la modalità provisioning e si dispone di un registro degli schemi configurato, non è possibile passare alla modalità su richiesta a meno che non si rimuova prima la configurazione del registro degli schemi. Per ulteriori informazioni, consulta [the section called “Modalità provisioning”](#)
- È possibile configurare un solo registro di schemi per Event Source Mapping (ESM). L'utilizzo di un registro di schemi con la fonte di eventi Kafka potrebbe aumentare l'utilizzo della Lambda Event Poller Unit (EPU), che è una dimensione di determinazione dei prezzi per la modalità Provisioned.

### Argomenti

- [Opzioni del registro dello schema](#)
- [In che modo Lambda esegue la convalida dello schema per i messaggi Kafka](#)
- [Configurazione di un registro di schemi Kafka](#)
- [Filtraggio per Avro e Protobuf](#)
- [Formati di payload e comportamento di deserializzazione](#)
- [Lavorare con dati deserializzati nelle funzioni Lambda](#)
- [Metodi di autenticazione per il registro degli schemi](#)
- [Gestione degli errori e risoluzione dei problemi relativi al registro dello schema](#)

## Opzioni del registro dello schema

Lambda supporta le seguenti opzioni di registro degli schemi:

- [AWS Glue Registro degli schemi](#)
- [Registro degli schemi di Confluent Cloud](#)
- [Self-managed Registro degli schemi confluenti](#)

Il registro degli schemi supporta la convalida dei messaggi nei seguenti formati di dati:

- Apache Avro
- Buffer di protocollo (Protobuf)
- Schema JSON () JSON-SE

Per utilizzare un registro degli schemi, assicuratevi innanzitutto che la mappatura dell'origine degli eventi sia in modalità provisioning. Quando si utilizza un registro di schemi, Lambda aggiunge i metadati sullo schema al payload. Per ulteriori informazioni, consulta [Formati di payload e comportamento di deserializzazione](#).

In che modo Lambda esegue la convalida dello schema per i messaggi Kafka

Quando si configura un registro degli schemi, Lambda esegue i seguenti passaggi per ogni messaggio Kafka:

1. Lambda esegue il polling del record Kafka dal cluster.
2. Lambda convalida gli attributi dei messaggi selezionati nel record rispetto a uno schema specifico nel registro degli schemi.
  - Se lo schema associato al messaggio non viene trovato nel registro, Lambda invia il messaggio a un DLQ con il codice del motivo. `SCHEMA_NOT_FOUND`
3. Lambda deserializza il messaggio in base alla configurazione del registro dello schema per convalidare il messaggio. Se il filtro degli eventi è configurato, Lambda esegue quindi il filtraggio in base ai criteri di filtro configurati.
  - Se la deserializzazione fallisce, Lambda invia il messaggio a un DLQ con il codice del motivo. `DESERIALIZATION_ERROR` Se non è configurato alcun DLQ, Lambda elimina il messaggio.
4. Se il messaggio viene convalidato dal registro degli schemi e non viene filtrato in base ai criteri di filtro, Lambda richiama la funzione con il messaggio.

Questa funzionalità ha lo scopo di convalidare i messaggi già prodotti utilizzando client Kafka integrati con un registro di schemi. Consigliamo di configurare i produttori Kafka in modo che funzionino con il registro degli schemi per creare messaggi formattati correttamente.

## Configurazione di un registro di schemi Kafka

I seguenti passaggi della console aggiungono una configurazione del registro dello schema Kafka alla mappatura dell'origine degli eventi.

Per aggiungere una configurazione del registro dello schema Kafka alla mappatura della sorgente degli eventi (console)

1. Apri la [pagina Funzione](#) della console Lambda.
2. Scegliere Configuration (Configurazione).
3. Scegli Triggers.
4. Seleziona la mappatura della sorgente degli eventi Kafka per la quale desideri configurare un registro degli schemi e scegli Modifica.
5. In Event poller configuration, scegli Configura il registro degli schemi. La mappatura dell'origine degli eventi deve essere in modalità provisioning per visualizzare questa opzione.
6. Per l'URI del registro di schema, inserisci l'ARN del tuo registro degli schemi o l'URL HTTPS del tuo registro di AWS Glue schema Confluent Cloud o del registro degli schemi di Confluent. Self-Managed
7. I seguenti passaggi di configurazione indicano a Lambda come accedere al registro degli schemi. Per ulteriori informazioni, consulta [???](#).
  - Per il tipo di configurazione di Access, scegli il tipo di autenticazione utilizzato da Lambda per accedere al registro degli schemi.
  - Per l'URI di configurazione di Access, inserisci l'ARN del segreto di Secrets Manager per autenticarti con il registro dello schema, se applicabile. Assicurati che il ruolo di [esecuzione della tua funzione](#) contenga le autorizzazioni corrette.
8. Il campo Encryption si applica solo se il registro dello schema è firmato da un'autorità di certificazione (CA) privata o da un'autorità di certificazione (CA) non presente nel trust store Lambda. Se applicabile, fornisci la chiave segreta contenente il certificato CA privato utilizzato dal registro dello schema per la crittografia TLS.
9. Per il formato dei record degli eventi, scegli come desideri che Lambda fornisca i record alla tua funzione dopo la convalida dello schema. Per ulteriori informazioni, consulta gli [esempi di formati di payload](#).

- Se scegli JSON, Lambda fornisce gli attributi selezionati nell'attributo di convalida dello schema riportato di seguito in formato JSON standard. Per gli attributi che non selezioni, Lambda li fornisce così come sono.
  - Se scegli SOURCE, Lambda fornisce gli attributi selezionati nell'attributo di convalida dello schema riportato di seguito nel formato sorgente originale.
10. Per l'attributo di convalida dello schema, seleziona gli attributi del messaggio che desideri che Lambda convalidi e deserializzi utilizzando il registro degli schemi. È necessario selezionare almeno una delle opzioni KEY o VALUE. Se hai scelto JSON come formato di registrazione degli eventi, Lambda deserializza anche gli attributi dei messaggi selezionati prima di inviarli alla tua funzione. Per ulteriori informazioni, consulta Formati di [payload e comportamento di deserializzazione](#).
11. Scegli Save (Salva).

Puoi anche utilizzare l'API Lambda per creare o aggiornare la mappatura della sorgente degli eventi con una configurazione del registro dello schema. Gli esempi seguenti mostrano come configurare un registro di schemi AWS Glue o Confluent utilizzando il AWS CLI, che corrisponde alle operazioni [CreateEventSourceMapping](#) API [UpdateEventSourceMapping](#) e nell'API Reference: AWS Lambda

### Important

Se si sta aggiornando un campo di configurazione del registro dello schema utilizzando AWS CLI o l'update-event-source-mappingAPI, è necessario aggiornare tutti i campi della configurazione del registro di schema.

## Create Event Source Mapping

```
aws lambda create-event-source-mapping \
  --function-name my-schema-validator-function \
  --event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/my-cluster/
a1b2c3d4-5678-90ab-cdef-11111EXAMPLE \
  --topics my-kafka-topic \
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=1 \
  --amazon-managed-kafka-event-source-mapping '{
    "SchemaRegistryConfig" : {
      "SchemaRegistryURI": "https://abcd-ef123.us-west-2.aws.confluent.cloud",
      "AccessConfigs": [{
```



```

        "Type": "BASIC_AUTH",
        "URI": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:secretName"
    }],
    "EventRecordFormat": "JSON",
    "SchemaValidationConfigs": [
        {
            "Attribute": "KEY"
        },
        {
            "Attribute": "VALUE"
        }
    ]
}
}'

```

## Update AWS Glue Schema Registry

```

aws lambda update-event-source-mapping \
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
  --amazon-managed-kafka-event-source-mapping '{
    "SchemaRegistryConfig" : {
      "SchemaRegistryURI": "arn:aws:glue:us-east-1:123456789012:registry/
registryName",
      "EventRecordFormat": "JSON",
      "SchemaValidationConfigs": [
        {
          "Attribute": "KEY"
        },
        {
          "Attribute": "VALUE"
        }
      ]
    }
  }'

```

## Update Confluent Schema Registry with Authentication

```

aws lambda update-event-source-mapping \
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
  --amazon-managed-kafka-event-source-mapping '{
    "SchemaRegistryConfig" : {
      "SchemaRegistryURI": "https://abcd-ef123.us-west-2.aws.confluent.cloud",
      "AccessConfigs": [{
        "Type": "BASIC_AUTH",

```

```

        "URI": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:secretName"
    }],
    "EventRecordFormat": "JSON",
    "SchemaValidationConfigs": [
    {
        "Attribute": "KEY"
    },
    {
        "Attribute": "VALUE"
    }
    ]
}
}'

```

## Update Confluent Schema Registry without Authentication

```

aws lambda update-event-source-mapping \
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
  --amazon-managed-kafka-event-source-mapping '{
    "SchemaRegistryConfig" : {
      "SchemaRegistryURI": "https://abcd-ef123.us-west-2.aws.confluent.cloud",
      "EventRecordFormat": "JSON",
      "SchemaValidationConfigs": [
        {
          "Attribute": "KEY"
        },
        {
          "Attribute": "VALUE"
        }
      ]
    }
  }'

```

## Remove Schema Registry Configuration

Per rimuovere una configurazione del registro dello schema dalla mappatura dell'origine degli eventi, puoi utilizzare il comando CLI [UpdateEventSourceMapping](#) nell'AWS Lambda API Reference.

```

aws lambda update-event-source-mapping \
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
  --amazon-managed-kafka-event-source-mapping '{
    "SchemaRegistryConfig" : {}
  }'

```

```
}'
```

## Filtraggio per Avro e Protobuf

Quando si utilizzano i formati Avro o Protobuf con un registro di schemi, è possibile applicare il filtro degli eventi alla funzione Lambda. I modelli di filtro vengono applicati alla classica rappresentazione JSON deserializzata dei dati dopo la convalida dello schema. Ad esempio, con uno schema Avro che definisce i dettagli del prodotto, incluso il prezzo, puoi filtrare i messaggi in base al valore del prezzo:

### Note

Quando viene deserializzato, Avro viene convertito in JSON standard, il che significa che non può essere riconvertito direttamente in un oggetto Avro. Se devi convertire in un oggetto Avro, usa invece il formato SOURCE.

Per la deserializzazione di Protobuf, i nomi dei campi nel JSON risultante corrispondono a quelli definiti nello schema, anziché essere convertiti in maiuscole e minuscole come normalmente fa Protobuf. Tienilo a mente quando crei modelli di filtro.

```
aws lambda create-event-source-mapping \
  --function-name myAvroFunction \
  --topics myAvroTopic \
  --starting-position TRIM_HORIZON \
  --kafka-bootstrap-servers '["broker1:9092", "broker2:9092"]' \
  --schema-registry-config '{
    "SchemaRegistryURI": "arn:aws:glue:us-east-1:123456789012:registry/
myAvroRegistry",
    "EventRecordFormat": "JSON",
    "SchemaValidationConfigs": [
      {
        "Attribute": "VALUE"
      }
    ]
  }' \
  --filter-criteria '{
    "Filters": [
      {
        "Pattern": "{ \"value\" : { \"field_1\" : [\"value1\"], \"field_2\" :
[\"value2\"] } }"
```

```
]
}'
```

In questo esempio, lo schema di filtro analizza l'oggetto `value`, abbinando i messaggi `field_1` con `"value1"` e `field_2` con `"value2"`. I criteri di filtro vengono valutati rispetto ai dati deserializzati, dopo che Lambda ha convertito il messaggio dal formato Avro a JSON.

Per informazioni più dettagliate sul filtro degli eventi, consulta [Filtraggio degli eventi Lambda](#).

### Formati di payload e comportamento di deserializzazione

Quando si utilizza un registro di schemi, Lambda fornisce il payload finale alla funzione in un formato simile al [normale payload degli eventi](#), con alcuni campi aggiuntivi. I campi aggiuntivi dipendono dal parametro `SchemaValidationConfigs`. Per ogni attributo selezionato per la convalida (chiave o valore), Lambda aggiunge i metadati dello schema corrispondenti al payload.

#### Note

È necessario aggiornare <https://github.com/aws/aws-lambda-java-libs/tree/main/aws-lambda-java-events> alla versione 3.16.0 o successiva per utilizzare i campi di metadati dello schema.

Ad esempio, se convalidi il campo, Lambda aggiunge un campo chiamato `valueSchemaMetadata`. Analogamente, per il key campo, Lambda aggiunge un campo chiamato `keySchemaMetadata`. Questi metadati contengono informazioni sul formato dei dati e sull'ID dello schema utilizzato per la convalida:

```
"valueSchemaMetadata": {
  "dataFormat": "AVRO",
  "schemaId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

Il `EventRecordFormat` parametro può essere impostato su `JSON` o `SOURCE`, il che determina in che modo Lambda gestisce i dati convalidati dallo schema prima di consegnarli alla funzione. Ogni opzione offre diverse funzionalità di elaborazione:

- **JSON**- Lambda deserializza gli attributi convalidati in formato JSON standard, rendendo i dati pronti per l'uso diretto nei linguaggi con supporto JSON nativo. Questo formato è ideale quando non è necessario preservare il formato binario originale o lavorare con classi generate.

- **SOURCE**- Lambda conserva il formato binario originale dei dati come Base64-encoded stringa, consentendo la conversione diretta in oggetti Avro o Protobuf. Questo formato è essenziale quando si lavora con linguaggi fortemente tipizzati o quando è necessario mantenere tutte le funzionalità degli schemi Avro o Protobuf.

In base a queste caratteristiche di formato e a considerazioni specifiche della lingua, consigliamo i seguenti formati:

Formati consigliati basati sul linguaggio di programmazione

| Lingua | Avro   | Protobuf | JSON   |
|--------|--------|----------|--------|
| Java   | SOURCE | SOURCE   | SOURCE |
| Python | JSON   | JSON     | JSON   |
| NodeJS | JSON   | JSON     | JSON   |
| .NET   | SOURCE | SOURCE   | SOURCE |
| Altri  | JSON   | JSON     | JSON   |

Le sezioni seguenti descrivono questi formati in dettaglio e forniscono esempi di payload per ogni formato.

## Formato JSON

Se scegli `JSON` come `EventRecordFormat`, Lambda convalida e deserializza gli attributi del messaggio che hai selezionato nel campo (gli attributi). `SchemaValidationConfigs` `key` and/or `value` Lambda fornisce questi attributi selezionati come stringhe codificate in base64 della loro rappresentazione JSON standard nella funzione.

### Note

Quando viene deserializzato, Avro viene convertito in JSON standard, il che significa che non può essere riconvertito direttamente in un oggetto Avro. Se devi convertire in un oggetto Avro, usa invece il formato `SOURCE`.

Per la deserializzazione di Protobuf, i nomi dei campi nel JSON risultante corrispondono a quelli definiti nello schema, anziché essere convertiti in maiuscole e minuscole come normalmente fa Protobuf. Tienilo a mente quando crei modelli di filtro.

Di seguito viene mostrato un esempio di payload, supponendo che tu scelga JSON come EventRecordFormat ed entrambi gli attributi key e value come SchemaValidationConfigs:

```
{
  "eventSource": "aws:kafka",
  "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/vpc-2priv-2pub/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111-1",
  "bootstrapServers": "b-2.demo-cluster-1.a1bcde.c1.kafka.us-east-1.amazonaws.com:9092,b-1.demo-cluster-1.a1bcde.c1.kafka.us-east-1.amazonaws.com:9092",
  "records": {
    "mytopic-0": [
      {
        "topic": "mytopic",
        "partition": 0,
        "offset": 15,
        "timestamp": 1545084650987,
        "timestampType": "CREATE_TIME",
        "key": "abcDEFghiJKLmnoPQRstuVWXyz1234==", //Base64 encoded string of JSON
        "keySchemaMetadata": {
          "dataFormat": "AVRO",
          "schemaId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        },
        "value": "abcDEFghiJKLmnoPQRstuVWXyz1234", //Base64 encoded string of JSON
        "valueSchemaMetadata": {
          "dataFormat": "AVRO",
          "schemaId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        },
        "headers": [
          {
            "headerKey": [
              104,
              101,
              97,
              100,
              101,
              114,
```

```

}
    }
]
    }
]
        }
            ]
                86,
                97,
                108,
                117,
                101
```

In questo esempio:

- Entrambe `key` e `value` sono stringhe codificate in base64 della loro rappresentazione JSON dopo la deserializzazione.
- Lambda include i metadati dello schema per entrambi gli attributi in `e.keySchemaMetadata` e `e.valueSchemaMetadata`
- La tua funzione può decodificare le stringhe `key` e `value` per accedere ai dati JSON deserializzati.

Il formato JSON è consigliato per i linguaggi che non sono fortemente digitati, come Python o Node.js. Questi linguaggi hanno un supporto nativo per la conversione di JSON in oggetti.

## Formato sorgente

Se lo scegli `SOURCEEventRecordFormat`, Lambda convalida comunque il record rispetto al registro dello schema, ma fornisce i dati binari originali alla tua funzione senza deserializzazione. Questi dati binari vengono forniti come stringa codificata Base64 dei dati in byte originali, con i metadati aggiunti dal produttore rimossi. Di conseguenza, è possibile convertire direttamente i dati binari grezzi in oggetti Avro e Protobuf all'interno del codice funzione. Consigliamo di utilizzare Powertools for AWS Lambda, che deserializza i dati binari grezzi e fornisce direttamente gli oggetti Avro e Protobuf.

Ad esempio, se configuri Lambda per convalidare entrambi gli attributi `key` e `value` ma utilizzi il formato `SOURCE`, la tua funzione riceve un payload come questo:

```
{
  "eventSource": "aws:kafka",
  "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/vpc-2priv-2pub/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111-1",

```

```

    "bootstrapServers": "b-2.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092,b-1.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092",
    "records": {
        "mytopic-0": [
            {
                "topic": "mytopic",
                "partition": 0,
                "offset": 15,
                "timestamp": 1545084650987,
                "timestampType": "CREATE_TIME",
                "key": "abcDEFghiJKLmnoPQRstuVWXYZ1234==", // Base64 encoded string of
Original byte data, producer-appended metadata removed
                "keySchemaMetadata": {
                    "dataFormat": "AVRO",
                    "schemaId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
                },
                "value": "abcDEFghiJKLmnoPQRstuVWXYZ1234==", // Base64 encoded string
of Original byte data, producer-appended metadata removed
                "valueSchemaMetadata": {
                    "dataFormat": "AVRO",
                    "schemaId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
                },
                "headers": [
                    {
                        "headerKey": [
                            104,
                            101,
                            97,
                            100,
                            101,
                            114,
                            86,
                            97,
                            108,
                            117,
                            101
                        ]
                    }
                ]
            }
        ]
    }
}

```



```
}
```

In questo esempio:

- Sia `key` che `value` contengono i dati binari originali come stringhe codificate in Base64.
- La tua funzione deve gestire la deserializzazione utilizzando le librerie appropriate.

La `SOURCE` scelta `EventRecordFormat` è consigliata se si utilizzano Protobuf-generated oggetti, Avro-generated in particolare con funzioni Java. Questo perché Java è fortemente tipizzato e richiede deserializzatori specifici per i formati Avro e Protobuf. Nel codice funzione, puoi utilizzare la tua libreria Avro o Protobuf preferita per deserializzare i dati.

Lavorare con dati deserializzati nelle funzioni Lambda

Powertools for AWS Lambda aiuta a deserializzare i record Kafka nel tuo codice funzione in base al formato che usi. Questa utilità semplifica l'utilizzo dei record Kafka gestendo la conversione dei dati e fornendo oggetti pronti all'uso.

Per utilizzare Powertools for AWS Lambda nella tua funzione, devi aggiungere Powertools for come livello o includerlo come AWS Lambda dipendenza quando crei la tua funzione Lambda. Per istruzioni di configurazione e ulteriori informazioni, consulta Powertools per la AWS Lambda documentazione relativa alla tua lingua preferita:

- [Powertools per AWS Lambda \(Java\)](#)
- [Powertools per AWS Lambda \(Python\)](#)
- [Powertools per \(\) AWS Lambda TypeScript](#)
- [Powertools per AWS Lambda \(.NET\)](#)

#### Note

Quando si lavora con l'integrazione del registro degli schemi, è possibile scegliere `SOURCE` o `JSON` formattare. Ogni opzione supporta diversi formati di serializzazione, come illustrato di seguito:

| Formato | Supporta                                                                |
|---------|-------------------------------------------------------------------------|
| SOURCE  | Avro e Protobuf (utilizzando l'integrazione del Lambda Schema Registry) |
| JSON    | Dati JSON                                                               |

Quando si utilizza il JSON formato SOURCE or, è possibile utilizzare Powertools per aiutare AWS a deserializzare i dati nel codice funzione. Ecco alcuni esempi di come gestire diversi formati di dati:

## AVRO

Esempio in Java:

```
package org.demo.kafka;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.demo.kafka.avro.AvroProduct;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import software.amazon.lambda.powertools.kafka.Deserialization;
import software.amazon.lambda.powertools.kafka.DeserializationType;
import software.amazon.lambda.powertools.logging.Logging;

public class AvroDeserializationFunction implements
    RequestHandler<ConsumerRecords<String, AvroProduct>, String> {

    private static final Logger LOGGER =
        LoggerFactory.getLogger(AvroDeserializationFunction.class);

    @Override
    @Logging
    @Deserialization(type = DeserializationType.KAFKA_AVRO)
    public String handleRequest(ConsumerRecords<String, AvroProduct> records,
        Context context) {
```

```

        for (ConsumerRecord<String, AvroProduct> consumerRecord : records) {
            LOGGER.info("ConsumerRecord: {}", consumerRecord);

            AvroProduct product = consumerRecord.value();
            LOGGER.info("AvroProduct: {}", product);

            String key = consumerRecord.key();
            LOGGER.info("Key: {}", key);
        }

        return "OK";
    }
}

```

### Esempio di Python:

```

from aws_lambda_powertools.utilities.kafka_consumer.kafka_consumer import
    kafka_consumer
from aws_lambda_powertools.utilities.kafka_consumer.schema_config import
    SchemaConfig
from aws_lambda_powertools.utilities.kafka_consumer.consumer_records import
    ConsumerRecords

from aws_lambda_powertools.utilities.typing import LambdaContext
from aws_lambda_powertools import Logger

logger = Logger(service="kafkaConsumerPowertools")

value_schema_str = open("customer_profile.avsc", "r").read()

schema_config = SchemaConfig(
    value_schema_type="AVRO",
    value_schema=value_schema_str)

@kafka_consumer(schema_config=schema_config)
def lambda_handler(event: ConsumerRecords, context: LambdaContext):

    for record in event.records:
        value = record.value
        logger.info(f"Received value: {value}")

```

### TypeScript esempio:

```

import { kafkaConsumer } from '@aws-lambda-powertools/kafka';

import type { ConsumerRecords } from '@aws-lambda-powertools/kafka/types';
import { Logger } from '@aws-lambda-powertools/logger';
import type { Context } from 'aws-lambda';

const logger = new Logger();

type Value = {
  id: number;
  name: string;
  price: number;
};

const schema = '{
  "type": "record",
  "name": "Product",
  "fields": [
    { "name": "id", "type": "int" },
    { "name": "name", "type": "string" },
    { "name": "price", "type": "double" }
  ]
}';

export const handler = kafkaConsumer<string, Value>(
  (event: ConsumerRecords<string, Value>, _context: Context) => {
    for (const record of event.records) {
      logger.info(Processing record with key: ${record.key});
      logger.info(Record value: ${JSON.stringify(record.value)});
      // You can add more processing logic here
    }
  },
  {
    value: {
      type: 'avro',
      schema: schema,
    },
  }
);

```

esempio.NET:

```
using Amazon.Lambda.Core;
```

```

using AWS.Lambda.Powertools.Kafka;
using AWS.Lambda.Powertools.Kafka.Avro;
using AWS.Lambda.Powertools.Logging;
using Com.Example;

// Assembly attribute to enable the Lambda function's Kafka event to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(PowertoolsKafkaAvroSerializer))]

namespace ProtoBufClassLibrary;

public class Function
{
    public string FunctionHandler(ConsumerRecords<string, CustomerProfile> records,
        ILambdaContext context)
    {
        foreach (var record in records)
        {
            Logger.LogInformation("Processing messagem from topic: {topic}",
                record.Topic);
            Logger.LogInformation("Partition: {partition}, Offset: {offset}",
                record.Partition, record.Offset);
            Logger.LogInformation("Produced at: {timestamp}", record.Timestamp);

            foreach (var header in record.Headers.DecodedValues())
            {
                Logger.LogInformation($"{header.Key}: {header.Value}");
            }

            Logger.LogInformation("Processing order for: {fullName}",
                record.Value.FullName);
        }

        return "Processed " + records.Count() + " records";
    }
}

```

## PROTOBUF

### Esempio Java:

```

package org.demo.kafka;

import org.apache.kafka.clients.consumer.ConsumerRecord;

```

```

import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.demo.kafka.protobuf.ProtobufProduct;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import software.amazon.lambda.powertools.kafka.Deserialization;
import software.amazon.lambda.powertools.kafka.DeserializationType;
import software.amazon.lambda.powertools.logging.Logging;

public class ProtobufDeserializationFunction
    implements RequestHandler<ConsumerRecords<String, ProtobufProduct>, String>
{

    private static final Logger LOGGER =
        LoggerFactory.getLogger(ProtobufDeserializationFunction.class);

    @Override
    @Logging
    @Deserialization(type = DeserializationType.KAFKA_PROTOBUF)
    public String handleRequest(ConsumerRecords<String, ProtobufProduct> records,
        Context context) {
        for (ConsumerRecord<String, ProtobufProduct> consumerRecord : records) {
            LOGGER.info("ConsumerRecord: {}", consumerRecord);

            ProtobufProduct product = consumerRecord.value();
            LOGGER.info("ProtobufProduct: {}", product);

            String key = consumerRecord.key();
            LOGGER.info("Key: {}", key);
        }

        return "OK";
    }
}

```

Esempio di Python:

```

from aws_lambda_powertools.utilities.kafka_consumer.kafka_consumer import
    kafka_consumer

```

```

from aws_lambda_powertools.utilities.kafka_consumer.schema_config import
    SchemaConfig
from aws_lambda_powertools.utilities.kafka_consumer.consumer_records import
    ConsumerRecords

from aws_lambda_powertools.utilities.typing import LambdaContext
from aws_lambda_powertools import Logger

from user_pb2 import User # protobuf generated class

logger = Logger(service="kafkaConsumerPowertools")

schema_config = SchemaConfig(
    value_schema_type="PROTOBUF",
    value_schema=User)

@kafka_consumer(schema_config=schema_config)
def lambda_handler(event: ConsumerRecords, context: LambdaContext):

    for record in event.records:
        value = record.value
        logger.info(f"Received value: {value}")

```

### TypeScript esempio:

```

import { kafkaConsumer } from '@aws-lambda-powertools/kafka';
import type { ConsumerRecords } from '@aws-lambda-powertools/kafka/types';
import { Logger } from '@aws-lambda-powertools/logger';
import type { Context } from 'aws-lambda';
import { Product } from './product.generated.js';

const logger = new Logger();

type Value = {
    id: number;
    name: string;
    price: number;
};

export const handler = kafkaConsumer<string, Value>(
    (event: ConsumerRecords<string, Value>, _context: Context) => {
        for (const record of event.records) {

```

```

        logger.info(Processing record with key: ${record.key});
        logger.info(Record value: ${JSON.stringify(record.value)});
    }
},
{
    value: {
        type: 'protobuf',
        schema: Product,
    },
}
);

```

### esempio.NET:

```

using Amazon.Lambda.Core;
using AWS.Lambda.Powertools.Kafka;
using AWS.Lambda.Powertools.Kafka.Protobuf;
using AWS.Lambda.Powertools.Logging;
using Com.Example;

// Assembly attribute to enable the Lambda function's Kafka event to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(PowertoolsKafkaProtobufSerializer))]

namespace ProtoBufClassLibrary;

public class Function
{
    public string FunctionHandler(ConsumerRecords<string, CustomerProfile> records,
        ILambdaContext context)
    {
        foreach (var record in records)
        {
            Logger.LogInformation("Processing messagem from topic: {topic}",
                record.Topic);
            Logger.LogInformation("Partition: {partition}, Offset: {offset}",
                record.Partition, record.Offset);
            Logger.LogInformation("Produced at: {timestamp}", record.Timestamp);

            foreach (var header in record.Headers.DecodedValues())
            {
                Logger.LogInformation($"{header.Key}: {header.Value}");
            }
        }
    }
}

```



```

        Logger.LogInformation("Processing order for: {fullName}",
record.Value.FullName);
    }

    return "Processed " + records.Count() + " records";
}
}

```

## JSON

### Esempio Java:

```

package org.demo.kafka;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import software.amazon.lambda.powertools.kafka.Deserialization;
import software.amazon.lambda.powertools.kafka.DeserializationType;
import software.amazon.lambda.powertools.logging.Logging;

public class JsonDeserializationFunction implements
RequestHandler<ConsumerRecords<String, Product>, String> {

    private static final Logger LOGGER =
LoggerFactory.getLogger(JsonDeserializationFunction.class);

    @Override
    @Logging
    @Deserialization(type = DeserializationType.KAFKA_JSON)
    public String handleRequest(ConsumerRecords<String, Product> consumerRecords,
Context context) {
        for (ConsumerRecord<String, Product> consumerRecord : consumerRecords) {
            LOGGER.info("ConsumerRecord: {}", consumerRecord);

            Product product = consumerRecord.value();
            LOGGER.info("Product: {}", product);
        }
    }
}

```

```

        String key = consumerRecord.key();
        LOGGER.info("Key: {}", key);
    }

    return "OK";
}
}

```

### Esempio di Python:

```

from aws_lambda_powertools.utilities.kafka_consumer.kafka_consumer import
    kafka_consumer

from aws_lambda_powertools.utilities.kafka_consumer.schema_config import
    SchemaConfig
from aws_lambda_powertools.utilities.kafka_consumer.consumer_records import
    ConsumerRecords

from aws_lambda_powertools.utilities.typing import LambdaContext
from aws_lambda_powertools import Logger

logger = Logger(service="kafkaConsumerPowertools")

schema_config = SchemaConfig(value_schema_type="JSON")

@kafka_consumer(schema_config=schema_config)
def lambda_handler(event: ConsumerRecords, context: LambdaContext):

    for record in event.records:
        value = record.value
        logger.info(f"Received value: {value}")

```

### TypeScript esempio:

```

import { kafkaConsumer } from '@aws-lambda-powertools/kafka';
import type { ConsumerRecords } from '@aws-lambda-powertools/kafka/types';
import { Logger } from '@aws-lambda-powertools/logger';
import type { Context } from 'aws-lambda';

const logger = new Logger();

type Value = {
    id: number;

```

```

    name: string;
    price: number;
};

export const handler = kafkaConsumer<string, Value>(
  (event: ConsumerRecords<string, Value>, _context: Context) => {
    for (const record of event.records) {
      logger.info(Processing record with key: ${record.key});
      logger.info(Record value: ${JSON.stringify(record.value)});
      // You can add more processing logic here
    }
  },
  {
    value: {
      type: 'json',
    },
  }
);

```

### esempio.NET:

```

using Amazon.Lambda.Core;
using AWS.Lambda.Powertools.Kafka;
using AWS.Lambda.Powertools.Kafka.Json;
using AWS.Lambda.Powertools.Logging;
using Com.Example;

// Assembly attribute to enable the Lambda function's Kafka event to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(PowertoolsKafkaJsonSerializer))]

namespace JsonClassLibrary;

public class Function
{
    public string FunctionHandler(ConsumerRecords<string, CustomerProfile> records,
        ILambdaContext context)
    {
        foreach (var record in records)
        {
            Logger.LogInformation("Processing messagem from topic: {topic}",
                record.Topic);
        }
    }
}

```

```

        Logger.LogInformation("Partition: {partition}, Offset: {offset}",
record.Partition, record.Offset);
        Logger.LogInformation("Produced at: {timestamp}", record.Timestamp);

        foreach (var header in record.Headers.DecodedValues())
        {
            Logger.LogInformation($"{header.Key}: {header.Value}");
        }

        Logger.LogInformation("Processing order for: {fullName}",
record.Value.FullName);
    }

    return "Processed " + records.Count() + " records";
}
}

```

## Metodi di autenticazione per il registro degli schemi

Per utilizzare un registro di schemi, Lambda deve essere in grado di accedervi in modo sicuro. Se stai lavorando con un registro di AWS Glue schemi, Lambda si basa sull'autenticazione IAM. Ciò significa che il ruolo di [esecuzione della funzione](#) deve disporre delle seguenti autorizzazioni per accedere al registro: AWS Glue

- [GetRegistry](#) nel AWS Glue Web API Reference
- [GetSchemaVersion](#) nel AWS Glue Web API Reference

Esempio della policy IAM richiesta:

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetRegistry",
        "glue:GetSchemaVersion"
      ],
    }
  ]
}

```

```

    "Resource": [
      "*"
    ]
  }
]
}

```

#### Note

Per i registri di AWS Glue schema, se si fornisce un AWS Glue registro, Lambda restituisce un'AccessConfigsecezione di convalida.

Se utilizzi un registro di schemi Confluent, puoi scegliere uno dei tre metodi di autenticazione supportati per il parametro del tuo oggetto: Type [KafkaSchemaRegistryAccessConfig](#)

- **BASIC\_AUTH:** Lambda utilizza nome utente e password o API Key e autenticazione API Secret per accedere al registro. Se scegli questa opzione, fornisci l'ARN di Secrets Manager contenente le tue credenziali nel campo URI.
- **CLIENT\_CERTIFICATE\_TLS\_AUTH** — Lambda utilizza l'autenticazione TLS reciproca con i certificati client. Per utilizzare questa opzione, Lambda deve accedere sia al certificato che alla chiave privata. Fornisci l'ARN di Secrets Manager contenente queste credenziali nel campo URI.
- **NO\_AUTH:** il certificato emesso da una CA pubblica deve essere firmato da un'autorità di certificazione (CA) presente nel trust store di Lambda. Per un CA/self-signed certificato privato, si configura il certificato CA principale del server. Per utilizzare questa opzione, ometti il AccessConfigs parametro.

Inoltre, se Lambda ha bisogno di accedere a un certificato CA privato per verificare il certificato TLS del registro degli schemi, scegli **SERVER\_ROOT\_CA\_CERT** come valore Type e fornisci l'ARN di Secrets Manager al certificato nel campo URI.

#### Note

Per configurare l'**SERVER\_ROOT\_CA\_CERT** opzione nella console, fornisci l'ARN segreto contenente il certificato nel campo Encryption.

La configurazione di autenticazione per il registro degli schemi è separata da qualsiasi autenticazione configurata per il cluster Kafka. È necessario configurarli entrambi separatamente, anche se utilizzano metodi di autenticazione simili.

## Gestione degli errori e risoluzione dei problemi relativi al registro dello schema

Quando si utilizza un registro di schemi con la sorgente di eventi Amazon MSK, è possibile che si verifichino diversi errori. Questa sezione fornisce indicazioni sui problemi più comuni e su come risolverli.

### Errori di configurazione

Questi errori si verificano durante l'impostazione della configurazione del registro dello schema.

È richiesta la modalità con provisioning

Messaggio di errore: `SchemaRegistryConfig` is only available for Provisioned Mode. To configure Schema Registry, please enable Provisioned Mode by specifying `MinimumPollers` in `ProvisionedPollerConfig`.

Risoluzione: abilita la modalità predisposta per la mappatura della sorgente degli eventi configurando il parametro in `MinimumPollers` `ProvisionedPollerConfig`

### URL del registro dello schema non valido

Messaggio di errore: `Malformed SchemaRegistryURI provided. Please provide a valid URI or ARN. For example, https://schema-registry.example.com:8081 or arn:aws:glue:us-east-1:123456789012:registry/ExampleRegistry.`

Risoluzione: fornire un URL HTTPS valido per Confluent Schema Registry o un ARN valido per Schema Registry. AWS Glue

### Formato di registrazione degli eventi non valido o mancante

Messaggio di errore: `EventRecordFormat` is a required field for `SchemaRegistryConfig`. Please provide one of supported format types: `SOURCE`, `JSON`.

Risoluzione: specificare `SOURCE` o `JSON` come valore `EventRecordFormat` nella configurazione del registro dello schema.

## Attributi di convalida duplicati

Messaggio di errore: Duplicate KEY/VALUE Attribute in SchemaValidationConfigs. SchemaValidationConfigs must contain at most one KEY/VALUE Attribute.

Risoluzione: rimuovi gli attributi KEY o VALUE duplicati dal tuo. SchemaValidationConfigs Ogni tipo di attributo può apparire una sola volta.

## Configurazione di convalida mancante

Messaggio di errore: SchemaValidationConfigs is a required field for SchemaRegistryConfig.

Risoluzione: aggiungi SchemaValidationConfigs alla tua configurazione, specificando almeno un attributo di convalida (KEY o VALUE).

## Errori di accesso e autorizzazione

Questi errori si verificano quando Lambda non può accedere al registro dello schema a causa di problemi di autorizzazione o autenticazione.

### AWS Glue Accesso allo Schema Registry negato

Messaggio di errore: Cannot access Glue Schema with provided role. Please ensure the provided role can perform the GetRegistry and GetSchemaVersion Actions on your schema.

Risoluzione: aggiungi le autorizzazioni necessarie (glue:GetRegistry e glue:GetSchemaVersion) per il ruolo di esecuzione di una funzione.

### Accesso al Confluent Schema Registry negato

Messaggio di errore: Cannot access Confluent Schema with the provided access configuration.

Risoluzione: verifica che le tue credenziali di autenticazione (archivate in Secrets Manager) siano corrette e dispongano delle autorizzazioni necessarie per accedere al registro degli schemi.

### Cross-account AWS Glue Registro degli schemi

Messaggio di errore: Cross-account Glue Schema Registry ARN not supported.

Risoluzione: utilizza un registro degli AWS Glue schemi che si trova nello stesso AWS account della funzione Lambda.

### Cross-region AWS Glue Registro degli schemi

Messaggio di errore: `Cross-region Glue Schema Registry ARN not supported.`

Risoluzione: utilizza un registro degli AWS Glue schemi che si trova nella stessa regione della funzione Lambda.

### Problemi di accesso segreto

Messaggio di errore: `Lambda received InvalidRequestException from Secrets Manager.`

Risoluzione: verifica che il ruolo di esecuzione della tua funzione disponga dell'autorizzazione per accedere al segreto e che il segreto non sia crittografato con una AWS KMS chiave predefinita se accedi da un account diverso.

### Errori di connessione

Questi errori si verificano quando Lambda non è in grado di stabilire una connessione al registro degli schemi.

### Problemi di connettività VPC

Messaggio di errore: `Cannot connect to your Schema Registry. Your Kafka cluster's VPC must be able to connect to the schema registry. You can provide access by configuring AWS PrivateLink or a NAT Gateway or VPC Peering between Kafka Cluster VPC and the schema registry VPC.`

Risoluzione: configura la tua rete VPC per consentire le connessioni al registro dello schema utilizzando AWS PrivateLink un gateway NAT o un peering VPC.

### Errore di handshake TLS

Messaggio di errore: `Unable to establish TLS handshake with the schema registry. Please provide correct CA-certificate or client certificate using Secrets Manager to access your schema registry.`

Risoluzione: verifica che i certificati CA e i certificati client (per MTL) siano corretti e configurati correttamente in Secrets Manager.



## Throttling

Messaggio di errore: `Receiving throttling errors when accessing the schema registry. Please increase API TPS limits for your schema registry.`

Risoluzione: aumenta i limiti di velocità delle API per il registro degli schemi o riduci il tasso di richieste dall'applicazione.

## Self-managed errori del registro dello schema

Messaggio di errore: `Lambda received an internal server an unexpected error from the provided self-managed schema registry.`

Risoluzione: controlla lo stato e la configurazione del tuo server di registro degli schemi autogestito.

## Elaborazione a bassa latenza per sorgenti di eventi Kafka

AWS Lambda supporta nativamente l'elaborazione di eventi a bassa latenza per applicazioni che richiedono latenze end-to-end costanti inferiori a 100 millisecondi. Questa pagina fornisce dettagli di configurazione e consigli per abilitare flussi di lavoro a bassa latenza.

### Abilita l'elaborazione a bassa latenza

Per abilitare l'elaborazione a bassa latenza sulla mappatura della sorgente di un evento Kafka, è necessaria la seguente configurazione di base:

- Abilita la modalità predisposta. Per ulteriori informazioni, consulta [Modalità provisioning](#).
- Imposta il `MaximumBatchingWindowInSeconds` parametro della mappatura della sorgente dell'evento su 0. Per ulteriori informazioni, consulta [Comportamento di batching](#).

### Fine-tuning il tuo Kafka ESM a bassa latenza

Considerate i seguenti consigli per ottimizzare la mappatura della sorgente degli eventi Kafka per una bassa latenza:

#### Configurazione della modalità con provisioning

In modalità provisioned per la mappatura delle sorgenti di eventi di Kafka, è possibile ottimizzare il throughput della mappatura delle sorgenti di eventi configurando un numero minimo e massimo di risorse denominate event poller. Un event poller (o poller) rappresenta una risorsa di calcolo che è alla base di una mappatura della sorgente di eventi in modalità provisioned e alloca fino a 5 velocità effettiva. MB/s Ogni event poller supporta fino a 5 invocazioni Lambda simultanee.

Per determinare la configurazione del poller ottimale per la tua applicazione, considera la frequenza massima di ingestione e i requisiti di elaborazione. Diamo un'occhiata a un esempio semplificato:

Con una dimensione batch di 20 record e una durata media della funzione target di 50 ms, ogni poller può gestire 2.000 record al secondo con un limite di 5. MB/s Viene calcolato come:  $(20 \text{ record} \times 1000\text{ms}/50\text{ms}) \times 5$  chiamate Lambda simultanee. Pertanto, se la velocità massima di acquisizione desiderata è di 20.000 record al secondo, sono necessari almeno 10 poller di eventi.

#### Note

Raccomandiamo di fornire poller di eventi aggiuntivi come buffer per evitare di operare costantemente alla massima capacità.

La modalità provisioned ridimensiona automaticamente i poller di eventi in base ai modelli di traffico entro i poller di eventi minimi e massimi configurati, il che può innescare un ribilanciamento e quindi introdurre una latenza aggiuntiva. È possibile disattivare il ridimensionamento automatico configurando lo stesso valore per il poller di eventi minimo e massimo.

### Ulteriori considerazioni

Alcune delle considerazioni aggiuntive includono:

- Le partenze a freddo partendo dall'invocazione della funzione target Lambda possono potenzialmente aumentare la latenza end-to-end. Per ridurre questo rischio, prendi in considerazione la possibilità di abilitare la coincidenza su [provisioning](#) o [SnapStart](#) sulla funzione target della mappatura della sorgente dell'evento. Inoltre, ottimizzate l'allocazione di memoria della funzione per garantire esecuzioni coerenti e ottimali.
- Quando `MaximumBatchingWindowInSeconds` è impostato su 0, Lambda elabora immediatamente tutti i record disponibili senza attendere di riempire la dimensione completa del batch. Ad esempio, se la dimensione del batch è impostata su 1.000 record ma sono disponibili solo 100 record, Lambda elabora immediatamente quei 100 record anziché attendere l'accumulo dei 1.000 record completi.

#### Important

La configurazione ottimale per l'elaborazione a bassa latenza varia notevolmente in base al carico di lavoro specifico. Consigliamo vivamente di testare diverse configurazioni con il carico di lavoro effettivo per determinare le impostazioni migliori per il tuo caso d'uso.

## Configurazione dei controlli di gestione degli errori per le sorgenti di eventi Kafka

Puoi configurare il modo in cui Lambda gestisce gli errori e i tentativi per le mappature delle sorgenti degli eventi Kafka. Queste configurazioni ti aiutano a controllare il modo in cui Lambda elabora i record non riusciti e gestisce il comportamento dei nuovi tentativi.

### Configurazioni di nuovo tentativo disponibili

Le seguenti configurazioni di ripetizione sono disponibili sia per Amazon MSK che per le sorgenti di eventi Kafka autogestite:

- Numero massimo di tentativi di ripetizione: il numero massimo di tentativi di Lambda quando la funzione restituisce un errore. Questo non conta il tentativo di richiamo iniziale. L'impostazione predefinita è -1 (infinito). Quando si configurano sia tentativi infiniti che una destinazione in [caso di errore](#), Lambda applica automaticamente un massimo di 10 tentativi di nuovo tentativo.
- Età massima del record: l'età massima di un record che Lambda invia alla tua funzione. L'impostazione predefinita è -1 (infinito).
- Dividi il batch in caso di errore: quando la funzione restituisce un errore, suddividi il batch in due batch più piccoli e riprova ciascuno separatamente. Questo aiuta a isolare i record problematici.
- Risposta parziale in batch: consenti alla funzione di restituire informazioni su quali record di un batch non sono stati elaborati, in modo che Lambda possa riprovare solo i record non riusciti.

### Configurazione dei controlli di gestione degli errori (console)

È possibile configurare il comportamento dei nuovi tentativi durante la creazione o l'aggiornamento di una mappatura della sorgente di eventi Kafka nella console Lambda.

Per configurare il comportamento dei tentativi di ripetizione per una sorgente di eventi Kafka (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della tua funzione.
3. Esegui una delle seguenti operazioni:
  - Per aggiungere un nuovo trigger Kafka, in Panoramica delle funzioni, scegli **Aggiungi trigger**.
  - Per modificare un trigger Kafka esistente, scegli il trigger e quindi scegli **Modifica**.
4. In **Event poller configuration**, seleziona la modalità **provisioned** per configurare i controlli di gestione degli errori:

- a. Per i tentativi di riprova, inserisci il numero massimo di tentativi (0-10000 o -1 per infinito).
  - b. Per Età massima del record, inserisci l'età massima in secondi (60-604800 o -1 per infinito).
  - c. Per abilitare la suddivisione in batch in caso di errori, seleziona `Dividi batch in caso di errore`.
  - d. Per abilitare la risposta parziale in batch, selezionare `ReportBatchItemFailures`.
5. Scegli `Aggiungi` o `Salva`.

## Configurazione del comportamento dei tentativi (AWS CLI)

Utilizzate i seguenti AWS CLI comandi per configurare il comportamento dei tentativi per le mappature delle sorgenti degli eventi Kafka.

### Creazione di una mappatura della sorgente di eventi con configurazioni di ripetizione

L'esempio seguente crea una mappatura autogestita della sorgente di eventi Kafka con controlli di gestione degli errori:

```
aws lambda create-event-source-mapping \  
  --function-name my-kafka-function \  
  --topics my-kafka-topic \  
  --source-access-configuration Type=SASL_SCRAM_512_AUTH,URI=arn:aws:secretsmanager:us-  
east-1:111122223333:secret:MyBrokerSecretName \  
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":  
["abc.xyz.com:9092"]}}' \  
  --starting-position LATEST \  
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=1 \  
  --maximum-retry-attempts 3 \  
  --maximum-record-age-in-seconds 3600 \  
  --bisect-batch-on-function-error \  
  --function-response-types "ReportBatchItemFailures"
```

Per le sorgenti di eventi Amazon MSK:

```
aws lambda create-event-source-mapping \  
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-  
fd1b-45ad-85dd-15b4a5a6247e-2 \  
  --topics AWSMSKKafkaTopic \  
  --starting-position LATEST \  
  --function-name my-kafka-function \  

```

```
--source-access-configurations '[{"Type": "SASL_SCRAM_512_AUTH", "URI":
"arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret"}]' \
--provisioned-poller-config MinimumPollers=1,MaximumPollers=1 \
--maximum-retry-attempts 3 \
--maximum-record-age-in-seconds 3600 \
--bisect-batch-on-function-error \
--function-response-types "ReportBatchItemFailures"
```

Aggiornamento delle configurazioni dei nuovi tentativi

Utilizzate il `update-event-source-mapping` comando per modificare le configurazioni dei nuovi tentativi per una mappatura della sorgente di eventi esistente:

```
aws lambda update-event-source-mapping \
--uuid 12345678-1234-1234-1234-123456789012 \
--maximum-retry-attempts 5 \
--maximum-record-age-in-seconds 7200 \
--bisect-batch-on-function-error \
--function-response-types "ReportBatchItemFailures"
```

## PartialBatchResponse

La risposta parziale in batch, nota anche come `ReportBatchItemFailures`, è una funzionalità chiave per la gestione degli errori nell'integrazione di Lambda con i sorgenti Kafka. Senza questa funzionalità, quando si verifica un errore in uno degli elementi di un batch, comporta la rielaborazione di tutti i messaggi in quel batch. Con la risposta parziale in batch abilitata e implementata, il gestore restituisce gli identificatori solo per i messaggi non riusciti, consentendo a Lambda di riprovare solo quegli elementi specifici. Ciò fornisce un maggiore controllo sul modo in cui vengono elaborati i batch contenenti messaggi non riusciti.

Per segnalare errori batch, utilizzate questo schema JSON:

```
{
  "batchItemFailures": [
    {
      "itemIdentifier": {
        "partition": "topic-partition_number",
        "offset": 100
      }
    },
    ...
  ]
}
```

```
]
}
```

### Important

Se restituisci un JSON o null valido vuoto, la mappatura dell'origine dell'evento considera un batch come elaborato correttamente. Qualsiasi topic-partition\_number o offset restituito non valido che non era presente nell'evento richiamato viene considerato un errore e l'intero batch viene riprovato.

I seguenti esempi di codice mostrano come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da sorgenti Kafka. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

Ecco un'implementazione del gestore Python Lambda che mostra questo approccio:

```
import base64
from typing import Any, Dict, List

def lambda_handler(event: Dict[str, Any], context: Any) -> Dict[str, List[Dict[str, Dict[str, Any]]]]:
    failures: List[Dict[str, Dict[str, Any]]] = []
    records_dict = event.get("records", {})

    for topic_partition, records_list in records_dict.items():
        for record in records_list:
            topic = record.get("topic")
            partition = record.get("partition")
            offset = record.get("offset")
            value_b64 = record.get("value")

            try:
                data = base64.b64decode(value_b64).decode("utf-8")
                process_message(data)
            except Exception as exc:
                print(f"Failed to process record topic={topic} partition={partition} offset={offset}: {exc}")
                item_identifier: Dict[str, Any] = {
                    "partition": f"{topic}-{partition}",
                    "offset": int(offset) if offset is not None else None,
                }
```

```

        failures.append({"itemIdentifier": item_identifier})

    return {"batchItemFailures": failures}

def process_message(data: str) -> None:
    # Your business logic for a single message
    pass

```

Ecco una versione: Node.js

```

const { Buffer } = require("buffer");

const handler = async (event) => {
    const failures = [];

    for (let topicPartition in event.records) {
        const records = event.records[topicPartition];

        for (const record of records) {
            const topic = record.topic;
            const partition = record.partition;
            const offset = record.offset;
            const valueBase64 = record.value;
            const data = Buffer.from(valueBase64, "base64").toString("utf8");

            try {
                await processMessage(data);
            } catch (error) {
                console.error("Failed to process record", { topic, partition, offset, error });
                const itemIdentifier = {
                    "partition": `${topic}-${partition}`,
                    "offset": Number(offset),
                };
                failures.push({ itemIdentifier });
            }
        }
    }

    return { batchItemFailures: failures };
};

async function processMessage(payload) {
    // Your business logic for a single message

```



```
}

module.exports = { handler };
```

Ecco una versione Java:

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import java.util.ArrayList;
import java.util.Base64;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class KafkaBatchHandler implements RequestHandler<Map<String, Object>,
    Map<String, Object>> {

    @SuppressWarnings("unchecked")
    @Override
    public Map<String, Object> handleRequest(Map<String, Object> event, Context
context) {
        List<Map<String, Object>> failures = new ArrayList<>();
        Map<String, List<Map<String, Object>>> records =
            (Map<String, List<Map<String, Object>>>) event.getDefault("records",
Map.of());

        for (Map.Entry<String, List<Map<String, Object>>> entry : records.entrySet()) {
            for (Map<String, Object> record : entry.getValue()) {
                String topic = (String) record.get("topic");
                Object partition = record.get("partition");
                Object offset = record.get("offset");
                String valueBase64 = (String) record.get("value");

                try {
                    String data = new String(Base64.getDecoder().decode(valueBase64),
"UTF-8");

                    processMessage(data);
                } catch (Exception e) {
                    System.err.printf("Failed to process record topic=%s partition=%s
offset=%s: %s%n",
                                topic, partition, offset, e.getMessage());
                    Map<String, Object> itemIdentifier = new HashMap<>();
```

```
        itemIdentifier.put("partition", topic + "-" + partition);
        itemIdentifier.put("offset", offset instanceof Number ? ((Number)
offset).longValue() : null);
        Map<String, Object> failure = new HashMap<>();
        failure.put("itemIdentifier", itemIdentifier);
        failures.add(failure);
    }
}

Map<String, Object> response = new HashMap<>();
response.put("batchItemFailures", failures);
return response;
}

private void processMessage(String data) {
    // Your business logic for a single message
}
}
```

## Acquisizione di batch scartati per un'origine eventi di Amazon MSK e Apache Kafka autogestito

Per mantenere i record delle chiamate non riuscite allo strumento di mappatura dell'origine degli eventi, aggiungi una destinazione allo strumento di mappatura dell'origine degli eventi della funzione. Ogni record inviato alla destinazione è un documento JSON contenente i metadati relativi alla chiamata non riuscita. Per le destinazioni Amazon S3, Lambda invia insieme ai metadati anche l'intero record di invocazione. Puoi configurare qualsiasi argomento Amazon SNS, coda Amazon SQS, bucket Amazon S3 o Kafka come destinazione.

Con le destinazioni Amazon S3, puoi utilizzare la funzionalità [Notifiche eventi Amazon S3](#) per ricevere notifiche quando gli oggetti vengono caricati nel bucket S3 di destinazione. Puoi anche configurare Notifiche eventi S3 per richiamare un'altra funzione Lambda per eseguire l'elaborazione automatica su batch non riusciti.

Il tuo ruolo di esecuzione deve avere le autorizzazioni per la destinazione:

- [Per una destinazione SQS: sqs: SendMessage](#)
- [Per una destinazione SNS: sns:publish](#)
- [Per una destinazione S3: s3: e s3: PutObject ListBucket](#)
- [Per una destinazione Kafka: kafka-cluster: WriteData](#)

È possibile configurare un argomento di Kafka come destinazione in caso di errore per le mappature delle sorgenti degli eventi Kafka. Quando Lambda non è in grado di elaborare i record dopo estenuanti tentativi o quando i record superano l'età massima, Lambda invia i record non riusciti all'argomento Kafka specificato per un'elaborazione successiva. Fai riferimento a [the section called "Destinazione Kafka in caso di errore"](#).

È necessario implementare un endpoint VPC per il servizio di destinazione in caso di errore all'interno del VPC del cluster Kafka.

Inoltre, se hai configurato una chiave KMS sulla destinazione, Lambda necessita delle seguenti autorizzazioni, a seconda del tipo di destinazione:

- [Se hai abilitato la crittografia con la tua chiave KMS per una destinazione S3, è richiesto kms: GenerateDataKey](#) Se la chiave KMS e la destinazione del bucket S3 si trovano in un account diverso dalla funzione Lambda e dal ruolo di esecuzione, configura la chiave KMS in modo che consideri attendibile il ruolo di esecuzione da consentire. kms: GenerateDataKey

- [Se hai abilitato la crittografia con la tua chiave KMS per la destinazione SQS, sono necessari KMS:decrypt e kms: GenerateDataKey](#) [Se la chiave KMS e la destinazione della coda SQS si trovano in un account diverso dalla funzione Lambda e dal ruolo di esecuzione, configura la chiave KMS in modo che consideri attendibile il ruolo di esecuzione per consentire kms: Decrypt,, kms: GenerateDataKey kms: e kms: DescribeKey ReEncrypt](#)
- [Se hai abilitato la crittografia con la tua chiave KMS per la destinazione SNS, sono obbligatori KMS:Decrypt e kms: GenerateDataKey](#) [Se la chiave KMS e la destinazione dell'argomento SNS si trovano in un account diverso dalla funzione Lambda e dal ruolo di esecuzione, configura la chiave KMS in modo che consideri attendibile il ruolo di esecuzione per consentire kms: Decrypt, kms:GenerateDataKey, kms: e kms: DescribeKey ReEncrypt](#)

Configurazione delle destinazioni in caso di errore per una mappatura della sorgente degli eventi Kafka

Per configurare una destinazione in caso di errore tramite la console, completa i seguenti passaggi:

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. In Function overview (Panoramica delle funzioni), scegliere Add destination (Aggiungi destinazione).
4. Per Origine, scegli Chiamata allo strumento di mappatura dell'origine degli eventi.
5. Per Strumento di mappatura dell'origine degli eventi, scegli un'origine dell'evento configurata per questa funzione.
6. Per Condizione, seleziona In caso di errore. Per le chiamate allo strumento di mappatura dell'origine degli eventi, questa è l'unica condizione accettata.
7. Per Tipo di destinazione, scegli il tipo di destinazione a cui Lambda deve inviare i record di chiamata.
8. Per Destination (Destinazione), scegliere una risorsa.
9. Scegli Save (Salva).

È inoltre possibile configurare una destinazione in errore utilizzando la AWS CLI. Ad esempio, il [create-event-source-mapping](#) comando seguente aggiunge una mappatura dell'origine degli eventi con una destinazione SQS in caso di errore a: MyFunction

```
aws lambda create-event-source-mapping \
```

```
--function-name "MyFunction" \
--event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2 \
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-
east-1:123456789012:dest-queue"}}'
```

Il [update-event-source-mapping](#) comando seguente aggiunge una destinazione S3 in caso di errore all'origine dell'evento associata all'input: uuid

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": "arn:aws:s3:::dest-bucket"}}'
```

Per rimuovere una destinazione, fornisci una stringa vuota come argomento del parametro `destination-config`:

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

### Best practice di sicurezza per destinazioni Amazon S3

L'eliminazione di un bucket S3 configurato come destinazione senza rimuovere la destinazione dalla configurazione della funzione può creare un rischio per la sicurezza. Se un altro utente conosce il nome del bucket di destinazione, può ricreare il bucket nel proprio Account AWS. I record delle invocazioni non riuscite verranno inviati al relativo bucket, esponendo potenzialmente i dati della tua funzione.

#### Warning

Per garantire che i record di invocazione della tua funzione non possano essere inviati a un bucket S3 in un altro Account AWS, aggiungi una condizione al ruolo di esecuzione della funzione che limiti le `s3:PutObject` autorizzazioni ai bucket del tuo account.

Di seguito viene illustrato un esempio di policy IAM che limita le autorizzazioni `s3:PutObject` della funzione ai bucket presenti nell'account. Questa policy fornisce inoltre a Lambda l'autorizzazione `s3:ListBucket` necessaria per utilizzare un bucket S3 come destinazione.

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Sid": "S3BucketResourceAccountWrite",
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3:::*/**",
      "arn:aws:s3:::*"
    ],
    "Condition": {
      "StringEquals": {
        "s3:ResourceAccount": "111122223333"
      }
    }
  }
]
```

Per aggiungere una politica di autorizzazioni al ruolo di esecuzione della tua funzione utilizzando Console di gestione AWS o AWS CLI, consulta le istruzioni nelle seguenti procedure:

## Console

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Seleziona la funzione Lambda di cui si desidera modificare il ruolo di esecuzione.
3. Nella scheda Configurazione scegli Autorizzazioni.
4. Nella scheda Ruolo di esecuzione, seleziona il nome del ruolo della funzione per aprire la pagina della console IAM del ruolo.
5. Aggiungi una policy di autorizzazioni di base al ruolo completando le seguenti operazioni:
  - a. Nel riquadro Policy di autorizzazioni, scegli Aggiungi autorizzazioni, poi Crea policy in linea.
  - b. Nell'editor delle policy, seleziona JSON.

- c. Incolla la policy che desideri aggiungere nell'editor (sostituendo il codice JSON esistente), quindi scegli Avanti.
- d. In Dettagli della policy, specifica un nome per la policy.
- e. Scegli Crea policy.

## AWS CLI

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (CLI)

1. Crea un documento di policy JSON con le autorizzazioni richieste e salvalo in una directory locale.
2. Utilizza il comando della CLI `put-role-policy` di IAM per aggiungere le autorizzazioni per il ruolo di esecuzione di una funzione. Esegui il comando seguente dalla directory in cui hai salvato il documento di policy JSON e sostituisci il nome del ruolo, il nome della policy e il documento di policy con i tuoi valori.

```
aws iam put-role-policy \
--role-name my_lambda_role \
--policy-name LambdaS3DestinationPolicy \
--policy-document file://my_policy.json
```

## Record di invocazione SNS e SQS di esempio

L'esempio seguente mostra il contenuto che Lambda invia a un argomento SNS o una coda SQS di destinazione per una chiamata non riuscita all'origine dell'evento Kafka. Ciascuna delle chiavi in `recordsInfo` contiene sia l'argomento sia lo shard di Kafka, separati da un trattino. Ad esempio, per la chiave `"Topic-0"`, `Topic` è l'argomento di Kafka e `0` è lo shard. Per ogni argomento e partizione, è possibile utilizzare i dati di offset e timestamp per individuare i record di chiamata originali.

```
{
  "requestContext": {
    "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction",
    "condition": "RetryAttemptsExhausted" | "MaximumPayloadSizeExceeded",
    "approximateInvokeCount": 1
  },
  "responseContext": { // null if record is MaximumPayloadSizeExceeded
```

```

    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Unhandled"
  },
  "version": "1.0",
  "timestamp": "2019-11-14T00:38:06.021Z",
  "KafkaBatchInfo": {
    "batchSize": 500,
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
    "bootstrapServers": "...",
    "payloadSize": 2039086, // In bytes
    "recordsInfo": {
      "Topic-0": {
        "firstRecordOffset":
"49601189658422359378836298521827638475320189012309704722",
        "lastRecordOffset":
"49601189658422359378836298522902373528957594348623495186",
        "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
        "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
      },
      "Topic-1": {
        "firstRecordOffset":
"49601189658422359378836298521827638475320189012309704722",
        "lastRecordOffset":
"49601189658422359378836298522902373528957594348623495186",
        "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
        "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
      }
    }
  }
}

```

## Record di invocazione di esempio di destinazione S3

Se le destinazioni sono S3, Lambda invia alla destinazione l'intero record di chiamata insieme ai metadati. L'esempio seguente mostra ciò che Lambda invia a un bucket S3 di destinazione per una chiamata non riuscita all'origine dell'evento Kafka. Oltre a tutti i campi dell'esempio precedente per le destinazioni SQS e SNS, il campo `payload` contiene il record di chiamata originale come stringa JSON con escape.

```

{
  "requestContext": {

```



```

    "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction",
    "condition": "RetryAttemptsExhausted" | "MaximumPayloadSizeExceeded",
    "approximateInvokeCount": 1
  },
  "responseContext": { // null if record is MaximumPayloadSizeExceeded
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Unhandled"
  },
  "version": "1.0",
  "timestamp": "2019-11-14T00:38:06.021Z",
  "KafkaBatchInfo": {
    "batchSize": 500,
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
    "bootstrapServers": "...",
    "payloadSize": 2039086, // In bytes
    "recordsInfo": {
      "Topic-0": {
        "firstRecordOffset":
"49601189658422359378836298521827638475320189012309704722",
        "lastRecordOffset":
"49601189658422359378836298522902373528957594348623495186",
        "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
        "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
      },
      "Topic-1": {
        "firstRecordOffset":
"49601189658422359378836298521827638475320189012309704722",
        "lastRecordOffset":
"49601189658422359378836298522902373528957594348623495186",
        "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
        "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
      }
    }
  },
  "payload": "<Whole Event>" // Only available in S3
}

```

 Tip

Ti consigliamo di abilitare il controllo delle versioni S3 sul bucket di destinazione.

## Usare un argomento di Kafka come destinazione in caso di errore

È possibile configurare un argomento di Kafka come destinazione in caso di errore per le mappature delle sorgenti degli eventi Kafka. Quando Lambda non è in grado di elaborare i record dopo estenuanti tentativi o quando i record superano l'età massima, Lambda invia i record non riusciti all'argomento Kafka specificato per un'elaborazione successiva. Quando configuri sia [tentativi infiniti](#) che una destinazione in caso di errore, Lambda applica automaticamente un massimo di 10 tentativi.

### Come funziona una destinazione Kafka in caso di errore

Quando si configura un argomento di Kafka come destinazione in caso di errore, Lambda funge da produttore Kafka e scrive i record non riusciti nell'argomento di destinazione. Questo crea un pattern DLT (Dead Letter Topic) all'interno dell'infrastruttura Kafka.

- Stesso requisito del cluster: l'argomento di destinazione deve esistere nello stesso cluster Kafka degli argomenti di origine.
- Contenuto effettivo dei record: le destinazioni Kafka ricevono i record effettivamente non riusciti insieme ai metadati di errore.
- Prevenzione della ricorsione: Lambda previene i loop infiniti bloccando le configurazioni in cui gli argomenti di origine e destinazione coincidono.

### Configurazione di una destinazione Kafka in caso di errore

È possibile configurare un argomento di Kafka come destinazione in caso di errore durante la creazione o l'aggiornamento di una mappatura dell'origine degli eventi Kafka.

### Configurazione di una destinazione Kafka (console)

#### Configurare un argomento di Kafka come destinazione in caso di errore (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliete il nome della funzione.
3. Esegui una delle seguenti operazioni:
  - Per aggiungere un nuovo trigger Kafka, in Panoramica delle funzioni, scegli Aggiungi trigger.
  - Per modificare un trigger Kafka esistente, scegli il trigger, quindi scegli Modifica.
4. In Impostazioni aggiuntive, per Destinazione in caso di errore, scegli l'argomento Kafka.

5. Per Nome argomento, inserisci il nome dell'argomento Kafka a cui desideri inviare i record non riusciti.
6. Scegli Aggiungi o Salva.

## Configurazione di una destinazione Kafka (AWS CLI)

Utilizzate il `kafka://` prefisso per specificare un argomento di Kafka come destinazione in caso di errore.

Creazione di una mappatura della sorgente degli eventi con la destinazione Kafka

L'esempio seguente crea una mappatura dell'origine degli eventi Amazon MSK con un argomento Kafka come destinazione in caso di errore:

```
aws lambda create-event-source-mapping \  
  --function-name my-kafka-function \  
  --topics AWSKafkaTopic \  
  --event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/my-cluster/abc123 \  
  --starting-position LATEST \  
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=3 \  
  --destination-config '{"OnFailure":{"Destination":"kafka://failed-records-topic"}}'
```

Per Kafka autogestito, usa la stessa sintassi:

```
aws lambda create-event-source-mapping \  
  --function-name my-kafka-function \  
  --topics AWSKafkaTopic \  
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":  
["abc.xyz.com:9092"]}}' \  
  --starting-position LATEST \  
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=3 \  
  --destination-config '{"OnFailure":{"Destination":"kafka://failed-records-topic"}}'
```

## Aggiornamento di una destinazione Kafka

Usa il `update-event-source-mapping` comando per aggiungere o modificare una destinazione Kafka:

```
aws lambda update-event-source-mapping \  
  --uuid 12345678-1234-1234-1234-123456789012 \  
  --function-name my-kafka-function
```

```
--destination-config '{"OnFailure":{"Destination":"kafka://failed-records-topic"}}'
```

## Formato di registrazione per una destinazione Kafka

Quando Lambda invia record non riusciti a un argomento di Kafka, ogni messaggio contiene sia metadati sull'errore che il contenuto effettivo del record.

### Metadati di errore

I metadati includono informazioni sul motivo per cui il record ha avuto esito negativo e dettagli sul batch originale:

```
{
  "requestContext": {
    "requestId": "e4b46cbf-b738-xmpl-8880-a18cdf61200e",
    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:
$LATEST",
    "condition": "RetriesExhausted",
    "approximateInvokeCount": 3
  },
  "responseContext": {
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Unhandled"
  },
  "version": "1.0",
  "timestamp": "2019-11-14T18:16:05.568Z",
  "KafkaBatchInfo": {
    "batchSize": 1,
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-cluster/abc123",
    "bootstrapServers": "b-1.mycluster.abc123.kafka.us-east-1.amazonaws.com:9098",
    "payloadSize": 1162,
    "recordInfo": {
      "offset": "49601189658422359378836298521827638475320189012309704722",
      "timestamp": "2019-11-14T18:16:04.835Z"
    }
  },
  "payload": {
    "bootstrapServers": "b-1.mycluster.abc123.kafka.us-east-1.amazonaws.com:9098",
    "eventSource": "aws:kafka",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-cluster/abc123",
    "records": {
      "my-topic-0": [
```

```

{
  "headers": [],
  "key": "dGVzdC1rZXk=",
  "offset": 100,
  "partition": 0,
  "timestamp": 1749116692330,
  "timestampType": "CREATE_TIME",
  "topic": "my-topic",
  "value": "dGVzdC12YWx1ZQ=="
}
]
}
}
}

```

## Comportamento delle chiavi di partizione

Lambda utilizza la stessa chiave di partizione del record originale per la produzione sull'argomento di destinazione. Se il record originale non aveva alcuna chiave, Lambda utilizza il partizionamento round-robin predefinito di Kafka su tutte le partizioni disponibili nell'argomento di destinazione.

## Requisiti e limitazioni

- È richiesta la modalità di provisioning: una destinazione Kafka in caso di errore è disponibile solo per le mappature delle sorgenti degli eventi con la modalità provisioning abilitata.
- Solo stesso cluster: l'argomento di destinazione deve esistere nello stesso cluster Kafka degli argomenti di origine.
- Autorizzazioni per gli argomenti: la mappatura dei sorgenti degli eventi deve disporre delle autorizzazioni di scrittura sull'argomento di destinazione. Esempio:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ClusterPermissions",
      "Effect": "Allow",
      "Action": [
        "kafka-cluster:Connect",
        "kafka-cluster:DescribeCluster",
        "kafka-cluster:DescribeTopic",
        "kafka-cluster:WriteData",
        "kafka-cluster:ReadData"
      ]
    }
  ]
}

```

```

    ],
    "Resource": [
        "arn:aws:kafka:*:*:cluster/*"
    ]
},
{
    "Sid": "TopicPermissions",
    "Effect": "Allow",
    "Action": [
        "kafka-cluster:DescribeTopic",
        "kafka-cluster:WriteData",
        "kafka-cluster:ReadData"
    ],
    "Resource": [
        "arn:aws:kafka:*:*:topic/*/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "kafka:DescribeCluster",
        "kafka:GetBootstrapBrokers",
        "kafka:Produce"
    ],
    "Resource": "arn:aws:kafka:*:*:cluster/*"
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2>DeleteNetworkInterface",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
    ],
    "Resource": "*"
}
]
}

```

- Nessuna ricorsione: il nome dell'argomento di destinazione non può essere lo stesso dei nomi degli argomenti di origine.

## Registrazione della mappatura delle sorgenti degli eventi Kafka

È possibile configurare la registrazione a livello di sistema per le mappature delle sorgenti di eventi Kafka per abilitare e filtrare i log di sistema a cui inviano i poller di eventi Lambda. CloudWatch

Questa funzionalità è disponibile solo per le mappature delle sorgenti di eventi Kafka e con la modalità Provisioned. <https://docs.aws.amazon.com/lambda/latest/dg/kafka-scaling-modes.html#kafka-provisioned-mode>

Per la mappatura delle sorgenti degli eventi con configurazione di registrazione, puoi anche controllare i log di sistema provenienti da query di registro predefinite nella scheda Monitor dalla pagina Console Lambda > Risorse aggiuntive > mappature delle sorgenti di eventi ora.

Come funziona la registrazione

Quando imposti la configurazione di registrazione con il livello di log nella mappatura della sorgente degli eventi, il poller di eventi Lambda invia i log corrispondenti (registri del sistema di mappatura delle sorgenti degli eventi).

La mappatura dell'origine degli eventi riutilizza la stessa destinazione del registro con la funzione Lambda. <https://docs.aws.amazon.com/lambda/latest/dg/monitoring-logs.html#configuring-log-destinations> Assicurati che il ruolo di esecuzione della tua funzione Lambda disponga delle autorizzazioni di registrazione necessarie.

La mappatura dell'origine degli eventi avrà un proprio flusso di log, con l'UUID di mappatura della data e dell'origine dell'evento come nome del flusso di log, ad esempio.  
2020/01/01/12345678-1234-1234-1234-12345678901

Per i registri del sistema di mappatura delle sorgenti degli eventi, puoi scegliere tra i seguenti livelli di registro.

| Livello di log            | Utilizzo                                                                             |
|---------------------------|--------------------------------------------------------------------------------------|
| DEBUG (dettaglio massimo) | Informazioni dettagliate sull'avanzamento dell'elaborazione dell'origine dell'evento |
| INFO                      | Messaggi sul normale funzionamento della mappatura della sorgente dell'evento        |



| Livello di log          | Utilizzo                                                                                  |
|-------------------------|-------------------------------------------------------------------------------------------|
| WARN (dettaglio minimo) | Messaggi su potenziali avvisi ed errori che potrebbero portare a comportamenti imprevisti |

Quando si seleziona un livello di registro, Lambda event poller invia i log a tale livello e a un livello inferiore. Ad esempio, se imposti il livello di log del sistema di mappatura delle sorgenti degli eventi su INFO, event poller non invia gli output del log a livello DEBUG.

### Configurazione della registrazione

È possibile impostare la configurazione di registrazione durante la creazione o l'aggiornamento di una mappatura della sorgente di eventi Kafka.

#### Configurazione della registrazione (console)

Per configurare la registrazione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della tua funzione.
3. Esegui una delle seguenti operazioni:
  - Per aggiungere un nuovo trigger Kafka, in Panoramica delle funzioni, scegli **Aggiungi trigger**.
  - Per modificare un trigger Kafka esistente, scegli il trigger e quindi scegli **Modifica**.
4. In **Event poller configuration**, per la modalità **Provisioned**, abilita la casella di controllo **Configura**. E verrebbe visualizzata l'impostazione del livello di registro.
5. Scegli l'elenco a discesa **Livello di registro** e seleziona un livello per la mappatura della sorgente dell'evento.
6. Scegli **Aggiungi** o **Salva** in basso per creare o aggiornare la mappatura della sorgente dell'evento.

#### Configurazione della registrazione (AWS CLI)

##### Creazione di una mappatura della sorgente di eventi con registrazione

L'esempio seguente crea una mappatura della sorgente di eventi Amazon MSK con configurazione di registrazione:

```
aws lambda create-event-source-mapping \
  --function-name my-kafka-function \
  --topics AWSKafkaTopic \
  --event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/my-cluster/abc123 \
  --starting-position LATEST \
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=3 \
  --logging-config '{"SystemLogLevel":"DEBUG"}'
```

Per Kafka autogestito, utilizza la stessa sintassi:

```
aws lambda create-event-source-mapping \
  --function-name my-kafka-function \
  --topics AWSKafkaTopic \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":
["abc.xyz.com:9092"]}}' \
  --starting-position LATEST \
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=3 \
  --logging-config '{"SystemLogLevel":"DEBUG"}'
```

## Aggiornamento della configurazione di registrazione

Usa il `update-event-source-mapping` comando per aggiungere o modificare la configurazione di registrazione:

```
aws lambda update-event-source-mapping \
  --uuid 12345678-1234-1234-1234-123456789012 \
  --logging-config '{"SystemLogLevel":"WARN"}'
```

Formato di registrazione per un registro del sistema di mappatura delle sorgenti di eventi di Kafka

Quando il poller di eventi Lambda invia il registro, ogni voce di registro contiene metadati generali di mappatura della sorgente degli eventi e anche contenuti specifici dell'evento.

### WARN (record del registro)

Il record WARN contiene errori o avvisi provenienti dal poller degli eventi e viene emesso quando si è verificato l'evento. Ad esempio:

```
{
  "eventType": "ESM_PROCESSING_EVENT",
  "timestamp": 1546347650000,
```

```

    "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:12345678-1234-1234-1234-123456789012",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/tests-
cluster/87654321-4321-4321-4321-876543221-s1",
    "eventProcessorId": "12345678-1234-1234-1234-123456789012/0",
    "logLevel": "WARN",
    "error": {
        "errorMessage": "Timeout expired while fetching topic metadata",
        "errorCode": "org.apache.kafka.common.errors.TimeoutException"
    }
}

```

## Record del registro INFO

Il record INFO contiene le configurazioni del client consumer Kafka in ogni poller di eventi e viene emesso nel caso in cui un consumatore venga creato o modificato. Ad esempio:

```

{
    "eventType": "POLLER_STATUS_EVENT",
    "timestamp": 1546347660000,
    "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:12345678-1234-1234-1234-123456789012",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/tests-
cluster/87654321-4321-4321-4321-876543221-s1",
    "eventProcessorId": "12345678-1234-1234-1234-123456789012/0",
    "logLevel": "INFO",
    "kafkaEventSourceConnection": {
        "brokerEndpoints": "boot-abcd1234.c2.kafka-serverless.us-
east-1.amazonaws.com:9098",
        "consumerId": "12345678-1234-1234-1234-123456789012-0",
        "topics": [
            "test"
        ],
        "consumerGroupId": "12345678-1234-1234-1234-123456789012",
        "securityProtocol": "SASL_SSL",
        "saslMechanism": "AWS_MSK_IAM",
        "totalPartitionCount": 2,
        "assignedPartitionCount": 2,
        "partitionsAssignmentGeneration": 5,
        "assignedPartitions": [
            "test-0",
            "test-1"
        ],
    },
}

```

```
    "networkConfig": {
      "ipAddresses": [
        "10.100.141.1"
      ],
      "subnetCidrBlock": "10.100.128.0/20",
      "securityGroups": [
        "sg-abcdefabcdefabcdef"
      ]
    }
  }
}
```

## Record di registro DEBUG

Il registro DEBUG contiene le informazioni relative agli offset di Kafka nell'elaborazione della mappatura della sorgente degli eventi e le informazioni sugli offset vengono emesse al minuto. Ad esempio:

```
{
  "eventType": "KAFKA_STATUS_EVENT",
  "timestamp": 1546347670000,
  "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:12345678-1234-1234-1234-123456789012",
  "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/tests-
cluster/87654321-4321-4321-4321-876543221-s1",
  "eventProcessorId": "12345678-1234-1234-1234-123456789012/0",
  "logLevel": "DEBUG",
  "kafkaPartitionOffsets": {
    "partition": "test-1",
    "endOffset": 5004,
    "consumedOffset": 5003,
    "processedOffset": 5003,
    "committedOffset": 5004
  }
}
```

## Risoluzione degli errori di mappatura delle sorgenti degli eventi Kafka

Negli argomenti seguenti vengono forniti suggerimenti per la risoluzione dei problemi relativi a errori e problemi che potrebbero verificarsi durante l'utilizzo di Amazon MSK o Apache Kafka autogestito con Lambda.

Per ulteriori informazioni sulla risoluzione dei problemi, visita il [Knowledge Center AWS](#).

### Errori di autenticazione e autorizzazione

Se manca una delle autorizzazioni necessarie per consumare i dati dal cluster Kafka, Lambda visualizza uno dei seguenti messaggi di errore nella mappatura delle sorgenti degli eventi sotto. LastProcessingResult

#### Messaggi di errore

- [Il cluster non è riuscito ad autorizzare Lambda](#)
- [Autenticazione SASL non riuscita](#)
- [Il server non è riuscito ad autenticare Lambda](#)
- [Lambda non è riuscita ad autenticare il server](#)
- [Il certificato o la chiave privata forniti non sono validi](#)

### Il cluster non è riuscito ad autorizzare Lambda

Per i SASL/SCRAM nostri MTL, questo errore indica che l'utente fornito non dispone di tutte le seguenti autorizzazioni necessarie per la lista di controllo degli accessi Kafka (ACL):

- DescribeConfigs Cluster
- Descrivi il gruppo
- Leggi il gruppo
- Descrivi l'argomento
- Leggi l'argomento

Quando crei Kafka ACLs con le `kafka-cluster` autorizzazioni richieste, specifica l'argomento e il gruppo come risorse. Il nome dell'argomento deve corrispondere all'argomento nella mappatura dell'origine eventi. Il nome del gruppo deve corrispondere all'UUID della mappatura dell'origine eventi.

Dopo avere aggiunto le autorizzazioni richieste al ruolo di esecuzione, potrebbero essere necessari alcuni minuti affinché le modifiche entrino in vigore.

Di seguito è riportato un esempio di registro ESM a livello di sistema dopo aver abilitato Logging [Config](#) per questo problema:

```
{
  "eventType": "ESM_PROCESSING_EVENT",
  "timestamp": 1734567890123,
  "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-kafka-cluster/12345678-abcd-1234-efgh-EXAMPLE11111-1",
  "eventProcessorId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111/0",
  "logLevel": "WARN",
  "error": {
    "errorMessage": "Not authorized to access topics: [my-topic]",
    "errorCode": "org.apache.kafka.common.errors.TopicAuthorizationException"
  }
}
```

## Autenticazione SASL non riuscita

Per SASL/SCRAM o SASL/PLAIN, questo errore indica che le credenziali di accesso fornite non sono valide.

Per il controllo degli accessi IAM, al ruolo di esecuzione manca l'`kafka-cluster:Connect` autorizzazione per il cluster. Aggiungi questa autorizzazione al ruolo e specifica l'Amazon Resource Name (ARN) del cluster come risorsa.

Potresti visualizzare questo errore in modo intermittente. Il cluster rifiuta le connessioni dopo che il numero di connessioni TCP supera la quota di servizio. Lambda cessa e ritenta finché una connessione non ha esito positivo. Dopo che Lambda si connette al cluster e ha eseguito il polling dei registri, l'ultimo risultato di elaborazione cambia in OK.

Di seguito è riportato un esempio di log ESM a livello di sistema dopo aver abilitato Logging [Config](#) per questo problema quando si utilizza l'autenticazione IAM:

```
{
  "eventType": "ESM_PROCESSING_EVENT",
  "timestamp": 1734567890456,
```

```
{
  "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-kafka-
cluster/12345678-abcd-1234-efgh-EXAMPLE22222-1",
  "eventProcessorId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222/0",
  "logLevel": "WARN",
  "error": {
    "errorMessage": "[a1b2c3d4-5678-90ab-cdef-EXAMPLE22222]: Access denied",
    "errorCode": "org.apache.kafka.common.errors.SaslAuthenticationException"
  }
}
```

## Il server non è riuscito ad autenticare Lambda

Questo errore indica che il broker Kafka non è riuscito ad autenticare Lambda. Questo errore può verificarsi per uno dei seguenti motivi:

- Non è stato fornito un certificato client per l'autenticazione mTLS.
- È stato fornito un certificato client, ma i broker Kafka non sono configurati per l'utilizzo dell'autenticazione mTLS.
- Un certificato client non è attendibile per i broker Kafka.

## Lambda non è riuscita ad autenticare il server

Questo errore indica che Lambda non è riuscita ad autenticare il broker Kafka. Questo errore può verificarsi per uno dei seguenti motivi:

- Per Apache Kafka autogestito: i broker Kafka utilizzano certificati autofirmati o una CA privata, ma non hanno fornito il certificato CA root del server.
- Per Apache Kafka autogestito: il certificato CA principale del server non corrisponde alla CA principale che ha firmato il certificato del broker.
- La convalida del nome host non è riuscita perché il certificato del broker non contiene il nome DNS o l'indirizzo IP del broker come nome alternativo dell'oggetto.

## Il certificato o la chiave privata forniti non sono validi

Questo errore indica che il consumatore Kafka non ha potuto utilizzare il certificato o la chiave privata fornita. Assicurati che il certificato e la chiave utilizzino il formato PEM e che la crittografia a chiave privata utilizzi un algoritmo. PBES1

Di seguito è riportato un esempio di registro ESM a livello di sistema dopo aver abilitato Logging [Config](#) per questo problema:

```
{
  "eventType": "ESM_PROCESSING_EVENT",
  "timestamp": 1734567891234,
  "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLE44444",
  "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-kafka-
cluster/12345678-abcd-1234-efgh-EXAMPLE44444-1",
  "eventProcessorId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE44444/0",
  "logLevel": "WARN",
  "error": {
    "errorMessage": "Invalid PEM keystore configs",
    "errorCode": "org.apache.kafka.common.errors.InvalidConfigurationException"
  }
}
```

## Errori di rete e connettività

I problemi di configurazione della rete possono impedire a Lambda di connettersi al cluster Kafka. I seguenti argomenti descrivono gli errori più comuni relativi alla rete.

### Messaggi di errore

- [Timeout di connessione dovuto alla configurazione del gruppo di sicurezza](#)
- [Gli endpoint del broker Kafka non possono essere risolti](#)

### Timeout di connessione dovuto alla configurazione del gruppo di sicurezza

Se il gruppo di sicurezza associato al tuo cluster Kafka non consente il traffico in entrata da solo, Lambda non può connettersi al cluster. Assicurati che le regole in entrata del gruppo di sicurezza consentano il traffico proveniente dal gruppo di sicurezza stesso sulle porte del broker Kafka.

Di seguito è riportato un esempio di registro ESM a livello di sistema dopo aver abilitato Logging [Config](#) per questo problema:

```
{
  "eventType": "ESM_PROCESSING_EVENT",
  "timestamp": 1734567892345,
  "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLE55555",
```



```

    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-kafka-
cluster/12345678-abcd-1234-efgh-EXAMPLE55555-1",
    "eventProcessorId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE55555/0",
    "logLevel": "WARN",
    "error": {
      "errorMessage": "Timeout expired while fetching topic metadata",
      "errorCode": "org.apache.kafka.common.errors.TimeoutException"
    }
  }
}

```

È inoltre possibile controllare il registro Kafka Consumer INFO per verificare la connessione e la configurazione di rete. Il `brokerEndpoints` campo mostra gli indirizzi del broker Kafka `securityProtocol` e `saslMechanism` (se applicabile) mostra il metodo di autenticazione e il `networkConfig` campo mostra gli indirizzi IP, il blocco CIDR di sottorete e i gruppi di sicurezza utilizzati dalla mappatura delle sorgenti degli eventi. Verifica che i gruppi di sicurezza elencati consentano il traffico in entrata richiesto:

```

{
  "eventType": "POLLER_STATUS_EVENT",
  "timestamp": 1734567892456,
  "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
  "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-kafka-cluster/
a1b2c3d4-5678-90ab-cdef-11111EXAMPLE-1",
  "eventProcessorId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/0",
  "logLevel": "INFO",
  "kafkaEventSourceConnection": {
    "brokerEndpoints": "boot-abcd1234.c2.kafka-serverless.us-
east-1.amazonaws.com:9098",
    "consumerId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE-0",
    "topics": [
      "my-topic"
    ],
    "consumerGroupId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
    "securityProtocol": "SASL_SSL",
    "saslMechanism": "AWS_MSK_IAM",
    "totalPartitionCount": 2,
    "assignedPartitionCount": 2,
    "partitionsAssignmentGeneration": 1,
    "assignedPartitions": [
      "my-topic-0",
      "my-topic-1"
    ]
  }
}

```

```

    ],
    "networkConfig": {
      "ipAddresses": [
        "10.0.0.37"
      ],
      "subnetCidrBlock": "10.0.0.32/28",
      "securityGroups": [
        "sg-0123456789abcdef0"
      ]
    }
  }
}

```

Gli endpoint del broker Kafka non possono essere risolti

Questo errore indica che il cluster Kafka non esiste o è stato eliminato. Verifica che il cluster specificato nella mappatura dell'origine degli eventi esista e sia in uno stato attivo.

Di seguito è riportato un esempio di registro ESM a livello di sistema dopo aver abilitato [Logging Config](#) per questo problema:

```

{
  "eventType": "ESM_PROCESSING_EVENT",
  "timestamp": 1734567893456,
  "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLE66666",
  "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-kafka-cluster/12345678-abcd-1234-efgh-EXAMPLE66666-1",
  "eventProcessorId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE66666/0",
  "logLevel": "WARN",
  "error": {
    "errorMessage": "No resolvable bootstrap urls given in bootstrap.servers",
    "errorCode": "org.apache.kafka.common.config.ConfigException"
  }
}

```

## Errori della mappatura dell'origine eventi

Quando aggiungi il cluster Apache Kafka come [origine eventi](#) per la funzione Lambda, se la funzione rileva un errore, il consumer Kafka arresta l'elaborazione dei record. I consumatori di uno shard dell'argomento sono quelli che sottoscrivono, leggono ed elaborano i record. Gli altri consumatori Kafka possono continuare a elaborare i record, a condizione che non riscontrino lo stesso errore.

Per determinare la causa di un consumatore interrotto, controlla il campo `StateTransitionReason` nella risposta di `EventSourceMapping`. Nell'elenco seguente vengono descritti gli errori dell'origine eventi che è possibile ricevere:

**ESM\_CONFIG\_NOT\_VALID**

La configurazione della mappatura della fonte evento non è valida.

**EVENT\_SOURCE\_AUTHN\_ERROR**

Lambda non ha potuto autenticare la fonte evento.

**EVENT\_SOURCE\_AUTHZ\_ERROR**

Lambda non dispone delle autorizzazioni necessarie per accedere alla fonte evento.

**FUNCTION\_CONFIG\_NOT\_VALID**

La configurazione della funzione non è valida.

 **Note**

Se i record degli eventi Lambda superano il limite di dimensione consentito di 6 MB, potrebbero non venire elaborati.

# Richiamo di funzione Lambda utilizzando un endpoint Gateway Amazon API

È possibile creare un'API web con un endpoint HTTP per la funzione Lambda utilizzando Amazon API Gateway. API Gateway fornisce strumenti per la creazione e la documentazione di API Web che indirizzano le richieste HTTP alle funzioni Lambda. È possibile proteggere l'accesso all'API con controlli di autenticazione e autorizzazione. Le API possono servire il traffico via Internet o possono essere accessibili solo all'interno del VPC.

## Tip

Lambda offre due modi per invocare una funzione tramite un endpoint HTTP: API Gateway e URL della funzione Lambda. In caso di dubbi su quale sia il metodo migliore per il tuo caso d'uso, consulta [the section called “API Gateway e URL di funzione”](#).

Le risorse nell'API definiscono uno o più metodi, ad esempio GET o POST. I metodi hanno un'integrazione che instrada le richieste a una funzione Lambda o a un altro tipo di integrazione. È possibile definire ogni risorsa e metodo singolarmente oppure utilizzare tipi di risorse e metodi speciali per soddisfare tutte le richieste che si adattano a un modello. Una [risorsa proxy](#) cattura tutti i percorsi sottostanti una risorsa. Il metodo ANY cattura tutti i metodi HTTP.

## Sections

- [Scelta di un tipo di API](#)
- [Aggiunta di un endpoint alla funzione Lambda](#)
- [Integrazione proxy](#)
- [Formato dell'evento](#)
- [Formato della risposta](#)
- [Permissions](#)
- [Applicazione di esempio](#)
- [Il gestore di eventi di Powertools per AWS Lambda](#)
- [Tutorial: uso di Lambda con Amazon API Gateway](#)
- [Gestione degli errori con un'API di API Gateway](#)
- [Selezionare un metodo per richiamare la funzione Lambda tramite una richiesta HTTP](#)

## Scelta di un tipo di API

API Gateway supporta tre tipi di API che richiamano le funzioni Lambda:

- [API HTTP](#): un'API RESTful leggera e a bassa latenza.
- [REST API](#): un'API RESTful personalizzabile e ricca di funzionalità.
- [WebSocket API](#): un'API web che mantiene connessioni permanenti con i client per comunicazioni full-duplex.

Le API HTTP e le REST API sono entrambe API RESTful che elaborano richieste HTTP e restituiscono risposte. Le API HTTP sono più recenti e vengono create con la versione 2 di API Gateway. Le seguenti funzionalità sono nuove per le API HTTP:

### Funzionalità API HTTP

- Distribuzioni automatiche: quando si modificano routing o integrazioni, le modifiche vengono distribuite automaticamente alle fasi in cui è abilitata la distribuzione automatica.
- Fase predefinita: è possibile creare una fase predefinita (`$default`) che invii le richieste nel percorso root dell'URL dell'API. Per le fasi denominate, è necessario includere il nome dello stage all'inizio del percorso.
- Configurazione CORS: è possibile configurare l'API in modo da aggiungere intestazioni CORS alle risposte in uscita, invece di aggiungerle manualmente nel codice della funzione.

Le REST API sono le API RESTful classiche supportate da API Gateway dall'avvio. Le REST API dispongono attualmente di più funzionalità di personalizzazione, integrazione e gestione.

### Funzionalità REST API

- Tipi di integrazione: le REST API supportano le integrazioni Lambda personalizzate. Con un'integrazione personalizzata, è possibile inviare solo il corpo della richiesta alla funzione, o applicare un modello di trasformazione al corpo della richiesta prima di inviarlo alla funzione.
- Controllo degli accessi: le REST API supportano più opzioni per l'autenticazione e l'autorizzazione.
- Monitoraggio e tracciamento: le API REST supportano il AWS X-Ray tracciamento e opzioni di registrazione aggiuntive.

Per un confronto dettagliato, consulta [Scelta tra le API HTTP e le REST API](#) nella Guida per gli sviluppatori di API Gateway.

**WebSocket** Le API utilizzano anche l'API API Gateway versione 2 e supportano un set di funzionalità simile. Utilizza un' WebSocket API per le applicazioni che traggono vantaggio da una connessione persistente tra il client e l'API. WebSocket Le API forniscono una comunicazione full-duplex, il che significa che sia il client che l'API possono inviare messaggi in modo continuo senza attendere una risposta.

Le API HTTP supportano un formato di evento semplificato (versione 2.0). Per un esempio di evento proveniente da un'API HTTP, consulta [Creare integrazioni AWS Lambda proxy per le API HTTP in API Gateway](#).

Per ulteriori informazioni, consulta [Creare integrazioni AWS Lambda proxy per le API HTTP in API Gateway](#).

## Aggiunta di un endpoint alla funzione Lambda

Per aggiungere un endpoint pubblico alla funzione Lambda

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. In Function overview (Panoramica delle funzioni), scegliere Add trigger (Aggiungi trigger).
4. Selezionare API Gateway.
5. Scegliere Create an API (Crea una nuova API) o Use an existing API (Usa un'API esistente).
  - a. Nuova API: per API type (Tipo di API), scegliere HTTP API (API HTTP). Per ulteriori informazioni, consulta [Scelta di un tipo di API](#).
  - b. API esistente: seleziona l'API dal menu a discesa o inserisci l'ID API (ad esempio, r3pmxmplak).
6. Per Security (Sicurezza), scegliere Open (Apri).
7. Scegliere Add (Aggiungi).

## Integrazione proxy

Le API di API Gateway sono costituite da fasi, risorse, metodi e integrazioni. La fase e la risorsa determinano il percorso dell'endpoint:

## Formato del percorso API

- `/prod/`: la fase e la risorsa radice di `prod`.
- `/prod/user`: la fase di `prod` e la risorsa di `user`.
- `/dev/{proxy+}` – Qualunque routing della fase `dev`.
- `/`: (API HTTP) la fase e la risorsa radice predefinite.

Un'integrazione Lambda mappa una combinazione di percorso e metodo HTTP a una funzione Lambda. È possibile configurare API Gateway in modo da passare il corpo della richiesta HTTP così com'è (integrazione personalizzata), o da incapsulare il corpo della richiesta in un documento che include tutte le informazioni sulla richiesta, incluse intestazioni, risorsa, percorso e metodo.

Per ulteriori informazioni, consulta [Integrazioni del proxy Lambda di API Gateway](#).

## Formato dell'evento

Amazon API Gateway richiama la funzione [in modo sincrono](#) con un evento che contiene una rappresentazione JSON della richiesta HTTP. Per un'integrazione personalizzata, l'evento è il corpo della richiesta. Per un'integrazione proxy, l'evento ha una struttura definita. Per un esempio di evento proxy proveniente da una REST API di API Gateway, consulta [Formato di input di una funzione Lambda per l'integrazione del proxy](#) nella Guida per gli sviluppatori di API Gateway.

## Formato della risposta

API Gateway attende una risposta dalla funzione e inoltra il risultato al chiamante. Per un'integrazione personalizzata, è possibile definire una risposta di integrazione e una risposta al metodo per convertire l'output dalla funzione a una risposta HTTP. Per un'integrazione proxy, la funzione deve rispondere con una rappresentazione della risposta in un formato specifico.

L'esempio seguente mostra un oggetto di risposta da una Node.js funzione. L'oggetto risposta rappresenta una risposta HTTP riuscita che contiene un documento JSON.

Example `index.mjs` — Oggetto di risposta per l'integrazione del proxy () Node.js

```
var response = {
  "statusCode": 200,
  "headers": {
    "Content-Type": "application/json"
  },
  "isBase64Encoded": false,
```

```

    "multiValueHeaders": {
      "X-Custom-Header": ["My value", "My other value"],
    },
    "body": "{\n  \"TotalCodeSize\": 104330022,\n  \"FunctionCount\": 26\n}"
  }
}

```

Il runtime Lambda serializza l'oggetto di risposta in JSON e lo invia all'API. L'API analizza la risposta e la utilizza per creare una risposta HTTP, che quindi la invia al client che ha effettuato la richiesta originale.

### Example Risposta HTTP

```

< HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 55
< Connection: keep-alive
< x-amzn-RequestId: 32998fea-xmpl-4268-8c72-16138d629356
< X-Custom-Header: My value
< X-Custom-Header: My other value
< X-Amzn-Trace-Id: Root=1-5e6aa925-ccecxmplbae116148e52f036
<
{
  "TotalCodeSize": 104330022,
  "FunctionCount": 26
}

```

## Permissions

Amazon API Gateway ottiene l'autorizzazione a richiamare la funzione dalla [policy basata su risorse](#) della funzione. È possibile concedere l'autorizzazione a un'intera API o concedere un accesso limitato a una fase, una risorsa o un metodo.

Quando aggiungi un'API alla funzione utilizzando la console Lambda, la console API Gateway o in un modello AWS SAM, la policy basata su risorse della funzione viene aggiornata automaticamente. Di seguito è riportata una policy di funzioni di esempio.

### Example Policy di funzione

#### JSON

```
{
```



```

"Version": "2012-10-17",
"Id": "default",
"Statement": [
  {
    "Sid": "nodejs-apig-functiongetEndpointPermissionProd-BWDBXMPLXE2F",
    "Effect": "Allow",
    "Principal": {
      "Service": "apigateway.amazonaws.com"
    },
    "Action": "lambda:InvokeFunction",
    "Resource": "arn:aws:lambda:us-east-2:111122223333:function:nodejs-apig-function-1G3MXMPLXVXYI",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "111122223333"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:execute-api:us-east-2:111122223333:ktyvxmpls1/*/GET/"
      }
    }
  }
]
}

```

È possibile gestire manualmente le autorizzazioni delle policy di funzione con le seguenti operazioni API:

- [AddPermission](#)
- [RemovePermission](#)
- [GetPolicy](#)

Per concedere l'autorizzazione di chiamata a un'API esistente, utilizzare il comando `add-permission`. Esempio:

```

aws lambda add-permission \
  --function-name my-function \
  --statement-id apigateway-get --action lambda:InvokeFunction \
  --principal apigateway.amazonaws.com \
  --source-arn "arn:aws:execute-api:us-east-2:123456789012:mnh1xmpli7/default/GET/"

```

Verrà visualizzato l'output seguente:

```
{
  "Statement": "{\"Sid\":\"apigateway-test-2\",\"Effect\":\"Allow\",\"Principal\":{\"Service\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":\"arn:aws:lambda:us-east-2:123456789012:function:my-function\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-east-2:123456789012:mnh1xmpli7/default/GET\"}}}"
}
```

#### Note

Se la funzione e l'API sono diverse Regioni AWS, l'identificatore della regione nell'ARN di origine deve corrispondere alla regione della funzione, non alla regione dell'API. Quando API Gateway richiama una funzione, utilizza un ARN di risorsa basato sull'ARN dell'API, ma modificato per corrispondere alla regione della funzione.

L'ARN di origine in questo esempio concede l'autorizzazione a un'integrazione nel metodo GET della risorsa root nella fase predefinita di un'API, con ID `mnh1xmpli7`. È possibile utilizzare un asterisco nell'ARN di origine per concedere autorizzazioni a più fasi, metodi o risorse.

#### Modelli di risorse

- `mnh1xmpli7/*/GET/*`: metodo GET su tutte le risorse in tutte le fasi.
- `mnh1xmpli7/prod/ANY/user`: metodo ANY sulla risorsa `user` nella fase `prod`.
- `mnh1xmpli7/*/*/*`: qualsiasi metodo su tutte le risorse in tutte le fasi.

Per informazioni dettagliate sulla visualizzazione delle policy e sulla rimozione delle istruzioni, vedere [Utilizzo di policy basate sulle risorse in Lambda](#).

#### Applicazione di esempio

L'[API Gateway con app di Node.js](#) esempio include una funzione con un AWS SAM modello che crea un'API REST con la AWS X-Ray traccia abilitata. Include anche script per l'implementazione, il richiamo della funzione, il test dell'API e la pulizia.

## Il gestore di eventi di Powertools per AWS Lambda

Il gestore di eventi del toolkit Powertools for AWS Lambda fornisce routing, middleware, configurazione CORS, generazione di specifiche OpenAPI, convalida delle richieste, gestione degli errori e altre utili funzionalità per la scrittura di funzioni Lambda richiamate da un endpoint API Gateway (HTTP o REST). L'utilità Event Handler è disponibile per Python e TypeScript/JavaScript. Per ulteriori informazioni, vedere [Event Handler REST API](#) nella documentazione di Powertools for AWS Lambda (Python) e [Event Handler HTTP API](#) nella documentazione di Powertools for Lambda (. AWS TypeScript

### Python

```
from aws_lambda_powertools import Logger
from aws_lambda_powertools.event_handler import APIGatewayRestResolver
from aws_lambda_powertools.logging import correlation_paths
from aws_lambda_powertools.utilities.typing.lambda_context import LambdaContext

app = APIGatewayRestResolver()
logger = Logger()

@app.get("/healthz")
def ping():
    return {"message": "health status ok"}

@logger.inject_lambda_context(correlation_id_path=correlation_paths.API_GATEWAY_REST)
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    return app.resolve(event, context)
```

### Typescript

```
import { Router } from '@aws-lambda-powertools/event-handler/experimental-rest';
import { Logger } from '@aws-lambda-powertools/logger';
import {
    correlationPaths,
    search,
} from '@aws-lambda-powertools/logger/correlationId';
import type { Context } from 'aws-lambda/handler';

const logger = new Logger({
    correlationIdSearchFn: search,
});
```

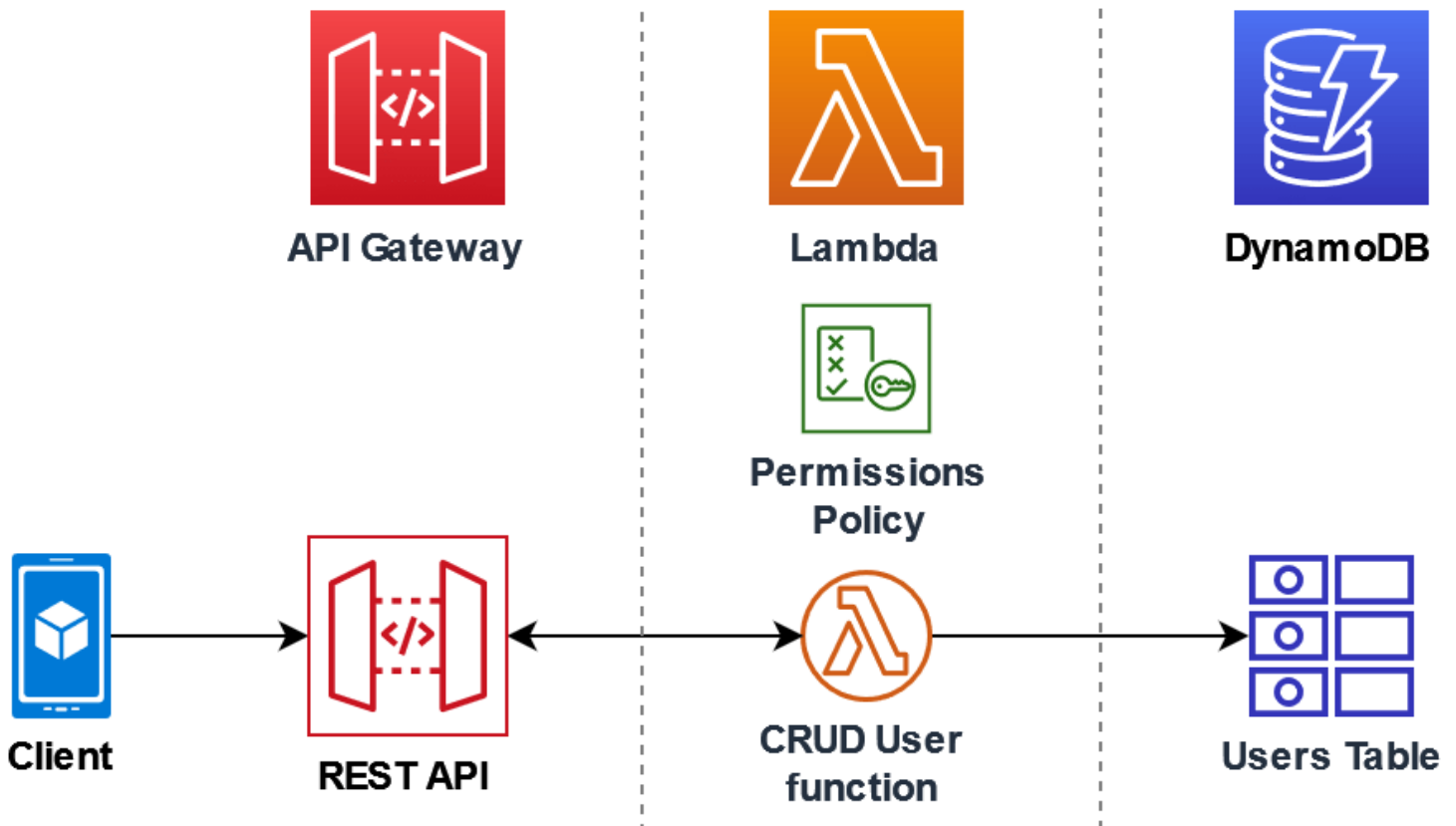
```
const app = new Router({ logger });

app.get("/healthz", async () => {
  return { message: "health status ok" };
});

export const handler = async (event: unknown, context: Context) => {
  // You can continue using other utilities just as before
  logger.addContext(context);
  logger.setCorrelationId(event, correlationPaths.API_GATEWAY_REST);
  return app.resolve(event, context);
};
```

## Tutorial: uso di Lambda con Amazon API Gateway

In questo tutorial, viene creata una REST API tramite la quale viene richiamata una funzione Lambda utilizzando una richiesta HTTP. La funzione Lambda esegue operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD) su una tabella DynamoDB. Questa funzione viene fornita qui a scopo dimostrativo, ma imparerai a configurare un'API REST API Gateway in grado di richiamare qualsiasi funzione Lambda.



L'utilizzo di Gateway API fornisce agli utenti un endpoint HTTP sicuro per richiamare la funzione Lambda e può aiutare a gestire grandi volumi di chiamate alla funzione limitando il traffico e convalidando e autorizzando automaticamente le chiamate API. API Gateway fornisce anche controlli di sicurezza flessibili utilizzando AWS Identity and Access Management (IAM) e Amazon Cognito. Ciò è utile nei casi d'uso in cui è richiesta un'autorizzazione preventiva per le chiamate all'applicazione.

 Tip

Lambda offre due modi per invocare una funzione tramite un endpoint HTTP: API Gateway e URL della funzione Lambda. In caso di dubbi su quale sia il metodo migliore per il tuo caso d'uso, consulta [the section called “API Gateway e URL di funzione”](#).

Per completare questo tutorial, segui le seguenti fasi:

1. Crea e configura una funzione Lambda in Python o Node.js per eseguire operazioni su una tabella DynamoDB.
2. Crea una REST API in Gateway API per connetterti alla funzione Lambda.
3. Crea una tabella DynamoDB e testala con la funzione Lambda nella console.
4. Implementa la tua API e testa la configurazione completa usando curl in un terminale.

Completando queste fasi, impari a utilizzare API Gateway per creare un endpoint HTTP in grado di richiamare in modo sicuro una funzione Lambda su qualsiasi scala. Imparerai anche come distribuire la tua API e come testarla nella console e inviando una richiesta HTTP tramite un terminale.

### Creazione di una policy di autorizzazione

Prima di poter creare un ruolo di [esecuzione](#) per la tua funzione Lambda, devi prima creare una policy di autorizzazioni per concedere alla tua funzione il permesso di accedere alle risorse richieste. AWS Per questo tutorial, la policy consente a Lambda di eseguire operazioni CRUD su una tabella DynamoDB e scrivere su Amazon Logs. CloudWatch

### Come creare la policy

1. Apri la pagina [Policies \(Policy\)](#) nella console IAM.
2. Scegli Crea policy.
3. Scegliere la scheda JSON e quindi incollare la seguente policy personalizzata nell'editor JSON.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1428341300017",
      "Action": [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "",
      "Resource": "*",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Effect": "Allow"
    }
  ]
}
```

4. Scegliere Next: Tags (Successivo: Tag).
5. Scegliere Next:Review (Successivo: Rivedi).
6. In Rivedi policy, per Nome della policy inserisci **lambda-apigateway-policy**.
7. Scegli Crea policy.

## Creazione di un ruolo di esecuzione

Un ruolo di [esecuzione](#) è un ruolo AWS Identity and Access Management (IAM) che concede a una funzione Lambda l'autorizzazione all'accesso e alle risorse. Servizi AWS Per consentire alla funzione

di eseguire operazioni su una tabella DynamoDB, collega la policy di autorizzazione che hai creato nella fase precedente.

Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione personalizzata

1. Aprire la [pagina Roles \(Ruoli\)](#) della console IAM.
2. Scegliere Crea ruolo.
3. Per il tipo di entità attendibile, scegli Servizio AWS , quindi per il caso d'uso seleziona Lambda.
4. Scegli Avanti.
5. Nella casella di ricerca delle policy, immettere **lambda-apigateway-policy**.
6. Nei risultati della ricerca, seleziona la policy creata (lambda-apigateway-policy), quindi scegli Next (Successivo).
7. In Role details (Dettagli del ruolo), per Role name (Nome del ruolo), specifica **lambda-apigateway-role**, quindi scegli Create role (Crea ruolo).

Creazione della funzione Lambda

1. Apri la [pagina Funzioni](#) della console Lambda e scegli Crea funzione.
2. Scegli Crea da zero.
3. Nel campo Function name (Nome funzione), immettere LambdaFunctionOverHttps.
4. Per Runtime, scegli il runtime più recente Node.js o Python.
5. In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito.
6. Scegli Utilizza un ruolo esistente, quindi seleziona il ruolo **lambda-apigateway-role** creato in precedenza.
7. Scegli Crea funzione.
8. Nel riquadro del codice sorgente, sostituisci il codice predefinito con il codice seguente Node.js o con il codice Python.

Node.js

L'regionimpostazione deve corrispondere al Regione AWS luogo in cui si distribuisce la funzione e si [crea la tabella DynamoDB](#).

Example index.mjs

```
import { DynamoDBDocumentClient, PutCommand, GetCommand,
```

```
UpdateCommand, DeleteCommand} from "@aws-sdk/lib-dynamodb";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const ddbClient = new DynamoDBClient({ region: "us-east-2" });
const ddbDocClient = DynamoDBDocumentClient.from(ddbClient);

// Define the name of the DDB table to perform the CRUD operations on
const tablename = "lambda-apigateway";

/**
 * Provide an event that contains the following keys:
 *
 * - operation: one of 'create,' 'read,' 'update,' 'delete,' or 'echo'
 * - payload: a JSON object containing the parameters for the table item
 *   to perform the operation on
 */
export const handler = async (event, context) => {

  const operation = event.operation;

  if (operation == 'echo'){
    return(event.payload);
  }

  else {
    event.payload.TableName = tablename;
    let response;

    switch (operation) {
      case 'create':
        response = await ddbDocClient.send(new
PutCommand(event.payload));
        break;
      case 'read':
        response = await ddbDocClient.send(new
GetCommand(event.payload));
        break;
      case 'update':
        response = ddbDocClient.send(new UpdateCommand(event.payload));
        break;
      case 'delete':
        response = ddbDocClient.send(new DeleteCommand(event.payload));
        break;
      default:
```



```
        response = 'Unknown operation: ${operation}';
    }
    console.log(response);
    return response;
}
};
```

## Python

### Example lambda\_function.py

```
import boto3

# Define the DynamoDB table that Lambda will connect to
table_name = "lambda-apigateway"

# Create the DynamoDB resource
dynamo = boto3.resource('dynamodb').Table(table_name)

# Define some functions to perform the CRUD operations
def create(payload):
    return dynamo.put_item(Item=payload['Item'])

def read(payload):
    return dynamo.get_item(Key=payload['Key'])

def update(payload):
    return dynamo.update_item(**{k: payload[k] for k in ['Key',
'UpdateExpression',
'ExpressionAttributeNames', 'ExpressionAttributeValues'] if k in payload})

def delete(payload):
    return dynamo.delete_item(Key=payload['Key'])

def echo(payload):
    return payload

operations = {
    'create': create,
    'read': read,
    'update': update,
    'delete': delete,
    'echo': echo,
```

```
}

def lambda_handler(event, context):
    '''Provide an event that contains the following keys:
        - operation: one of the operations in the operations dict below
        - payload: a JSON object containing parameters to pass to the
          operation being performed
    '''

    operation = event['operation']
    payload = event['payload']

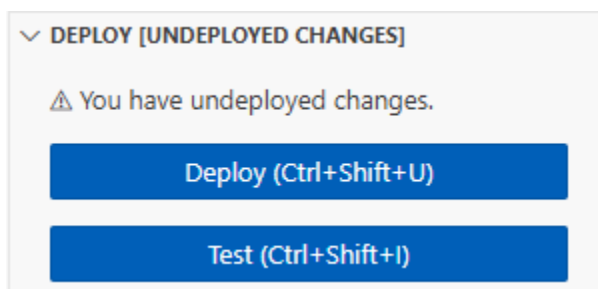
    if operation in operations:
        return operations[operation](payload)

    else:
        raise ValueError(f'Unrecognized operation "{operation}"')
```

#### Note

In questo esempio, il nome della tabella DynamoDB è definito come variabile nel codice della funzione. In un'applicazione reale, la best practice consiste nell'inviare questo parametro come variabile d'ambiente ed evitare di codificare il nome della tabella. Per ulteriori informazioni, consulta [Utilizzo AWS Lambda delle variabili di ambiente](#).

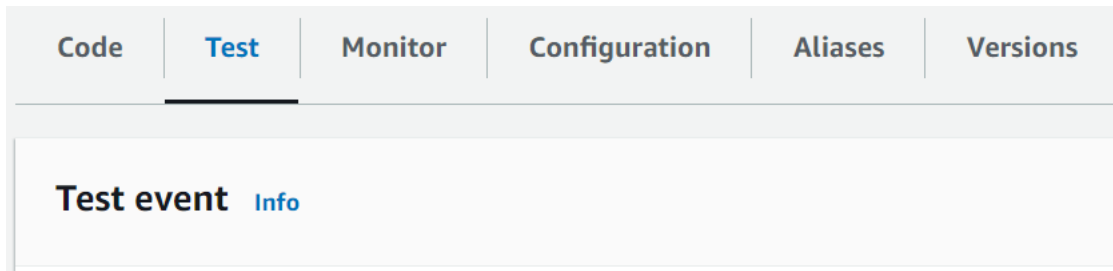
9. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



## Test della funzione

Prima di integrare la tua funzione con Gateway API, verifica di aver implementato correttamente la funzione. Utilizza la console Lambda per inviare un evento di test alla tua funzione.

1. Nella pagina della console Lambda della funzione, scegli la scheda Test.



2. Scorri alla sezione JSON dell'evento e sostituisci l'evento predefinito con il seguente. Questo evento corrisponde alla struttura prevista dalla funzione Lambda.

```
{
  "operation": "echo",
  "payload": {
    "somekey1": "somevalue1",
    "somekey2": "somevalue2"
  }
}
```

3. Scegli Test (Esegui test).
4. In Esecuzione funzione: riuscita, espandi Dettagli. Dovrebbe essere visualizzata la seguente risposta:

```
{
  "somekey1": "somevalue1",
  "somekey2": "somevalue2"
}
```

## Creazione di una REST API utilizzando API Gateway

In questo passaggio, crei l'API REST API Gateway che usi per richiamare la tua funzione Lambda.

Per creare l'API

1. Apri la [console API Gateway](#).
2. Seleziona Create API (Crea API).
3. Nella casella REST API, scegliere Build.
4. In Dettagli API, lasciare selezionata Nuova API e per Nome API, inserire **DynamoDBOperations**.
5. Seleziona Create API (Crea API).

## Creazione di una risorsa sulla REST API

Per aggiungere un metodo HTTP all'API, è necessario prima creare una risorsa su cui quel metodo possa operare. Qui viene creata la risorsa per gestire la tabella DynamoDB.

Per creare la risorsa

1. Nella [console API Gateway](#), nella pagina Risorse dell'API, scegliere Crea risorsa.
2. In Dettagli risorsa, per Nome risorsa, inserire **DynamoDBManager**.
3. Scegliere Create Resource (Crea risorsa).

## Creazione di un metodo HTTP POST

In questa fase, viene creato un metodo (POST) per la risorsa DynamoDBManager. Il metodo POST viene collegato alla funzione Lambda in modo che quando il metodo riceve una richiesta HTTP, Gateway API richiama la funzione Lambda.

### Note

Ai fini di questo tutorial, viene utilizzato un metodo HTTP (POST) per richiamare una singola funzione Lambda che esegue tutte le operazioni sulla tabella DynamoDB. In un'applicazione reale, la best practice consiste nell'utilizzare una funzione Lambda e un metodo HTTP diversi per ogni operazione. Per ulteriori informazioni, consulta [Il monolite Lambda](#) in Serverless Land.

## Creazione del metodo POST

1. Nella pagina Risorse dell'API, assicurarsi che la risorsa `/DynamoDBManager` sia evidenziata. Quindi, nel riquadro Metodi, scegliere Crea metodo.
2. Per Metodo HTTP scegliere POST.
3. Per Tipo di integrazione lasciare selezionato Funzione Lambda.
4. Per la funzione Lambda, scegliere nome della risorsa Amazon (ARN) per la funzione (`LambdaFunctionOverHttps`).
5. Scegli Crea metodo.

## Creazione di una tabella DynamoDB

Crea una tabella DynamoDB vuota su cui la tua funzione Lambda esegue le operazioni CRUD.

### Creare la tabella DynamoDB

1. Aprire la pagina [Tables \(Tabelle\)](#) della console DynamoDB.
2. Scegliere Create table (Crea tabella).
3. In Table details (Dettagli tabella), effettuare le seguenti operazioni:
  1. Nel campo Table name (Nome tabella) immetti **lambda-apigateway**.
  2. Per Partition key (Chiave di partizione), immettere **id** e mantenere il tipo di dati impostato come String (Stringa).
4. In Table settings (Impostazioni tabella), mantieni le impostazioni predefinite.
5. Scegliere Create table (Crea tabella).

### Test dell'integrazione di Gateway API, Lambda e DynamoDB

A questo punto sei pronto per testare l'integrazione del metodo API di Gateway API con la funzione Lambda e la tabella DynamoDB. Utilizzando la console di Gateway API, invii le richieste direttamente al metodo POST utilizzando la funzione di test della console. In questo passaggio, viene utilizzata prima un'operazione create per aggiungere un nuovo elemento alla tabella DynamoDB, quindi viene utilizzata un'operazione update per modificare l'elemento.

#### Test 1: creazione di un nuovo elemento nella tabella DynamoDB

1. Nella [console di Gateway API](#), scegli l'API (DynamoDBOperations).
2. Scegli il metodo POST sotto la risorsa DynamoDBManager.
3. Seleziona la scheda Test. Potrebbe essere necessario scegliere il pulsante freccia destra per visualizzare la scheda.
4. In Metodo di prova, lasciare vuote le stringhe di query e le intestazioni. Per Corpo della richiesta, incollare il file JSON seguente:

```
{
  "operation": "create",
  "payload": {
    "Item": {
```

```
{
  "id": "1234ABCD",
  "number": 5
}
```

5. Scegli Test (Esegui test).

I risultati visualizzati al termine del test devono mostrare lo stato 200. Questo codice di stato indica che l'operazione create è riuscita.

Come verifica, controlla che la tabella DynamoDB contenga il nuovo elemento.

6. Apri la pagina [Tables](#) (Tabelle) della console DynamoDB e scegli la tabella `lambda-apigateway`.
7. Scegli Explore table items (Esplora elementi della tabella). Nel riquadro Items returned (Elementi restituiti), dovresti vedere un elemento con l'id 1234ABCD e il numero 5. Esempio:

### Items returned (1)

| <input type="checkbox"/> | id (String)              | <input type="checkbox"/> | number |
|--------------------------|--------------------------|--------------------------|--------|
| <input type="checkbox"/> | <a href="#">1234ABCD</a> |                          | 5      |

### Test 2: aggiornamento dell'elemento nella tabella DynamoDB

- Nella [Console API Gateway](#), tornare alla scheda Test del metodo POST.
- In Metodo di prova, lasciare vuote le stringhe di query e le intestazioni. Per Corpo della richiesta, incollare il file JSON seguente:

```
{
  "operation": "update",
  "payload": {
    "Key": {
      "id": "1234ABCD"
    },
    "UpdateExpression": "SET #num = :newNum",
    "ExpressionAttributeNames": {
      "#num": "number"
    }
  }
}
```

```

    },
    "ExpressionAttributeValues": {
      ":newNum": 10
    }
  }
}

```

### 3. Scegli Test (Esegui test).

I risultati visualizzati al termine del test devono mostrare lo stato 200. Questo codice di stato indica che l'operazione update è riuscita.

Per confermare, controlla che l'elemento nella tua tabella DynamoDB sia stato modificato.

4. Apri la pagina [Tables](#) (Tabelle) della console DynamoDB e scegli la tabella `lambda-apigateway`.
5. Scegli Explore table items (Esplora elementi della tabella). Nel riquadro Items returned (Elementi restituiti), dovresti vedere un elemento con l'id 1234ABCD e il numero 10.

### Items returned (1)

| <input type="checkbox"/>   id (String)              | ▼   number |
|-----------------------------------------------------|------------|
| <input type="checkbox"/>   <a href="#">1234ABCD</a> | 10         |

## Distribuzione dell'API

Per consentire al client di chiamare l'API, è necessario creare una implementazione e una fase associata. Una fase rappresenta un'istanza dell'API, inclusi i suoi metodi e le sue integrazioni.

### Per distribuire l'API

1. Apri la pagina APIs (API) della [console Gateway API](#) quindi scegli l'API `DynamoDBOperations`.
2. Nella pagina Risorse per la tua API, scegliere Distribuzione dell'API.
3. Per Fase, scegliere \*Nuova fase\* quindi per Nome fase specificare **test**.
4. Seleziona Implementa.
5. Nel riquadro Editor fase (Editor fasi) del test, copia l'URL di richiamo. Lo usi nel passaggio successivo per richiamare la tua funzione utilizzando una richiesta HTTP.

## Uso di curl per richiamare la funzione tramite le richieste HTTP

A questo punto è possibile richiamare la funzione Lambda inviando una richiesta HTTP all'API. In questo passaggio, crei un nuovo elemento nella tabella DynamoDB e quindi esegui le operazioni di lettura, aggiornamento ed eliminazione su quell'elemento.

Per creare un elemento nella tabella DynamoDB utilizzando curl

1. Apri un terminale o un prompt dei comandi sul computer locale ed esegui il comando `curl` seguente utilizzando l'URL di invocazione che hai copiato nel passaggio precedente. Questo comando utilizza le seguenti opzioni:
  - `-H`: aggiunge un'intestazione personalizzata alla richiesta. Qui specifica il tipo di contenuto come JSON.
  - `-d`: invia i dati nel corpo della richiesta. Per impostazione predefinita, questa opzione utilizza un metodo HTTP POST.

Linux/macOS

```
curl https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager \
-H "Content-Type: application/json" \
-d '{"operation": "create", "payload": {"Item": {"id": "5678EFGH", "number": 15}}}'
```

PowerShell

```
curl.exe 'https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager' -H 'Content-Type: application/json' -d '{"operation": "create", "payload": {"Item": {"id": "5678EFGH", "number": 15}}}'
```

Se l'operazione ha avuto esito positivo, dovrebbe essere restituita una risposta con un codice di stato HTTP di 200.

2. È inoltre possibile utilizzare la console DynamoDB per verificare che il nuovo elemento sia nella tabella effettuando le operazioni seguenti:
  1. Apri la pagina [Tables](#) (Tabelle) della console DynamoDB e scegli la tabella `lambda-apigateway`.



2. Scegli Explore table items (Esplora elementi della tabella). Nel riquadro Items returned (Elementi restituiti), dovresti vedere un elemento con l'id 5678EFGH e il numero 15.

Per leggere l'elemento nella tabella DynamoDB utilizzando curl

- Nel terminale o nel prompt dei comandi, esegui il comando `curl` seguente per leggere il valore dell'elemento appena creato. Utilizzo dei propri URL di richiamo

Linux/macOS

```
curl https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager \
-H "Content-Type: application/json" \
-d '{"operation": "read", "payload": {"Key": {"id": "5678EFGH"}}}'
```

PowerShell

```
curl.exe 'https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager' -H 'Content-Type: application/json' -d '{"operation": "read", "payload": {"Key": {"id": "5678EFGH"}}}'
```

Dovresti vedere un output come uno dei seguenti a seconda che tu abbia scelto il codice funzione Node.js o Python:

Node.js

```
{"$metadata":  
{"statusCode":200,"requestId":"7BP3G5Q0C001E50FBQI9NS099JVV4KQNS05AEMVJF66Q9ASUAAJG",  
"attempts":1,"totalRetryDelay":0},"Item":{"id":"5678EFGH","number":15}}
```

Python

```
{"Item":{"id":"5678EFGH","number":15},"ResponseMetadata":  
{"RequestId":"QNDJICE52E86B82VETR6RKBE5BVV4KQNS05AEMVJF66Q9ASUAAJG",  
"HTTPStatusCode":200,"HTTPHeaders":{"server":"Server","date":"Wed, 31 Jul 2024  
00:37:01 GMT","content-type":"application/x-amz-json-1.0",  
"content-length":"52","connection":"keep-alive","x-amzn-  
requestid":"QNDJICE52E86B82VETR6RKBE5BVV4KQNS05AEMVJF66Q9ASUAAJG","x-amz-  
crc32":"2589610852"},
```

```
"RetryAttempts":0}}
```

Per aggiornare l'elemento nella tabella DynamoDB utilizzando curl

1. Nel terminale o nel prompt dei comandi, esegui il comando `curl` seguente per aggiornare l'elemento appena creato modificando il valore `number`. Utilizzo dei propri URL di richiamo

Linux/macOS

```
curl https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager \
-H "Content-Type: application/json" \
-d '{"operation": "update", "payload": {"Key": {"id": "5678EFGH"},
  "UpdateExpression": "SET #num = :new_value", "ExpressionAttributeNames":
  {"#num": "number"}, "ExpressionAttributeValues": {":new_value": 42}}}'
```

PowerShell

```
curl.exe 'https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/
DynamoDBManager' -H 'Content-Type: application/json' -d '{"operation":
  "update", "payload": {"Key": {"id": "5678EFGH"}, "UpdateExpression
  ": "SET #num = :new_value", "ExpressionAttributeNames": {"#num": "number
  "}, "ExpressionAttributeValues": {":new_value": 42}}}'
```

2. Per verificare che il valore di `number` per l'elemento sia stato aggiornato, esegui un altro comando di lettura:

Linux/macOS

```
curl https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager \
-H "Content-Type: application/json" \
-d '{"operation": "read", "payload": {"Key": {"id": "5678EFGH"}}}'
```

PowerShell

```
curl.exe 'https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/
DynamoDBManager' -H 'Content-Type: application/json' -d '{"operation": "read
  ", "payload": {"Key": {"id": "5678EFGH"}}}'
```

Per eliminare l'elemento nella tabella DynamoDB utilizzando curl

1. Nel terminale o nel prompt dei comandi, esegui il comando `curl` seguente per eliminare l'elemento appena creato. Utilizzo dei propri URL di richiamo

Linux/macOS

```
curl https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager \
-H "Content-Type: application/json" \
-d '{"operation": "delete", "payload": {"Key": {"id": "5678EFGH"}}}'
```

PowerShell

```
curl.exe 'https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager' -H 'Content-Type: application/json' -d '{"operation": "delete", "payload": {"Key": {"id": "5678EFGH"}}}'
```

2. Verifica che l'operazione di eliminazione è andata a buon fine. Nel riquadro Elementi restituiti della pagina Esplora elementi della console DynamoDB, verifica che l'elemento con id 5678EFGH non sia più presente nella tabella.

Pulizia delle risorse (facoltativo)

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti inutili addebiti al tuo Account AWS

Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **confirm** nel campo di immissione del testo, quindi scegli Elimina.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina [Ruoli](#) della console IAM.
2. Selezionare il ruolo di esecuzione creato.

3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare l'API

1. Aprire la [pagina API](#) della console API Gateway.
2. Selezionare l'API creata.
3. Scegliere Actions (Operazioni), Delete (Elimina).
4. Scegliere Delete (Elimina).

Per eliminare la tabella DynamoDB

1. Aprire la pagina [Tables \(Tabelle\)](#) della console DynamoDB.
2. Selezionare la tabella creata.
3. Scegliere Delete (Elimina).
4. Immettere **delete** nella casella di testo.
5. Seleziona Delete Table (Elimina tabella).

Se riscontri errori quando richiami la tua funzione tramite API Gateway, consulta [the section called "Errori"](#)

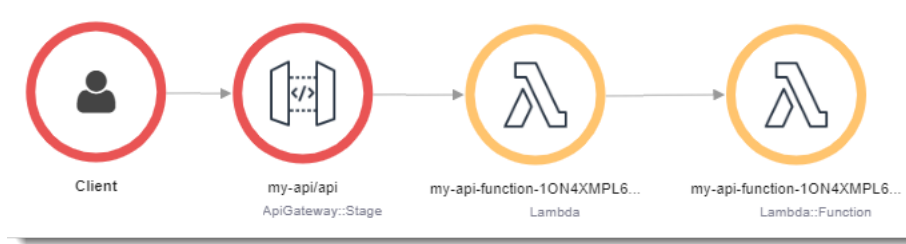
## Gestione degli errori con un'API di API Gateway

API Gateway considera tutti gli errori di invocazione e di funzione come errori interni. Se l'API Lambda rifiuta la richiesta di invocazione, API Gateway restituisce un codice di errore 500. Se la funzione viene eseguita ma restituisce un errore o una risposta nel formato errato, API Gateway restituisce il codice 502. In entrambi i casi, il corpo della risposta da API Gateway è `{"message": "Internal server error"}`.

### Note

API Gateway non riprova le invocazioni Lambda. Se Lambda restituisce un errore, API Gateway restituisce una risposta di errore al client.

L'esempio seguente mostra una mappa di X-Ray tracciamento per una richiesta che ha provocato un errore di funzione e un 502 da API Gateway. Il client riceve il messaggio di errore generico.

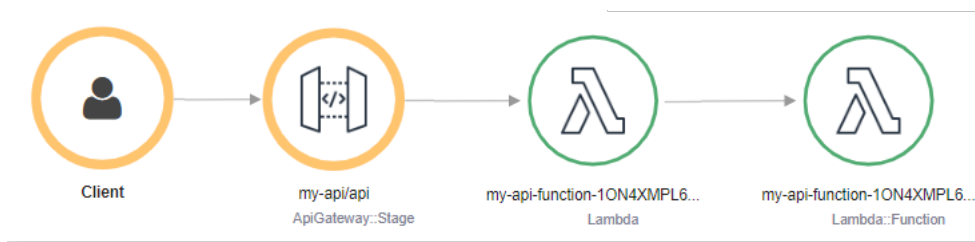


Per personalizzare la risposta di errore, è necessario rilevare gli errori nel codice e formattare una risposta nel formato richiesto.

Example [index.mjs](#) — Errore di formattazione

```
var formatError = function(error){
  var response = {
    "statusCode": error.statusCode,
    "headers": {
      "Content-Type": "text/plain",
      "x-amzn-ErrorType": error.code
    },
    "isBase64Encoded": false,
    "body": error.code + ": " + error.message
  }
  return response
}
```

API Gateway converte questa risposta in un errore HTTP con un codice di stato personalizzato e un corpo. Nella mappa di tracciamento, il nodo della funzione è verde perché ha gestito l'errore.



Selezionare un metodo per richiamare la funzione Lambda tramite una richiesta HTTP

Numerosi casi d'uso comuni per Lambda implicano l'invocazione della funzione tramite una richiesta HTTP. Ad esempio, potrebbe essere preferibile che un'applicazione Web invochi la funzione tramite

una richiesta del browser. Le funzioni Lambda possono essere utilizzate anche per creare REST API complete, gestire le interazioni degli utenti da app mobili, elaborare dati da servizi esterni tramite chiamate HTTP o creare webhook personalizzati.

Le sezioni seguenti spiegano quali sono le tue scelte per richiamare Lambda tramite HTTP e forniscono informazioni per aiutarti a prendere la decisione giusta per il tuo caso d'uso particolare.

Quali sono le tue scelte quando selezioni un metodo di invocazione HTTP?

Lambda offre due metodi principali per richiamare una funzione utilizzando una richiesta HTTP: [URL di funzione](#) e [API Gateway](#). Le differenze tra queste due opzioni sono le seguenti:

- Gli URL delle funzioni Lambda forniscono un endpoint HTTP semplice e diretto per una funzione Lambda. Sono ottimizzati per semplicità ed economicità e forniscono il percorso più veloce per esporre una funzione Lambda tramite HTTP.
- API Gateway è un servizio più avanzato per la creazione di API complete. API Gateway è ottimizzato per la creazione e la gestione di API di produzione su larga scala e fornisce strumenti completi per la sicurezza, il monitoraggio e la gestione del traffico.

Suggerimenti se conosci già le tue esigenze

Se hai già le idee chiare sulle tue esigenze, ecco i nostri suggerimenti di base:

Consigliamo gli URL delle [funzioni](#) per applicazioni semplici o prototipi in cui sono necessari solo metodi e request/response gestione di autenticazione di base e dove si desidera ridurre al minimo i costi e la complessità.

[API Gateway](#) è la scelta migliore per le applicazioni di produzione su larga scala o per i casi in cui sono necessarie funzionalità più avanzate come il [supporto per](#) OpenAPI Description, una scelta di opzioni di autenticazione, nomi di dominio personalizzati o una request/response gestione avanzata che include limitazione, memorizzazione nella cache e trasformazione. request/response

Cosa considerare quando si seleziona un metodo per richiamare la funzione Lambda

Quando si sceglie tra le funzioni URL e API Gateway, è necessario considerare i seguenti fattori:

- Le tue esigenze di autenticazione, ad esempio se richiedi OAuth o Amazon Cognito per autenticare gli utenti
- I tuoi requisiti di scalabilità e la complessità dell'API che desideri implementare

- Se hai bisogno di funzionalità avanzate come la convalida e la formattazione delle richieste request/response
- I tuoi requisiti di monitoraggio
- I tuoi obiettivi di costo

Comprendendo questi fattori, è possibile selezionare l'opzione che meglio bilancia i requisiti di sicurezza, complessità e costi.

Le informazioni seguente riepilogano le differenze principali tra le due opzioni.

### Autenticazione

- Gli URL delle funzioni forniscono opzioni di autenticazione di base tramite AWS Identity and Access Management (IAM). Puoi configurare gli endpoint in modo che siano pubblici (senza autenticazione) o che richiedano l'autenticazione IAM. Con l'autenticazione IAM, puoi utilizzare AWS credenziali standard o ruoli IAM per controllare l'accesso. Sebbene sia semplice da configurare, questo approccio offre opzioni limitate rispetto ad altri metodi di autenticazione.
- API Gateway fornisce l'accesso a una gamma più completa di opzioni di autenticazione. Oltre all'autenticazione IAM, puoi utilizzare autorizzatori [Lambda](#) (logica di autenticazione personalizzata), pool di [utenti](#) Amazon Cognito e flussi. OAuth2.0 Questa flessibilità consente di implementare schemi di autenticazione complessi, inclusi provider di autenticazione di terze parti, autenticazione basata su token e autenticazione a più fattori.

### Request/response gestione

- Le funzioni URL forniscono la gestione di base delle richieste e delle risposte HTTP. Supportano i metodi HTTP standard e includono il supporto integrato per CORS (cross-origin resource sharing). Sebbene siano in grado di gestire i payload JSON e i parametri di query in modo naturale, non offrono funzionalità di trasformazione o convalida delle richieste. La gestione delle risposte è altrettanto semplice: il client riceve la risposta dalla funzione Lambda esattamente come la restituisce Lambda.
- API Gateway offre sofisticate funzionalità di gestione delle richieste e delle risposte. È possibile definire validatori di richieste, trasformare richieste e risposte utilizzando modelli di mappatura, impostare le request/response intestazioni e implementare la memorizzazione nella cache delle risposte. API Gateway supporta anche payload binari e nomi di dominio personalizzati e può modificare le risposte prima che raggiungano il client. È possibile impostare modelli per la request/response convalida e la trasformazione utilizzando JSON Schema.

## Dimensionamento

- Le funzioni URL si adattano direttamente ai limiti di simultaneità della funzione Lambda e gestiscono i picchi di traffico scalando la funzione fino al limite di simultaneità massimo configurato. Una volta raggiunto tale limite, Lambda risponde alle richieste aggiuntive con risposte HTTP 429. Non esiste un meccanismo di accodamento integrato, pertanto la gestione della scalabilità dipende interamente dalla configurazione della funzione Lambda. Per impostazione predefinita, le funzioni Lambda hanno un limite di 1.000 esecuzioni simultanee per Regione AWS.
- API Gateway offre funzionalità di scalabilità aggiuntive oltre alla scalabilità propria di Lambda. Include controlli integrati per l'accodamento e la limitazione delle richieste che consentono di gestire i picchi di traffico con maggiore efficienza. Per impostazione predefinita, API Gateway è in grado di gestire fino a 10.000 richieste al secondo per regione, con una capacità di espansione di 5.000 richieste al secondo. Fornisce inoltre strumenti per limitare le richieste a diversi livelli (API, fase o metodo) per proteggere il backend.

## Monitoraggio

- Gli URL delle funzioni offrono un monitoraggio di base tramite le CloudWatch metriche di Amazon, tra cui il numero di richieste, la latenza e i tassi di errore. Puoi accedere a parametri e log Lambda standard, che mostrano le richieste non elaborate che arrivano alla tua funzione. Sebbene ciò fornisca una visibilità operativa essenziale, i parametri si concentrano principalmente sull'esecuzione delle funzioni.
- API Gateway offre funzionalità di monitoraggio complete, tra cui parametri dettagliati, opzioni di registrazione e tracciamento. Puoi monitorare le chiamate API, la latenza, i tassi di errore e i tassi di cache tramite hit/miss CloudWatch API Gateway si integra anche con il AWS X-Ray tracciamento distribuito e fornisce formati di registrazione personalizzabili.

## Costo

- Le funzioni URL seguono il modello di prezzo standard di Lambda: paghi solo per le invocazioni di funzione e il tempo di calcolo. Non sono previsti costi aggiuntivi per l'endpoint URL stesso. Ciò lo rende una scelta conveniente per API semplici o applicazioni a basso traffico se non sono necessarie le funzionalità aggiuntive di API Gateway.
- API Gateway offre un [piano gratuito](#) che include un milione di chiamate API ricevute per le REST API e un milione di chiamate API ricevute per le API HTTP. Successivamente, API Gateway addebita i costi per le chiamate API, il trasferimento dei dati e la memorizzazione nella cache (se



abilitata). Consulta la [pagina dei prezzi](#) di API Gateway per conoscere i costi relativi al tuo caso d'uso.

## Altre funzionalità

- Le funzioni URL sono progettati per la semplicità e l'integrazione diretta con Lambda. Supportano sia endpoint HTTP che HTTPS, offrono supporto CORS integrato e forniscono endpoint dual stack (IPv4 e IPv6). Sebbene non dispongano di funzionalità avanzate, eccellono negli scenari in cui è necessario un modo rapido e diretto per esporre le funzioni Lambda tramite HTTP.
- API Gateway include numerose funzionalità aggiuntive come il controllo delle versioni delle API, la gestione delle fasi, le chiavi API per i piani di utilizzo, la documentazione delle API tramite, le API Swagger/OpenAPI, WebSocket le API private all'interno di un VPC e l'integrazione WAF per una maggiore sicurezza. Supporta inoltre implementazioni canarie, integrazioni simulate per i test e l'integrazione con altre applicazioni oltre a Lambda. Servizi AWS

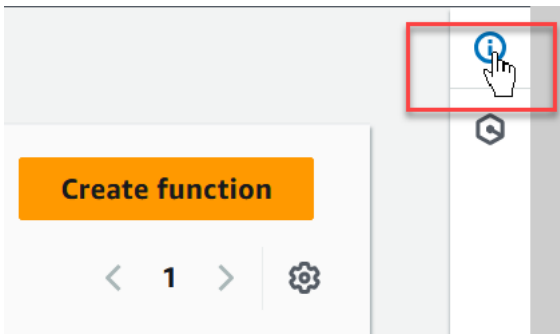
## Selezionare un metodo per richiamare la funzione Lambda

Ora che hai letto i criteri per la selezione tra le funzioni Lambda URL e l'API Gateway e le principali differenze tra questi, puoi selezionare l'opzione più adatta alle tue esigenze e utilizzare le seguenti risorse per iniziare a utilizzarla.

## Function URLs

Iniziare a usare le funzioni URL con le seguenti risorse

- Segui il tutorial [Creazione di una funzione Lambda con la funzione URL](#)
- Scopri di più sulle funzioni URL nel capitolo [the section called “URL delle funzioni”](#) di questa guida
- Prova il tutorial guidato dalla console Creare una semplice app Web effettuando le seguenti operazioni:
  1. Apri la pagina [Funzioni](#) della console Lambda.
  2. Apri il pannello di aiuto scegliendo l'icona nell'angolo in alto a destra della schermata.



3. Seleziona Tutorial.
4. In Crea una semplice app Web, scegli Avvia tutorial.

## API Gateway

Iniziare a usare Lambda e API Gateway con le seguenti risorse

- Segui il tutorial [Utilizzo di Lambda con API Gateway](#) per creare una REST API integrata con una funzione Lambda di backend.
- Scopri di più sui diversi tipi di API offerti da API Gateway nelle seguenti sezioni della Guida per gli sviluppatori di Gateway Amazon API:
  - [REST API di API Gateway](#)
  - [API HTTP di API Gateway](#)
  - [WebSocket API Gateway](#)
- Prova uno o più esempi nella sezione [Tutorial e workshop](#) della Guida per gli sviluppatori di Gateway Amazon API.

## Utilizzo AWS Lambda con AWS Infrastructure Composer

AWS Infrastructure Composer è un visual builder per la progettazione di applicazioni moderne su AWS. Progetta l'architettura della tua applicazione trascinando, raggruppando e connettendoti Servizi AWS in una tela visiva. Infrastructure Composer crea modelli di infrastructure as code (IaC) a partire dal tuo progetto che puoi implementare utilizzando [AWS SAM](#) oppure [CloudFormation](#).

### Esportazione di una funzione Lambda in Infrastructure Composer

Per iniziare a utilizzare Infrastructure Composer, crea un nuovo progetto basato sulla configurazione di una funzione Lambda esistente utilizzando la console Lambda. Per esportare la configurazione e il codice della funzione in Infrastructure Composer per creare un nuovo progetto, procedi come segue:

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Seleziona la funzione che desideri utilizzare come base per il tuo progetto Infrastructure Composer.
3. Nel riquadro Panoramica della funzione, scegli Esporta in Infrastructure Composer.

Per esportare la configurazione e il codice della funzione in Infrastructure Composer, Lambda crea un bucket Amazon S3 nel tuo account in cui archiviare temporaneamente questi dati.

4. Nella finestra di dialogo, scegli Conferma e crea progetto per accettare il nome predefinito per questo bucket ed esportare la configurazione e il codice della funzione in Infrastructure Composer.
5. (Facoltativo) Per scegliere un altro nome per il bucket Amazon S3 creato da Lambda, immetti un nuovo nome e scegli Conferma e crea progetto. I nomi dei bucket Amazon S3 devono essere univoci a livello globale e seguire le [regole di denominazione dei bucket](#).
6. Per salvare i file di progetto e funzione in Infrastructure Composer, attiva la [modalità di sincronizzazione locale](#).

#### Note

Se hai già utilizzato la funzionalità Esporta in Strumento per la creazione di applicazioni e hai creato un bucket Amazon S3 utilizzando il nome predefinito, Lambda può riutilizzare questo bucket, se esiste ancora. Accetta il nome predefinito del bucket nella finestra di dialogo per riutilizzare il bucket esistente.

## Configurazione del bucket di trasferimento Amazon S3

Il bucket Amazon S3 creato da Lambda per trasferire la configurazione della funzione crittografa automaticamente gli oggetti utilizzando lo standard di crittografia AES 256. Lambda configura inoltre il bucket in modo che utilizzi la condizione del proprietario del [bucket](#) per garantire che solo tu Account AWS sia in grado di aggiungere oggetti al bucket.

Lambda configura il bucket per eliminare automaticamente gli oggetti 10 giorni dopo il caricamento. Tuttavia, Lambda non elimina automaticamente il bucket stesso. Per eliminare il bucket dal tuo Account AWS, segui le istruzioni in Eliminazione di un bucket. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/delete-bucket.html> Il nome predefinito del bucket utilizza il prefisso `lambdasam`, una stringa alfanumerica di 10 cifre e la funzione creata in: Regione AWS

```
lambdasam-06f22da95b-us-east-1
```

Per evitare di aggiungere costi aggiuntivi al tuo Account AWS, ti consigliamo di eliminare il bucket Amazon S3 non appena hai finito di esportare la tua funzione in Infrastructure Composer.

Si applicano i [prezzi standard di Amazon S3](#).

#### Autorizzazioni richieste

Per utilizzare l'integrazione Lambda con la funzionalità Infrastructure Composer, sono necessarie determinate autorizzazioni per scaricare un AWS SAM modello e scrivere la configurazione della funzione su Amazon S3.

Per scaricare un AWS SAM modello, devi disporre dell'autorizzazione per utilizzare le seguenti azioni API:

- [GetPolicy](#)
- [sono: GetPolicyVersion](#)
- [sono: GetRole](#)
- [sono: GetRolePolicy](#)
- [sono: ListAttachedRolePolicies](#)
- [sono: ListRolePolicies](#)
- [sono: ListRoles](#)

Puoi concedere il permesso di utilizzare tutte queste azioni aggiungendo la policy Lambda\_ReadOnlyAccess [AWS gestita da](#) AWS al tuo ruolo utente IAM.

Affinché Lambda scriva la configurazione della tua funzione su Amazon S3, devi disporre dell'autorizzazione a utilizzare le seguenti operazioni dell'API:

- [S3:PutObject](#)
- [S3:CreateBucket](#)
- [S3:PutBucketEncryption](#)
- [S3:PutBucketLifecycleConfiguration](#)

Se non riesci a esportare la configurazione della tua funzione in Infrastructure Composer, verifica che il tuo account disponga delle autorizzazioni necessarie per queste operazioni. Se disponi delle

autorizzazioni richieste ma non riesci comunque a esportare la configurazione della funzione, verifica se ne sono presenti [policy basate sulle risorse](#) che potrebbero limitare l'accesso ad Amazon S3.

## Altre risorse

Per un tutorial più dettagliato su come progettare un'applicazione serverless in Infrastructure Composer basata su una funzione Lambda esistente, consulta [the section called “Infrastructure as code \(IaC\)”](#).

Per utilizzare Infrastructure Composer e AWS SAM progettare e distribuire un'applicazione serverless completa utilizzando Lambda, puoi anche seguire il [AWS Infrastructure Composer tutorial del Serverless Patterns](#) Workshop. [AWS](#)

## Utilizzo AWS Lambda con CloudFormation

In un AWS CloudFormation modello, puoi specificare una funzione Lambda come destinazione di una risorsa personalizzata. Utilizza risorse personalizzate per elaborare parametri, recuperare valori di configurazione o chiamare altre persone Servizi AWS durante gli eventi del ciclo di vita dello stack.

L'esempio seguente invoca una funzione definita in un altro punto del modello.

### Example– Definizione di risorse personalizzate

```
Resources:
  primerinvoke:
    Type: AWS::CloudFormation::CustomResource
    Version: "1.0"
    Properties:
      ServiceToken: !GetAtt primer.Arn
      FunctionName: !Ref randomerror
```

Il token di servizio è l'Amazon Resource Name (ARN) della funzione che CloudFormation richiama quando crei, aggiorni o elimini lo stack. Puoi anche includere proprietà aggiuntive come `FunctionName`, che vengono CloudFormation passate alla funzione così com'è.

CloudFormation richiama la funzione Lambda in modo [asincrono](#) con un evento che include un URL di callback.

### Example— CloudFormation evento del messaggio

```
{
  "RequestType": "Create",
  "ServiceToken": "arn:aws:lambda:us-east-1:123456789012:function:lambda-error-processor-primer-14R0R2T3JKU66",
  "ResponseURL": "https://cloudformation-custom-resource-response-useast1.s3-us-east-1.amazonaws.com/arn%3Aaws%3Acloudformation%3Aus-east-1%3A123456789012%3Astack/lambda-error-processor/1134083a-2608-1e91-9897-022501a2c456%7Cprimerinvoke%7C5d478078-13e9-baf0-464a-7ef285ecc786?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Expires=1555451971&Signature=28UijZePE5I4dvukKQqM%2F9Rf1o4%3D",
  "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/lambda-error-processor/1134083a-2608-1e91-9897-022501a2c456",
  "RequestId": "5d478078-13e9-baf0-464a-7ef285ecc786",
  "LogicalResourceId": "primerinvoke",
  "ResourceType": "AWS::CloudFormation::CustomResource",
```

```

    "ResourceProperties": {
      "ServiceToken": "arn:aws:lambda:us-east-1:123456789012:function:lambda-error-processor-primer-14R0R2T3JKU66",
      "FunctionName": "lambda-error-processor-randomerror-ZWUC391MQAJK"
    }
  }
}

```

Da questa funzione dipende la risposta all'URL di callback che indica un esito positivo o negativo. Per la sintassi di risposta completa, consulta [Oggetti di risposta delle risorse personalizzate](#).

Example— risposta CloudFormation personalizzata delle risorse

```

{
  "Status": "SUCCESS",
  "PhysicalResourceId": "2019/04/18/[$LATEST]b3d1bfc65f19ec610654e4d9b9de47a0",
  "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/lambda-error-processor/1134083a-2608-1e91-9897-022501a2c456",
  "RequestId": "5d478078-13e9-baf0-464a-7ef285ecc786",
  "LogicalResourceId": "primerinvoke"
}

```

CloudFormation fornisce una libreria chiamata `cfn-response` che gestisce l'invio della risposta. Se definisci la tua funzione all'interno di un modello, puoi richiedere la libreria per nome. CloudFormation quindi aggiunge la libreria al pacchetto di distribuzione creato per la funzione.

Se alla funzione utilizzata da una risorsa personalizzata è collegata un'[interfaccia di rete elastica](#), aggiungi le seguenti risorse alla policy VPC, dove **region** è la regione in cui si trova la funzione senza i trattini. Ad esempio, `us-east-1` è `useast1`. Ciò consentirà alla risorsa personalizzata di rispondere all'URL di callback che invia un segnale allo CloudFormation stack.

```

arn:aws:s3:::cloudformation-custom-resource-response-region",
"arn:aws:s3:::cloudformation-custom-resource-response-region/*",

```

La seguente funzione di esempio invoca una seconda funzione. Se la chiamata ha esito positivo, la funzione invia una risposta positiva a CloudFormation e l'aggiornamento dello stack continua. Il modello utilizza il tipo di [AWS::Serverless::Function](#) risorsa fornito da AWS Serverless Application Model

Example: funzione relativa alle risorse personalizzate

```

Transform: 'AWS::Serverless-2016-10-31'

```

```
Resources:
  primer:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs16.x
      InlineCode: |
        var aws = require('aws-sdk');
        var response = require('cfn-response');
        exports.handler = function(event, context) {
          // For Delete requests, immediately send a SUCCESS response.
          if (event.RequestType == "Delete") {
            response.send(event, context, "SUCCESS");
            return;
          }
          var responseStatus = "FAILED";
          var responseData = {};
          var functionName = event.ResourceProperties.FunctionName
          var lambda = new aws.Lambda();
          lambda.invoke({ FunctionName: functionName }, function(err, invokeResult) {
            if (err) {
              responseData = {Error: "Invoke call failed"};
              console.log(responseData.Error + ":\n", err);
            }
            else responseStatus = "SUCCESS";
            response.send(event, context, responseStatus, responseData);
          });
        };
      Description: Invoke a function to create a log stream.
      MemorySize: 128
      Timeout: 8
      Role: !GetAtt role.Arn
      Tracing: Active
```

Se la funzione richiamata dalla risorsa personalizzata non è definita in un modello, puoi ottenere il codice sorgente `cfn-response` dal [modulo `cfn-response`](#) nella Guida per l'utente. AWS CloudFormation

Per ulteriori informazioni sulle risorse personalizzate, consulta [Risorse personalizzate](#) nella Guida per l'utente di AWS CloudFormation .



## Elaborare gli eventi Amazon DocumentDB con Lambda

Per elaborare gli eventi in un [flusso di modifica di Amazon DocumentDB \(compatibile con MongoDB\)](#), puoi utilizzare una funzione Lambda configurando un cluster Amazon DocumentDB come origine degli eventi. Successivamente puoi automatizzare i carichi di lavoro basati sugli eventi invocando la funzione Lambda con il cluster Amazon DocumentDB ogni volta che i dati cambiano.

### Note

Lambda supporta solo le versioni 4.0 e 5.0 di Amazon DocumentDB. Lambda non supporta la versione 3.6.

Inoltre, per gli strumenti di mappatura dell'origine degli eventi, Lambda supporta solo cluster basati su istanze e cluster regionali. Lambda non supporta [cluster elastici](#) o [cluster globali](#). Questa limitazione non si applica quando si utilizza Lambda come client per connettersi ad Amazon DocumentDB. Lambda può connettersi a tutti i tipi di cluster per eseguire operazioni CRUD.

Lambda elabora gli eventi di Amazon DocumentDB nei flussi di modifica in sequenza secondo l'ordine in cui arrivano. Per questo motivo, la tua funzione può gestire solo una invocazione simultanea da Amazon DocumentDB alla volta. Per monitorare la tua funzione, puoi tenere traccia dei relativi [parametri di simultaneità](#).

### Warning

Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per saperne di più, consulta [Come posso rendere idempotente la mia funzione Lambda](#) nel Knowledge Center. AWS

### Argomenti

- [Esempio di evento Amazon DocumentDB](#)
- [Prerequisiti e autorizzazioni](#)
- [Configurare la sicurezza della rete](#)

- [Creazione di una mappatura dell'origine degli eventi Amazon DocumentDB \(console\)](#)
- [Creazione di una mappatura dell'origine degli eventi Amazon DocumentDB \(SDK o CLI\)](#)
- [Posizioni di partenza di polling e flussi](#)
- [Monitoraggio dell'origine degli eventi di Amazon DocumentDB](#)
- [Tutorial: Utilizzo AWS Lambda con Amazon DocumentDB Streams](#)

## Esempio di evento Amazon DocumentDB

```
{
  "eventSourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:canaryclusterb2a659a2-qo5tcmqkc103",
  "events": [
    {
      "event": {
        "_id": {
          "_data": "0163eeb6e7000000090100000009000041e1"
        },
        "clusterTime": {
          "$timestamp": {
            "t": 1676588775,
            "i": 9
          }
        },
        "documentKey": {
          "_id": {
            "$oid": "63eeb6e7d418cd98afb1c1d7"
          }
        },
        "fullDocument": {
          "_id": {
            "$oid": "63eeb6e7d418cd98afb1c1d7"
          },
          "anyField": "sampleValue"
        },
        "ns": {
          "db": "test_database",
          "coll": "test_collection"
        },
        "operationType": "insert"
      }
    }
  ]
}
```

```
],  
  "eventSource": "aws:docdb"  
}
```

Per ulteriori informazioni sugli eventi in questo esempio e sulle loro forme, consulta la sezione [Eventi di modifica](#) sul sito Web della documentazione di MongoDB.

## Prerequisiti e autorizzazioni

Prima di utilizzare Amazon DocumentDB come origine degli eventi della funzione Lambda, è necessario tenere a mente i seguenti prerequisiti. Devi:

- Disponi di un cluster Amazon DocumentDB esistente nella stessa Account AWS funzione e nella tua funzione. Regione AWS Se non hai un cluster esistente, puoi crearlo seguendo i passaggi riportati nella sezione [Nozioni di base su Amazon DocumentDB](#) nella Guida per gli sviluppatori di Amazon DocumentDB. In alternativa, la prima serie di passaggi riportati in [Tutorial: Utilizzo AWS Lambda con Amazon DocumentDB Streams](#) ti guideranno nella creazione di un cluster Amazon DocumentDB con tutti i prerequisiti necessari.
- Concedere a Lambda l'accesso alle risorse di Amazon Virtual Private Cloud (Amazon VPC) associate al cluster Amazon DocumentDB. Per ulteriori informazioni, consulta [Configurare la sicurezza della rete](#).
- Abilitare TLS sul cluster Amazon DocumentDB. Questa è l'impostazione predefinita. Se TLS è disabilitato, Lambda non è in grado di comunicare con il cluster.
- È necessario attivare i flussi di modifica sul cluster Amazon DocumentDB. Per ulteriori informazioni, consulta la sezione [Utilizzo dei flussi di modifica di Amazon DocumentDB](#) nella Guida per gli sviluppatori di Amazon DocumentDB.
- Fornire a Lambda le credenziali per accedere al cluster Amazon DocumentDB. Quando configuri l'origine degli eventi, fornisci la chiave [AWS Secrets Manager](#) che contiene i dettagli di autenticazione (nome utente e password) necessari per accedere al cluster. Per fornire questa chiave durante la configurazione, esegui una delle seguenti operazioni:
  - Se per la configurazione stai utilizzando la console Lambda, fornisci la chiave nel campo Chiave di Secrets manager.
  - Se stai usando AWS Command Line Interface (AWS CLI) per la configurazione, fornisci questa chiave nell'`source-access-configurations` opzione. È possibile includere questa opzione sia con il comando [create-event-source-mapping](#) sia con il comando [update-event-source-mapping](#). Ad esempio:

```
aws lambda create-event-source-mapping \  
    ...  
    --source-access-configurations  
    '["Type":"BASIC_AUTH","URI":"arn:aws:secretsmanager:us-  
west-2:123456789012:secret:DocDBSecret-AbC4E6"]]' \  
    ...
```

- Concedere a Lambda le autorizzazioni per gestire le risorse correlate al flusso di Amazon DocumentDB. Aggiungi le seguenti autorizzazioni al [ruolo di esecuzione](#) della tua funzione:
  - [rds:DescribeDBClusters](#)
  - [barre: DescribeDBClusterParameters](#)
  - [rossi: DescribeDBSubnetGroups](#)
  - [ec2: CreateNetworkInterface](#)
  - [ec2: DescribeNetworkInterfaces](#)
  - [ec2: DescribeVpcs](#)
  - [ec2: DeleteNetworkInterface](#)
  - [ec2: DescribeSubnets](#)
  - [ec2: DescribeSecurityGroups](#)
  - [kms:Decrypt](#)
  - [responsabile dei segreti: GetSecretValue](#)
- È necessario mantenere la dimensione degli eventi del flusso di modifica di Amazon DocumentDB che invii a Lambda inferiore a 6 MB. Lambda supporta payload di dimensioni massime di 6 MB. Se il flusso di modifica tenta di inviare a Lambda un evento di dimensioni superiori a 6 MB, Lambda tralascia il messaggio ed emette il parametro `OversizedRecordCount`. Lambda emette tutte i parametri sulla base del miglior tentativo.

#### Note

Sebbene le funzioni Lambda abbiano di solito un timeout massimo di 15 minuti, le mappature dell'origine degli eventi per Amazon MSK, Apache Kafka autogestito, Amazon DocumentDB e Amazon MQ per ActiveMQ e RabbitMQ supportano soltanto funzioni con timeout massimi di 14 minuti. Questa limitazione garantisce che lo strumento di mappatura dell'origine degli eventi possa gestire correttamente errori di funzioni e nuovi tentativi.

## Configurare la sicurezza della rete

Per concedere a Lambda l'accesso completo ad Amazon DocumentDB tramite lo strumento di mappatura dell'origine degli eventi, il cluster deve utilizzare un endpoint pubblico (indirizzo IP pubblico) oppure devi fornire l'accesso all'Amazon VPC in cui hai creato il cluster.

Quando usi Amazon DocumentDB con Lambda, crea [endpoint VPC AWS PrivateLink](#) che forniscono alla funzione l'accesso alle risorse del tuo Amazon VPC.

### Note

**AWS PrivateLink** Gli endpoint VPC sono necessari per le funzioni con mappature delle sorgenti di eventi che utilizzano la modalità predefinita (su richiesta) per i poller di eventi. Se la mappatura dell'origine degli eventi utilizza la modalità [provisioned](#), non è necessario configurare gli endpoint VPC. AWS PrivateLink

Crea un endpoint per fornire l'accesso alle seguenti risorse:

- Lambda: crea un endpoint per il principale del servizio Lambda.
- AWS STS — Crea un endpoint per consentire AWS STS a un responsabile del servizio di assumere un ruolo per tuo conto.
- Secrets Manager: se il tuo cluster utilizza Secrets Manager per archiviare le credenziali, crea un endpoint per Secrets Manager.

In alternativa, configura un gateway NAT su ogni sottorete pubblica in Amazon VPC. Per ulteriori informazioni, consulta [the section called “Accesso Internet per funzioni del VPC”](#).

Quando crei uno strumento di mappatura dell'origine degli eventi per Amazon DocumentDB, Lambda verifica se le interfacce di rete elastiche (ENI) sono già presenti per le sottoreti e i gruppi di sicurezza configurati per il tuo Amazon VPC. Se Lambda trova ENI esistenti, prova a riutilizzarle. Altrimenti, Lambda crea nuove ENI per connettersi all'origine dell'evento e richiamare la tua funzione.

### Note

Le funzioni Lambda vengono sempre eseguite all'interno di VPC di proprietà del servizio Lambda. La configurazione VPC della funzione non influisce sullo strumento di mappatura

dell'origine degli eventi. Solo la configurazione di rete dell'origine dell'evento determina il modo in cui Lambda si connette all'origine dell'evento.

Configura i gruppi di sicurezza per l'Amazon VPC contenente il tuo cluster. Per impostazione predefinita, Amazon DocumentDB utilizza le seguenti porte: 27017.

- Regole in ingresso: consenti tutto il traffico sulla porta del broker predefinita per il gruppo di sicurezza associato all'origine eventi. In alternativa, puoi utilizzare una regola autoreferenziale del gruppo di sicurezza per consentire l'accesso da istanze all'interno dello stesso gruppo di sicurezza.
- Regole in uscita: consenti tutto il traffico sulla porta 443 per destinazioni esterne se la tua funzione deve comunicare con i servizi. AWS In alternativa, puoi anche utilizzare una regola di gruppo di sicurezza autoreferenziale per limitare l'accesso al broker se non hai bisogno di comunicare con altri servizi. AWS
- Regole di ingresso degli endpoint Amazon VPC: se utilizzi un endpoint Amazon VPC, il gruppo di sicurezza associato all'endpoint Amazon VPC deve consentire il traffico in entrata sulla porta 443 dal gruppo di sicurezza del cluster.

Se il cluster utilizza l'autenticazione, puoi anche limitare la policy degli endpoint per l'endpoint Secrets Manager. Per chiamare l'API Secrets Manager, Lambda utilizza il ruolo della funzione, non il principale del servizio Lambda.

Example Policy dell'endpoint VPC: endpoint Secrets Manager

```
{
  "Statement": [
    {
      "Action": "secretsmanager:GetSecretValue",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws::iam::123456789012:role/my-role"
        ]
      },
      "Resource": "arn:aws::secretsmanager:us-west-2:123456789012:secret:my-secret"
    }
  ]
}
```

Quando utilizzi gli endpoint Amazon VPC, AWS indirizza le tue chiamate API per richiamare la tua funzione utilizzando l'Elastic Network Interface (ENI) dell'endpoint. Il principale del servizio Lambda deve chiamare `lambda:InvokeFunction` su tutti i ruoli e le funzioni che utilizzano tali ENI.

Per impostazione predefinita, gli endpoint Amazon VPC dispongono di policy IAM aperte che consentono un ampio accesso alle risorse. La best practice consiste nel limitare queste policy per eseguire le azioni necessarie utilizzando quell'endpoint. Per garantire che lo strumento di mappatura dell'origine degli eventi sia in grado di invocare la funzione Lambda, la policy degli endpoint VPC deve consentire al principale del servizio Lambda di chiamare `sts:AssumeRole` e `lambda:InvokeFunction`. Limitare le policy degli endpoint VPC per consentire solo le chiamate API provenienti dall'organizzazione impedisce il corretto funzionamento dello strumento di mappatura dell'origine degli eventi, pertanto `"Resource": "*"` è richiesto in queste policy.

Il seguente esempio di policy degli endpoint VPC mostra come concedere l'accesso richiesto al principale del servizio Lambda per gli endpoint AWS STS e Lambda.

#### Example Politica VPC Endpoint — AWS STS endpoint

```
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

#### Example Policy dell'endpoint VPC: endpoint Lambda

```
{
  "Statement": [
    {
      "Action": "lambda:InvokeFunction",
      "Effect": "Allow",
      "Principal": {
```

```
        "Service": [
            "lambda.amazonaws.com"
        ],
        "Resource": "*"
    }
}
```

## Creazione di una mappatura dell'origine degli eventi Amazon DocumentDB (console)

Per configurare una funzione Lambda per la lettura dal flusso di modifica di un cluster Amazon DocumentDB, crea una [mappatura dell'origine degli eventi](#). In questa sezione viene descritto come eseguire questa operazione dalla console Lambda. Per AWS SDK e AWS CLI istruzioni, consulta [the section called “Creazione di una mappatura dell'origine degli eventi Amazon DocumentDB \(SDK o CLI\)”](#)

### Creazione di una mappatura dell'origine degli eventi per Amazon DocumentDB (console)

1. Aprire la pagina [Functions](#) (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. In Function overview (Panoramica delle funzioni), scegliere Add trigger (Aggiungi trigger).
4. In Configurazione del trigger, nell'elenco a discesa, scegli DocumentDB.
5. Configurare le opzioni richieste, quindi scegliere Add (Aggiungi).

Lambda supporta le seguenti opzioni per le origini degli eventi Amazon DocumentDB:

- Cluster DocumentDB: seleziona un cluster Amazon DocumentDB.
- Attiva il trigger: seleziona l'opzione se vuoi attivare il trigger immediatamente. Se selezioni questa casella, la funzione inizia immediatamente a ricevere traffico dal flusso di modifica specificato di Amazon DocumentDB al momento della creazione della mappatura dell'origine degli eventi. Ai fini del test, è preferibile deselezionare la casella in modo da creare una mappatura dell'origine degli eventi non attiva. Dopo la creazione, puoi attivare lo strumento di mappatura dell'origine degli eventi in qualsiasi momento.
- Nome database: immetti il nome di un database da utilizzare nel cluster.
- (Facoltativo) Nome della raccolta: immetti il nome di una raccolta da utilizzare nel database. Se non specifichi una raccolta, Lambda ascolta tutti gli eventi di ciascuna raccolta del database.



- **Dimensioni batch:** imposta, fino a 10.000, il numero massimo di messaggi da recuperare in un singolo batch. La dimensione predefinita del batch è pari a 100.
- **Posizione iniziale:** scegli la posizione del flusso da cui iniziare a leggere i record.
  - **Ultimi:** elabora solo i nuovi record aggiunti al flusso. La funzione inizia a elaborare i record solo dopo che Lambda ha terminato la creazione dell'origine degli eventi. Ciò significa che alcuni record potrebbero essere eliminati fino a quando la fonte dell'evento non viene creata correttamente.
  - **Trim Horizon (Orizzonte di taglio):** elabora tutti i record contenuti nel flusso. Lambda utilizza la durata di conservazione dei log del tuo cluster per stabilire da dove iniziare a leggere gli eventi. In particolare Lambda inizia a leggere da `current_time - log_retention_duration`. Il flusso di modifica deve essere già attivo prima di questo timestamp affinché Lambda legga tutti gli eventi correttamente.
  - **At timestamp (Al timestamp):** elaborare record a partire da una determinata ora. Il flusso di modifica deve essere già attivo prima del timestamp specificato affinché Lambda legga tutti gli eventi correttamente.
- **.Autenticazione:** scegli il metodo di autenticazione per l'accesso dei broker al cluster.
  - **BASIC\_AUTH:** con l'autenticazione di base è necessario fornire la chiave Secrets Manager che contiene le credenziali per accedere al cluster.
- **Chiave Secrets Manager:** scegli la chiave Secrets Manager che contiene i dettagli di autenticazione (nome utente e password) necessari per accedere al cluster Amazon DocumentDB.
- **(Facoltativo) Finestra batch:** specifica il tempo massimo, espresso in secondi fino a un massimo di 300, per la raccolta dei record prima che la funzione venga richiamata.
- **(Facoltativo) Configurazione completa del documento:** per le operazioni di aggiornamento del documento, scegli che cosa inviare al flusso. Il valore predefinito è `Default`, il che significa che Amazon DocumentDB invia solo un delta che descrive le modifiche apportate per ogni evento del flusso di modifica. Per ulteriori informazioni su questo campo, consulta la documentazione dell'[FullDocument](#) API Javadoc di MongoDB.
  - **Impostazione predefinita:** Lambda invia solo un documento parziale che descrive le modifiche apportate.
  - **UpdateLookup**— Lambda invia un delta che descrive le modifiche, insieme a una copia dell'intero documento.

## Creazione di una mappatura dell'origine degli eventi Amazon DocumentDB (SDK o CLI)

Per creare o gestire una mappatura dell'origine degli eventi Amazon DocumentDB tramite un [SDK AWS](#), puoi utilizzare le seguenti operazioni dell'API:

- [CreateEventSourceMapping](#)
- [ListEventSourceMappings](#)
- [GetEventSourceMapping](#)
- [UpdateEventSourceMapping](#)
- [DeleteEventSourceMapping](#)

Per creare la mappatura della sorgente dell'evento con AWS CLI, usa il comando. [create-event-source-mapping](#) L'esempio seguente utilizza questo comando per mappare una funzione denominata `my-function` a un flusso di modifica Amazon DocumentDB. L'origine degli eventi è indicata da un nome della risorsa Amazon (ARN), con una dimensione batch di 500, a partire dal timestamp in formato Unix. Il comando specifica anche la chiave Secrets Manager che Lambda utilizza per connettersi ad Amazon DocumentDB. Inoltre, include `document-db-event-source-config` parametri che specificano il database e la raccolta da cui leggere.

```
aws lambda create-event-source-mapping --function-name my-function \
    --event-source-arn arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-
    epzcyvu4pjoy
    --batch-size 500 \
    --starting-position AT_TIMESTAMP \
    --starting-position-timestamp 1541139109 \
    --source-access-configurations
    '[{"Type":"BASIC_AUTH","URI":"arn:aws:secretsmanager:us-
    east-1:123456789012:secret:DocDBSecret-BAtjxi"}]' \
    --document-db-event-source-config '{"DatabaseName":"test_database",
    "CollectionName": "test_collection"}' \
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{
  "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
  "BatchSize": 500,
  "DocumentDBEventSourceConfig": {
    "CollectionName": "test_collection",
```

```

    "DatabaseName": "test_database",
    "FullDocument": "Default"
  },
  "MaximumBatchingWindowInSeconds": 0,
  "EventSourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-epzcyvu4pjjoy",
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
  "LastModified": 1541348195.412,
  "LastProcessingResult": "No records processed",
  "State": "Creating",
  "StateTransitionReason": "User action"
}

```

Dopo la creazione, puoi utilizzare il comando [update-event-source-mapping](#) per modificare le impostazioni di aggiornamento relative all'origine degli eventi di Amazon DocumentDB. Nell'esempio seguente, la dimensione del batch viene aggiornata a 1.000 e la finestra batch a 10 secondi. Per questo comando è necessario l'UUID della mappatura dell'origine degli eventi, recuperabile utilizzando il comando `list-event-source-mapping` o dalla console Lambda.

```

aws lambda update-event-source-mapping --function-name my-function \
  --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
  --batch-size 1000 \
  --batch-window 10

```

L'output visualizzato dovrebbe essere di questo tipo:

```

{
  "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
  "BatchSize": 500,
  "DocumentDBEventSourceConfig": {
    "CollectionName": "test_collection",
    "DatabaseName": "test_database",
    "FullDocument": "Default"
  },
  "MaximumBatchingWindowInSeconds": 0,
  "EventSourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-epzcyvu4pjjoy",
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
  "LastModified": 1541359182.919,
  "LastProcessingResult": "OK",
  "State": "Updating",
  "StateTransitionReason": "User action"
}

```

```
}
```

Lambda aggiorna le impostazioni in modo asincrono, quindi potresti non vedere queste modifiche nell'output fino al completamento del processo. Per visualizzare le impostazioni correnti della mappatura dell'origine degli eventi, utilizza il comando [get-event-source-mapping](#).

```
aws lambda get-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{
  "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
  "DocumentDBEventSourceConfig": {
    "CollectionName": "test_collection",
    "DatabaseName": "test_database",
    "FullDocument": "Default"
  },
  "BatchSize": 1000,
  "MaximumBatchingWindowInSeconds": 10,
  "EventSourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-epzcyvu4pjoy",
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
  "LastModified": 1541359182.919,
  "LastProcessingResult": "OK",
  "State": "Enabled",
  "StateTransitionReason": "User action"
}
```

Per eliminare la mappatura dell'origine degli eventi di Amazon DocumentDB, utilizza il comando [delete-event-source-mapping](#):

```
aws lambda delete-event-source-mapping \
  --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284
```

## Posizioni di partenza di polling e flussi

Tieni presente che il polling dei flussi durante la creazione e gli aggiornamenti dello strumento di mappatura dell'origine degli eventi alla fine è coerente.

- Durante la creazione dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.

- Durante gli aggiornamenti dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.

Questo comportamento implica che se specifichi LATEST come posizione iniziale del flusso, lo strumento di mappatura dell'origine degli eventi potrebbe perdere eventi durante la creazione o gli aggiornamenti. Per non perdere alcun evento, specifica la posizione iniziale del flusso come TRIM\_HORIZON o AT\_TIMESTAMP.

## Monitoraggio dell'origine degli eventi di Amazon DocumentDB

Per aiutarti a monitorare l'origine degli eventi Amazon DocumentDB, Lambda emette il parametro `IteratorAge` quando la funzione termina l'elaborazione di un batch di record. L'età dell'iteratore è la differenza tra il timestamp dell'evento più recente e il timestamp corrente. Sostanzialmente, il parametro `IteratorAge` indica l'età dell'ultimo record elaborato nel batch.

Se la funzione continua a elaborare nuovi eventi, puoi utilizzare la cronologia di iterazione per stimare la latenza tra il momento in cui un record viene aggiunto e il momento in cui viene elaborato dalla tua funzione. Una tendenza in aumento in `IteratorAge` può indicare problemi con la funzione. Per ulteriori informazioni, consulta [Utilizzo delle CloudWatch metriche con Lambda](#).

I flussi di modifiche di Amazon DocumentDB non sono ottimizzati per gestire ampi intervalli di tempo tra gli eventi. Se la tua sorgente di eventi Amazon DocumentDB non riceve alcun evento per un lungo periodo di tempo, Lambda potrebbe disabilitare la mappatura della sorgente degli eventi. La durata di questo periodo di tempo può variare da alcune settimane a qualche mese a seconda delle dimensioni del cluster e di altri carichi di lavoro.

Lambda supporta payload di dimensioni massime di 6 MB. Tuttavia, gli eventi del flusso di modifica di Amazon DocumentDB possono avere dimensioni fino a 16 MB. Se il flusso di modifica tenta di inviare a Lambda un evento di flusso di modifica di dimensioni superiori a 6 MB, Lambda trasmette il messaggio ed emette il parametro `OversizedRecordCount`. Lambda emette tutti i parametri sulla base del miglior tentativo.

## Tutorial: Utilizzo AWS Lambda con Amazon DocumentDB Streams

In questo tutorial creerai una funzione Lambda di base che utilizza gli eventi provenienti da un flusso di modifica Amazon DocumentDB (compatibile con MongoDB). Per completare questo tutorial, segui le seguenti fasi:

- Configura il tuo cluster Amazon DocumentDB, connettiti a esso e attiva i flussi di modifica su di esso.
- Crea la funzione Lambda e configura il cluster Amazon DocumentDB come origine degli eventi Amazon DocumentDB come origine degli eventi per la funzione.
- Verifica la configurazione inserendo elementi nel tuo database Amazon DocumentDB.

## Creare il cluster Amazon DocumentDB

1. Apri la [console Amazon DocumentDB](#). In Cluster, scegli Crea.
2. Crea un cluster con la seguente configurazione:
  - Per il tipo di cluster, scegli Instance-based cluster. Questa è l'opzione predefinita.
  - In Configurazione del cluster, assicurati che sia selezionata la versione 5.0.0 del motore. Questa è l'opzione predefinita.
  - In Configurazione dell'istanza:
    - Per la classe di istanza DB, seleziona Classi ottimizzate per la memoria. Questa è l'opzione predefinita.
    - Per Numero di istanze di replica regolari, scegli 1.
    - Per la classe Instance, utilizzate la selezione predefinita.
  - In Autenticazione, inserisci un nome utente per l'utente principale, quindi scegli Gestito autonomamente. Inserisci una password, quindi confermala.
  - Mantieni tutte le altre impostazioni predefinite.
3. Scegli Crea cluster.

## Creazione del segreto di Gestione dei segreti

Mentre Amazon DocumentDB crea il tuo cluster, crea un AWS Secrets Manager segreto per archiviare le credenziali del tuo database. Fornirai questo segreto quando creerai la mappatura della sorgente degli eventi Lambda in una fase successiva.

## Creazione del segreto in Gestione dei segreti

1. Apri la console [Gestione dei segreti](#) e scegli Archivia un nuovo segreto.
2. In Scegli il tipo di segreto, scegli una delle seguenti opzioni:
  - In Dettagli di base:

- Tipo di segreto: credenziali per il database Amazon DocumentDB
  - In Credenziali, inserisci lo stesso nome utente e password che hai usato per creare il tuo cluster Amazon DocumentDB.
  - Database: scegli il tuo cluster Amazon DocumentDB.
  - Scegli Next (Successivo).
3. Per Configura segreto, scegli tra le seguenti opzioni:
    - Nome segreto: DocumentDBSecret
    - Scegli Next (Successivo).
  4. Scegli Next (Successivo).
  5. Scegli Store.
  6. Aggiorna la console per verificare di aver archiviato correttamente il segreto DocumentDBSecret.

Nota l'ARN segreto. Ne avrai bisogno in una fase successiva.

## Connessione al cluster

Connettiti al tuo cluster Amazon DocumentDB utilizzando AWS CloudShell

1. Nella console di gestione Amazon DocumentDB, in Clusters, individua il cluster che hai creato. Scegli il tuo cluster selezionando la casella di controllo accanto ad esso.
2. Scegli Connetti al cluster. Viene visualizzata la schermata del comando CloudShell Esegui.
3. Nel campo Nome nuovo ambiente, inserisci un nome univoco, ad esempio «test» e scegli Crea ed esegui.
4. Specifica la password, quando richiesto. Quando viene visualizzato il prompt `primary` `<env-name>`, la connessione al cluster Amazon DocumentDB è stata completata.

## Attivazione dei flussi di modifica

In questo tutorial, monitorerai le modifiche alla `products` raccolta del `docdbdemo` database nel tuo cluster Amazon DocumentDB. Puoi farlo attivando i [flussi di modifica](#).

## Creazione di un nuovo database all'interno del cluster

1. Esegui il comando seguente per creare un nuovo database chiamato `docdbdemo`:

```
use docdbdemo
```

2. Nella finestra del terminale, usa il seguente comando per inserire un record indocdbdemo:

```
db.products.insertOne({"hello":"world"})
```

Dovresti vedere un output come questo:

```
{
  acknowledged: true,
  insertedId: ObjectId('67f85066ca526410fd531d59')
}
```

3. Quindi, attiva i flussi di modifica sulla raccolta `products` del database `docdbdemo` utilizzando il comando seguente:

```
db.adminCommand({modifyChangeStreams: 1,
  database: "docdbdemo",
  collection: "products",
  enable: true});
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{ "ok" : 1, "operationTime" : Timestamp(1680126165, 1) }
```

## Creazione di endpoint VPC dell'interfaccia

Successivamente, crea gli [endpoint VPC di interfaccia](#) per garantire che Lambda e Gestione dei segreti (utilizzati in seguito per archiviare le credenziali di accesso al cluster) possano connettersi al tuo VPC predefinito.

## Creazione di endpoint VPC dell'interfaccia

1. Apri la [console VPC](#). Nel menu a sinistra, in Cloud privato virtuale, scegli Endpoint.
2. Seleziona Crea endpoint. Crea un endpoint con la seguente configurazione:
  - Per Nome tag, inserisci `lambda-default-vpc`.
  - Per la categoria Servizi, scegli AWS servizi.



- Per Servizi, digita `lambda` nella casella di ricerca. Scegli il servizio con il formato `com.amazonaws.<region>.lambda`.
  - Per VPC, scegli il VPC in cui si trova il tuo cluster Amazon DocumentDB. Questo è in genere il VPC predefinito.
  - Per Sottoreti, seleziona le caselle accanto a ciascuna zona di disponibilità. Scegli l'ID della sottorete corretta per ogni zona di disponibilità.
  - Per Tipo di indirizzo IP, seleziona IPv4.
  - Per i gruppi di sicurezza, scegli il gruppo di sicurezza utilizzato dal tuo cluster Amazon DocumentDB. Questo è in genere il gruppo `default` di sicurezza.
  - Mantieni tutte le altre impostazioni predefinite.
  - Seleziona Crea endpoint.
3. Seleziona di nuovo Crea endpoint. Crea un endpoint con la seguente configurazione:
- Per Nome tag, inserisci `secretsmanager-default-vpc`.
  - Per la categoria Servizi, scegli AWS servizi.
  - Per Servizi, digita `secretsmanager` nella casella di ricerca. Scegli il servizio con il formato `com.amazonaws.<region>.secretsmanager`.
  - Per VPC, scegli il VPC in cui si trova il tuo cluster Amazon DocumentDB. Questo è in genere il VPC predefinito.
  - Per Sottoreti, seleziona le caselle accanto a ciascuna zona di disponibilità. Scegli l'ID della sottorete corretta per ogni zona di disponibilità.
  - Per Tipo di indirizzo IP, seleziona IPv4.
  - Per i gruppi di sicurezza, scegli il gruppo di sicurezza utilizzato dal tuo cluster Amazon DocumentDB. Questo è in genere il gruppo `default` di sicurezza.
  - Mantieni tutte le altre impostazioni predefinite.
  - Seleziona Crea endpoint.

Questo completa la sezione del tutorial dedicata alla configurazione del cluster.

## Creazione del ruolo di esecuzione

Nella prossima serie di passaggi, creerai la tua funzione Lambda. Innanzitutto, devi creare il ruolo di esecuzione che fornisce alla funzione l'autorizzazione per accedere al cluster. Per farlo, creerai prima una policy IAM, dopodiché la collegherai a un ruolo IAM.

## Creazione di una policy IAM

1. Nella console IAM, apri la pagina [Policy](#), quindi scegli Crea policy.
2. Scegli la scheda JSON. Nella policy seguente, sostituisci l'ARN della risorsa Gestione dei segreti nell'ultima riga dell'istruzione con l'ARN del segreto utilizzato in precedenza e copia la policy nell'editor.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "LambdaESMNetworkingAccess",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2>DeleteNetworkInterface",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "kms:Decrypt"
      ],
      "Resource": "*"
    },
    {
      "Sid": "LambdaDocDBESMAccess",
      "Effect": "Allow",
      "Action": [
        "rds:DescribeDBClusters",
        "rds:DescribeDBClusterParameters",
        "rds:DescribeDBSubnetGroups"
      ],
      "Resource": "*"
    },
    {
      "Sid": "LambdaDocDBESMGetSecretValueAccess",
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],

```

```
"Resource": "arn:aws:secretsmanager:us-east-1:123456789012:secret:DocumentDBSecret"
    }
  ]
}
```

3. Scegli Successivo: Tag, quindi Successivo: Verifica.
4. In Nome, inserisci `AWSDocumentDBLambdaPolicy`.
5. Scegli Create Policy (Crea policy).

### Per creare il ruolo IAM

1. Nella console IAM, apri la pagina [Ruoli](#), quindi scegli Crea ruolo.
2. Per Seleziona un'entità attendibile, scegli le seguenti opzioni:
  - Tipo di entità attendibile: servizio AWS
  - Servizio o caso d'uso: Lambda
  - Scegli Next (Successivo).
3. Per Aggiungi autorizzazioni, scegli la `AWSDocumentDBLambdaPolicy` policy che hai appena creato e quella `AWSLambdaBasicExecutionRole` per assegnare alla tua funzione i permessi di scrittura su Amazon Logs. CloudWatch
4. Scegli Next (Successivo).
5. Per Nome ruolo, inserisci `AWSDocumentDBLambdaExecutionRole`.
6. Scegli Crea ruolo.

### Creazione della funzione Lambda

Questo tutorial utilizza il runtime Python 3.14, ma abbiamo anche fornito file di codice di esempio per altri runtime. Puoi selezionare la scheda nella casella seguente per visualizzare il codice per il runtime che ti interessa.

Il codice riceve l'input di un evento Amazon DocumentDB ed elabora il messaggio in esso contenuto.

### Creazione della funzione Lambda

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli Crea funzione.

3. Scegli Crea da zero.
4. In Basic information (Informazioni di base) eseguire queste operazioni:
  - a. Nel campo Nome funzione, inserisci `ProcessDocumentDBRecords`.
  - b. Per Runtime, scegli Python 3.14.
  - c. In Architecture (Architettura), scegli `x86_64`.
5. Nella scheda Modifica ruolo di esecuzione predefinito, effettua le seguenti operazioni:
  - a. Espandi la scheda, quindi scegli Utilizza un ruolo esistente.
  - b. Seleziona il `AWSDocumentDBLambdaExecutionRole` che hai creato in precedenza.
6. Scegli Crea funzione.

## Implementazione del codice della funzione

1. Scegli la scheda Python nella casella seguente e copia il codice.

.NET

SDK per .NET

### Note

C'è altro su GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite .NET.

```
using Amazon.Lambda.Core;
using System.Text.Json;
using System;
using System.Collections.Generic;
using System.Text.Json.Serialization;
//Assembly attribute to enable the Lambda function's JSON input to be
//converted into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace LambdaDocDb;
```

```
public class Function
{
    /// <summary>
    /// Lambda function entry point to process Amazon DocumentDB events.
    /// </summary>
    /// <param name="event">The Amazon DocumentDB event.</param>
    /// <param name="context">The Lambda context object.</param>
    /// <returns>A string to indicate successful processing.</returns>
    public string FunctionHandler(Event evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Events)
        {
            ProcessDocumentDBEvent(record, context);
        }

        return "OK";
    }

    private void ProcessDocumentDBEvent(DocumentDBEventRecord record,
    ILambdaContext context)
    {
        var eventData = record.Event;
        var operationType = eventData.OperationType;
        var databaseName = eventData.Ns.Db;
        var collectionName = eventData.Ns.Coll;
        var fullDocument = JsonSerializer.Serialize(eventData.FullDocument,
        new JsonSerializerOptions { WriteIndented = true });

        context.Logger.LogLine($"Operation type: {operationType}");
        context.Logger.LogLine($"Database: {databaseName}");
        context.Logger.LogLine($"Collection: {collectionName}");
        context.Logger.LogLine($"Full document:\n{fullDocument}");
    }

    public class Event
    {
        [JsonPropertyName("eventSourceArn")]
        public string EventSourceArn { get; set; }
    }
}
```

```
[JsonPropertyName("events")]
public List<DocumentDBEventRecord> Events { get; set; }

[JsonPropertyName("eventSource")]
public string EventSource { get; set; }
}

public class DocumentDBEventRecord
{
    [JsonPropertyName("event")]
    public EventData Event { get; set; }
}

public class EventData
{
    [JsonPropertyName("_id")]
    public IdData Id { get; set; }

    [JsonPropertyName("clusterTime")]
    public ClusterTime ClusterTime { get; set; }

    [JsonPropertyName("documentKey")]
    public DocumentKey DocumentKey { get; set; }

    [JsonPropertyName("fullDocument")]
    public Dictionary<string, object> FullDocument { get; set; }

    [JsonPropertyName("ns")]
    public Namespace Ns { get; set; }

    [JsonPropertyName("operationType")]
    public string OperationType { get; set; }
}

public class IdData
{
    [JsonPropertyName("_data")]
    public string Data { get; set; }
}

public class ClusterTime
{
    [JsonPropertyName("$timestamp")]
```

```
        public Timestamp Timestamp { get; set; }
    }

    public class Timestamp
    {
        [JsonPropertyName("t")]
        public long T { get; set; }

        [JsonPropertyName("i")]
        public int I { get; set; }
    }

    public class DocumentKey
    {
        [JsonPropertyName("_id")]
        public Id Id { get; set; }
    }

    public class Id
    {
        [JsonPropertyName("$oid")]
        public string Oid { get; set; }
    }

    public class Namespace
    {
        [JsonPropertyName("db")]
        public string Db { get; set; }

        [JsonPropertyName("coll")]
        public string Coll { get; set; }
    }
}
```

## Go

### SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite Go.

```
package main

import (
    "context"
    "encoding/json"
    "fmt"

    "github.com/aws/aws-lambda-go/lambda"
)

type Event struct {
    Events []Record `json:"events"`
}

type Record struct {
    Event struct {
        OperationType string `json:"operationType"`
        NS            struct {
            DB    string `json:"db"`
            Coll string `json:"coll"`
        } `json:"ns"`
        FullDocument interface{} `json:"fullDocument"`
    } `json:"event"`
}

func main() {
    lambda.Start(handler)
}
```



```
func handler(ctx context.Context, event Event) (string, error) {
    fmt.Println("Loading function")
    for _, record := range event.Events {
        logDocumentDBEvent(record)
    }

    return "OK", nil
}

func logDocumentDBEvent(record Record) {
    fmt.Printf("Operation type: %s\n", record.Event.OperationType)
    fmt.Printf("db: %s\n", record.Event.NS.DB)
    fmt.Printf("collection: %s\n", record.Event.NS.Coll)
    docBytes, _ := json.MarshalIndent(record.Event.FullDocument, "", " ")
    fmt.Printf("Full document: %s\n", string(docBytes))
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite Java.

```
import java.util.List;
import java.util.Map;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class Example implements RequestHandler<Map<String, Object>, String> {

    @SuppressWarnings("unchecked")
    @Override
```

```
public String handleRequest(Map<String, Object> event, Context context) {
    List<Map<String, Object>> events = (List<Map<String, Object>>)
event.get("events");
    for (Map<String, Object> record : events) {
        Map<String, Object> eventData = (Map<String, Object>)
record.get("event");
        processEventData(eventData);
    }

    return "OK";
}

@SuppressWarnings("unchecked")
private void processEventData(Map<String, Object> eventData) {
    String operationType = (String) eventData.get("operationType");
    System.out.println("operationType: %s".formatted(operationType));

    Map<String, Object> ns = (Map<String, Object>) eventData.get("ns");

    String db = (String) ns.get("db");
    System.out.println("db: %s".formatted(db));
    String coll = (String) ns.get("coll");
    System.out.println("coll: %s".formatted(coll));

    Map<String, Object> fullDocument = (Map<String, Object>)
eventData.get("fullDocument");
    System.out.println("fullDocument: %s".formatted(fullDocument));
}
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Consumare un evento Amazon DocumentDB utilizzando Lambda. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
    event.events.forEach(record => {
        logDocumentDBEvent(record);
    });
    return 'OK';
};

const logDocumentDBEvent = (record) => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
    console.log('collection: ' + record.event.ns.coll);
    console.log('Full document:', JSON.stringify(record.event.fullDocument,
        null, 2));
};
```

## Consumare un evento Amazon DocumentDB con Lambda utilizzando TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from
    'aws-lambda';

console.log('Loading function');

export const handler = async (
    event: DocumentDBEventSubscriptionContext,
    context: any
): Promise<string> => {
    event.events.forEach((record: DocumentDBEventRecord) => {
        logDocumentDBEvent(record);
    });
    return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
    console.log('collection: ' + record.event.ns.coll);
```

```
console.log('Full document:', JSON.stringify(record.event.fullDocument,  
null, 2));  
};
```

## PHP

### SDK per PHP

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite PHP.

```
<?php  
  
require __DIR__.'./vendor/autoload.php';  
  
use Bref\Context\Context;  
use Bref\Event\Handler;  
  
class DocumentDBEventHandler implements Handler  
{  
    public function handle($event, Context $context): string  
    {  
        $events = $event['events'] ?? [];  
        foreach ($events as $record) {  
            $this->logDocumentDBEvent($record['event']);  
        }  
        return 'OK';  
    }  
  
    private function logDocumentDBEvent($event): void  
    {  
        // Extract information from the event record  
  
        $operationType = $event['operationType'] ?? 'Unknown';
```

```

$db = $event['ns']['db'] ?? 'Unknown';
$collection = $event['ns']['coll'] ?? 'Unknown';
$fullDocument = $event['fullDocument'] ?? [];

// Log the event details

echo "Operation type: $operationType\n";
echo "Database: $db\n";
echo "Collection: $collection\n";
echo "Full document: " . json_encode($fullDocument,
JSON_PRETTY_PRINT) . "\n";
}
}
return new DocumentDBEventHandler();

```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite Python.

```

import json

def lambda_handler(event, context):
    for record in event.get('events', []):
        log_document_db_event(record)
    return 'OK'

def log_document_db_event(record):
    event_data = record.get('event', {})
    operation_type = event_data.get('operationType', 'Unknown')
    db = event_data.get('ns', {}).get('db', 'Unknown')
    collection = event_data.get('ns', {}).get('coll', 'Unknown')
    full_document = event_data.get('fullDocument', {})

```

```
print(f"Operation type: {operation_type}")
print(f"db: {db}")
print(f"collection: {collection}")
print("Full document:", json.dumps(full_document, indent=2))
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite Ruby.

```
require 'json'

def lambda_handler(event:, context:)
  event['events'].each do |record|
    log_document_db_event(record)
  end
  'OK'
end

def log_document_db_event(record)
  event_data = record['event'] || {}
  operation_type = event_data['operationType'] || 'Unknown'
  db = event_data.dig('ns', 'db') || 'Unknown'
  collection = event_data.dig('ns', 'coll') || 'Unknown'
  full_document = event_data['fullDocument'] || {}

  puts "Operation type: #{operation_type}"
  puts "db: #{db}"
  puts "collection: #{collection}"
  puts "Full document: #{JSON.pretty_generate(full_document)}"
end
```

## Rust

### SDK per Rust

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite Ruby.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::documentdb::{DocumentDbEvent, DocumentDbInnerEvent},
};

// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features
    = ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<DocumentDbEvent>) ->Result<(),
    Error> {

    tracing::info!("Event Source ARN: {:?}", event.payload.event_source_arn);
    tracing::info!("Event Source: {:?}", event.payload.event_source);

    let records = &event.payload.events;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }
}
```

```
        for record in records{
            log_document_db_event(record);
        }

        tracing::info!("Document db records processed");

        // Prepare the response
        Ok(())
    }

    fn log_document_db_event(record: &DocumentDbInnerEvent)-> Result<(), Error>{
        tracing::info!("Change Event: {:?}", record.event);

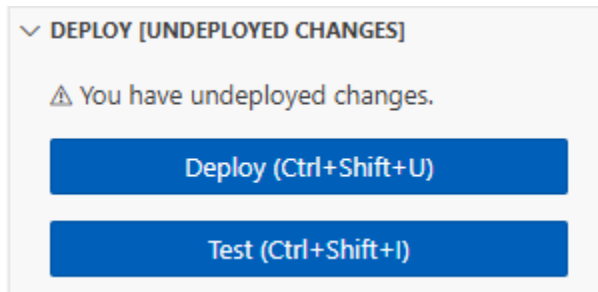
        Ok(())
    }

    #[tokio::main]
    async fn main() -> Result<(), Error> {
        tracing_subscriber::fmt()
            .with_max_level(tracing::Level::INFO)
            .with_target(false)
            .without_time()
            .init();

        let func = service_fn(function_handler);
        lambda_runtime::run(func).await?;
        Ok(())
    }
}
```

2. Nel riquadro Codice sorgente della console Lambda, incolla il codice nell'editor di codice, sostituendo il codice creato da Lambda.
3. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:





## Creazione della mappatura dell'origine degli eventi Lambda

Crea lo strumento di mappatura dell'origine degli eventi che associa il flusso di modifica di Amazon DocumentDB alla funzione Lambda. Dopo aver creato questa mappatura della sorgente degli eventi, inizia AWS Lambda immediatamente il polling dello stream.

### Creazione di una mappatura dell'origine degli eventi

1. Apri la [pagina Funzioni](#) della console Lambda.
2. Scegli la funzione `ProcessDocumentDBRecords` creata in precedenza.
3. Scegli la scheda Configurazione, quindi scegli `Trigger` nel menu a sinistra.
4. Selezionare `Add trigger` (Aggiungi trigger).
5. In Configurazione del trigger, per l'origine seleziona Amazon DocumentDB.
6. Crea la mappatura dell'origine degli eventi con la seguente configurazione:
  - Cluster Amazon DocumentDB: scegli il cluster che hai creato in precedenza.
  - Nome del database: `docdbdemo`
  - Nome della collezione: `prodotti`
  - Dimensione del lotto: `1`
  - Posizione di partenza: `Ultima`
  - Autenticazione: `BASIC_AUTH`
  - Chiave Secrets Manager: scegli il segreto per il tuo cluster Amazon DocumentDB. Si chiama qualcosa del genere `!cluster-12345678-a6f0-52c0-b290-db4aga89274f`.
  - Finestra batch: `1`
  - Configurazione completa del documento: `UpdateLookup`
7. Scegliere `Aggiungi`. La creazione della mappatura dell'origine degli eventi può richiedere alcuni minuti.

## Test della funzione

Attendi che la mappatura della sorgente dell'evento raggiunga lo stato **Abilitato**. Questo processo può richiedere diversi minuti. Quindi, verifica la configurazione end-to-end inserendo, aggiornando ed eliminando i record del database. Prima di iniziare:

1. [Riconnettiti al cluster Amazon DocumentDB nel tuo ambiente.](#) CloudShell
2. Esegui il comando seguente per assicurarti di utilizzare il docdbdemo database:

```
use docdbdemo
```

### Inserire un record

Inserisci un record nella raccolta `products` del database `docdbdemo`:

```
db.products.insertOne({"name":"Pencil", "price": 1.00})
```

Verifica che la tua funzione abbia elaborato correttamente questo evento [controllando CloudWatch i log](#). Dovresti vedere una voce di registro come questa:

| ▶ | Timestamp                | Message                             |
|---|--------------------------|-------------------------------------|
| ▶ | 2025-05-05T23:55:23.474Z | Operation type: insert              |
| ▶ | 2025-05-05T23:55:23.474Z | db: docdbdemo                       |
| ▶ | 2025-05-05T23:55:23.474Z | collection: products                |
| ▶ | 2025-05-05T23:55:23.474Z | Full document: {                    |
| ▶ | 2025-05-05T23:55:23.474Z | "_id": {                            |
| ▶ | 2025-05-05T23:55:23.474Z | "\$oid": "68194fea08d12462ea531d58" |
| ▶ | 2025-05-05T23:55:23.474Z | },                                  |
| ▶ | 2025-05-05T23:55:23.474Z | "name": "Pencil",                   |
| ▶ | 2025-05-05T23:55:23.474Z | "price": 1                          |
| ▶ | 2025-05-05T23:55:23.474Z | }                                   |

## Aggiornare un record

Aggiorna il record che hai appena inserito con il seguente comando:

```
db.products.updateOne(  
  { "name": "Pencil" },  
  { $set: { "price": 0.50 } }  
)
```

Verifica che la tua funzione abbia elaborato correttamente questo evento [controllando CloudWatch i log](#). Dovresti vedere una voce di registro come questa:

| ▶ | Timestamp                | Message                             |
|---|--------------------------|-------------------------------------|
| ▶ | 2025-05-05T23:55:27.918Z | Operation type: update              |
| ▶ | 2025-05-05T23:55:27.918Z | db: docdbdemo                       |
| ▶ | 2025-05-05T23:55:27.918Z | collection: products                |
| ▶ | 2025-05-05T23:55:27.918Z | Full document: {                    |
| ▶ | 2025-05-05T23:55:27.918Z | "_id": {                            |
| ▶ | 2025-05-05T23:55:27.918Z | "\$oid": "68194fea08d12462ea531d58" |
| ▶ | 2025-05-05T23:55:27.918Z | },                                  |
| ▶ | 2025-05-05T23:55:27.918Z | "name": "Pencil",                   |
| ▶ | 2025-05-05T23:55:27.918Z | "price": 0.5                        |
| ▶ | 2025-05-05T23:55:27.918Z | }                                   |

## Eliminazione di un registro

Elimina il record che hai appena aggiornato con il seguente comando:

```
db.products.deleteOne( { "name": "Pencil" } )
```

Verifica che la tua funzione abbia elaborato correttamente questo evento [controllando CloudWatch i log](#). Dovresti vedere una voce di registro come questa:

| ▶ | Timestamp                | Message                |
|---|--------------------------|------------------------|
| ▶ | 2025-05-05T23:55:32.362Z | Operation type: delete |
| ▶ | 2025-05-05T23:55:32.362Z | db: docdbdemo          |
| ▶ | 2025-05-05T23:55:32.362Z | collection: products   |
| ▶ | 2025-05-05T23:55:32.362Z | Full document: {}      |

## Risoluzione dei problemi

Se non vedi alcun evento del database nei CloudWatch log della tua funzione, controlla quanto segue:

- Assicurati che la mappatura della sorgente degli eventi Lambda (nota anche come trigger) sia nello stato **Abilitato**. La creazione delle mappature delle sorgenti degli eventi può richiedere diversi minuti.
- Se la mappatura dell'origine degli eventi è **abilitata** ma non vedi ancora gli eventi del database in: CloudWatch
- Assicurati che il nome del **database** nella mappatura dell'origine degli eventi sia impostato su: **docdbdemo**

☐ | **Trigger**



**Amazon DocumentDB:** [arn:aws:rds:us-east-2:██████████:cluster:docdb-2025-██████████](#)

arn:aws:rds:us-east-2:██████████:cluster:docdb-2025-██████████

state: **Enabled**

### ▼ Details

Activate trigger: **Yes**

Authentication: **BASIC\_AUTH**

Batch size: **1**

Batch window: **1**

Collection name: **products**

Database name: **docdbdemo**



- Controlla il campo del risultato dell'ultima elaborazione della mappatura della sorgente dell'evento per il seguente messaggio «PROBLEMA: errore di connessione. Il tuo VPC deve essere in grado di connettersi a Lambda e STS, nonché a Secrets Manager se è richiesta l'autenticazione». Se visualizzi questo errore, assicurati di aver [creato gli endpoint di interfaccia](#)

[VPC Lambda e Secrets Manager e che gli endpoint](#) utilizzino lo stesso VPC e le stesse sottoreti utilizzati dal tuo cluster Amazon DocumentDB.

#### Trigger



Amazon DocumentDB: [arn:aws:rds:us-west-2:██████████:cluster:docdb-2025-██████████](#)

arn:aws:rds:us-west-2:██████████:cluster:docdb-2025-██████████

state: **Enabled**

#### ▼ Details

Activate trigger: **Yes**

Authentication: **BASIC\_AUTH**

Batch size: **1**

Batch window: **1**

Collection name: **products**

Database name: **docdbdemo**

Event source mapping ARN: [arn:aws:lambda:us-west-2:██████████:event-source-mapping:██████████-██████████](#)

Full document configuration: **UpdateLookup**

Last processing result: **PROBLEM: Connection error. Your VPC must be able to connect to Lambda and STS, as well as Secrets Manager if authentication is required. You can provide access by configuring PrivateLink or a NAT Gateway.**

On-failure destination: **None**

## Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando le risorse AWS che non si utilizzano più, è possibile evitare addebiti superflui sul proprio account Account AWS.

Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **confirm** nel campo di immissione del testo, quindi scegli Elimina.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina [Ruoli](#) della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Elimina.

## Eliminazione degli endpoint VPC

1. Apri la [console VPC](#). Nel menu a sinistra, in Cloud privato virtuale, scegli Endpoint.
2. Seleziona gli endpoint creati.
3. Seleziona Actions (Operazioni), Delete VPC endpoints (Eliminazione di endpoint VPC).
4. Inserisci **delete** nel campo di immissione del testo.
5. Scegli Elimina.

## Eliminazione del cluster Amazon DocumentDB

1. Apri la [console Amazon DocumentDB](#).
2. Scegli il cluster Amazon DocumentDB che hai creato per questo tutorial e disabilita la protezione dall'eliminazione.
3. Nella pagina principale Cluster, scegli nuovamente il tuo cluster Amazon DocumentDB.
4. Scegli Operazioni > Elimina.
5. Per Crea snapshot finale del cluster, seleziona No.
6. Inserisci **delete** nel campo di immissione del testo.
7. Scegli Elimina.

## Eliminazione del segreto in Gestione dei segreti

1. Apri la [console Secrets Manager](#).
2. Scegli il segreto creato per questo tutorial.
3. Scegli Operazioni, Elimina segreto.
4. Scegliere Schedule deletion (Pianifica eliminazione).

# Utilizzo AWS Lambda con Amazon DynamoDB

## Note

Se desideri inviare dati a una destinazione diversa da una funzione Lambda o arricchire i dati prima di inviarli, consulta Amazon Pipes. [EventBridge](#)

Puoi utilizzare una AWS Lambda funzione per elaborare i record in uno stream [Amazon DynamoDB](#). Con DynamoDB Streams, è possibile attivare una funzione Lambda per eseguire lavoro aggiuntivo ogni volta che una tabella DynamoDB viene aggiornata.

Durante l'elaborazione dei flussi DynamoDB, è necessario implementare una logica di risposta batch parziale per evitare che i record elaborati correttamente vengano ritentati quando alcuni record di un batch falliscono. L'utilità [Batch Processor](#) di Powertools for AWS Lambda è disponibile in Python TypeScript, .NET e Java e semplifica questa implementazione gestendo automaticamente la logica di risposta batch parziale, riducendo i tempi di sviluppo e migliorando l'affidabilità.

## Argomenti

- [Flussi di polling e batching](#)
- [Posizioni di partenza di polling e flussi](#)
- [Lettori simultanei di uno shard in DynamoDB Streams](#)
- [Esempio di evento](#)
- [Elaborare i record DynamoDB con Lambda](#)
- [Configurazione della risposta batch parziale con DynamoDB e Lambda](#)
- [Conservare i record scartati per un'origine eventi DynamoDB in Lambda](#)
- [Implementazione dell'elaborazione del flusso DynamoDB stateful in Lambda](#)
- [Parametri Lambda per gli strumenti di mappatura dell'origine degli eventi di Amazon DynamoDB](#)
- [Utilizzo del filtro eventi con un'origine eventi DynamoDB](#)
- [Tutorial: Utilizzo AWS Lambda con stream Amazon DynamoDB](#)

## Flussi di polling e batching

Lambda esegue il polling delle partizioni presenti nel proprio flusso DynamoDB ricercando i record a una velocità di base di 4 volte al secondo. Quando sono disponibili dei record, Lambda invoca la



funzione e attende il risultato. Se l'elaborazione ha esito positivo, Lambda riprende il polling fino a quando non riceve più record.

Per impostazione predefinita, Lambda richiama la funzione non appena i record sono disponibili. Se il batch che Lambda legge dall'origine eventi contiene un solo record, Lambda invia solo un record alla funzione. Per evitare di richiamare la funzione con pochi record è possibile, configurando un periodo di batch, chiedere all'origine eventi di memorizzare nel buffer i registri per un massimo di 5 minuti. Prima di richiamare la funzione, Lambda continua a leggere i registri dall'origine eventi fino a quando non ha raccolto un batch completo, fino alla scadenza del periodo di batch o fino a quando il batch non ha raggiunto il limite del payload di 6 MB. Per ulteriori informazioni, consulta [Comportamento di batching](#).

 Warning

Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per saperne di più, consulta [How do I make my Lambda function idempotent](#) nel Knowledge Center. AWS

Lambda non attende il completamento di alcuna [estensione](#) prima di inviare il batch successivo per l'elaborazione. In altre parole, le estensioni possono continuare a funzionare mentre Lambda elabora il successivo batch di record. Ciò può causare problemi di limitazione in caso di violazione delle impostazioni o dei limiti di [simultaneità](#). Per rilevare se si tratta di un potenziale problema, monitora le tue funzioni e verifica se i [parametri di simultaneità](#) per lo strumento di mappatura dell'origine degli eventi sono superiori al previsto. A causa degli intervalli ridotti tra le invocazioni, Lambda potrebbe segnalare brevemente un utilizzo della simultaneità maggiore rispetto al numero di partizioni. Ciò può essere vero anche per le funzioni Lambda senza estensioni.

Configura l' [ParallelizationFactor](#) impostazione per elaborare uno shard di uno stream DynamoDB con più di una chiamata Lambda contemporaneamente. Puoi specificare il numero di batch simultanei che Lambda esegue il polling da uno shard utilizzando un fattore di parallelizzazione compreso tra 1 (predefinito) e 10. Ad esempio, se `ParallelizationFactor` imposti su 2, puoi avere un massimo di 200 chiamate Lambda simultanee per elaborare 100 stream shard DynamoDB (anche se in pratica potresti vedere valori diversi per la metrica). `ConcurrentExecutions` Ciò permette di dimensionare verso l'alto il throughput di elaborazione quando il volume dei dati è volatile e l' [IteratorAge](#) è alta.

Se si aumenta il numero di batch simultanei per shard, Lambda garantisce comunque l'ordine di elaborazione a livello di elemento (partizione e chiave di ordinamento).

## Posizioni di partenza di polling e flussi

Tieni presente che il polling dei flussi durante la creazione e gli aggiornamenti dello strumento di mappatura dell'origine degli eventi alla fine è coerente.

- Durante la creazione dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.
- Durante gli aggiornamenti dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.

Questo comportamento implica che se specifichi LATEST come posizione iniziale del flusso, lo strumento di mappatura dell'origine degli eventi potrebbe perdere eventi durante la creazione o gli aggiornamenti. Per garantire che nessun evento venga perso, specifica la posizione di inizio del flusso come TRIM\_HORIZON.

## Lettori simultanei di uno shard in DynamoDB Streams

Per le tabelle di una regione singola che non sono tabelle globali, è possibile progettare contemporaneamente fino a due funzioni Lambda per leggere dallo stesso shard di DynamoDB Streams nello stesso momento. Il superamento di questo limite comporta una limitazione delle richieste. Per le tabelle globali consigliamo di limitare il numero di lettori simultanei a uno per evitare richieste di limitazione della larghezza di banda della rete.

## Esempio di evento

### Example

```
{
  "Records": [
    {
      "eventID": "1",
      "eventVersion": "1.0",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        }
      }
    }
  ]
}
```

```

    },
    "NewImage": {
      "Message": {
        "S": "New item!"
      },
      "Id": {
        "N": "101"
      }
    },
    "StreamViewType": "NEW_AND_OLD_IMAGES",
    "SequenceNumber": "111",
    "SizeBytes": 26
  },
  "awsRegion": "us-west-2",
  "eventName": "INSERT",
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2024-06-10T19:26:16.525",
  "eventSource": "aws:dynamodb"
},
{
  "eventID": "2",
  "eventVersion": "1.0",
  "dynamodb": {
    "OldImage": {
      "Message": {
        "S": "New item!"
      },
      "Id": {
        "N": "101"
      }
    },
    "SequenceNumber": "222",
    "Keys": {
      "Id": {
        "N": "101"
      }
    }
  },
  "SizeBytes": 59,
  "NewImage": {
    "Message": {
      "S": "This item has changed"
    },
    "Id": {
      "N": "101"
    }
  }
}

```

```
    }  
  },  
  "StreamViewType": "NEW_AND_OLD_IMAGES"  
},  
"awsRegion": "us-west-2",  
"eventName": "MODIFY",  
"eventSourceARN": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/  
stream/2024-06-10T19:26:16.525",  
"eventSource": "aws:dynamodb"  
}  
]}
```

## Elaborare i record DynamoDB con Lambda

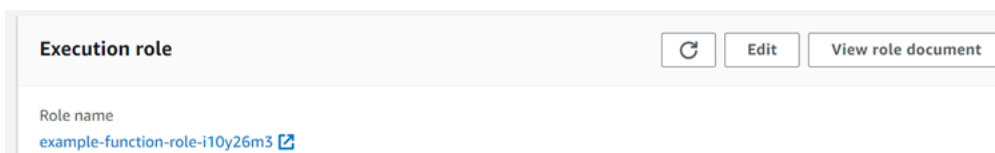
Crea una mappatura dell'origine eventi per indicare a Lambda di inviare i record dal proprio flusso a una funzione Lambda. È possibile creare più mappature dell'origine eventi per elaborare gli stessi dati con più funzioni Lambda o per elaborare elementi da più flussi con una singola funzione.

È possibile configurare gli strumenti di mappatura dell'origine degli eventi per elaborare i record da un flusso in un altro Account AWS. Per ulteriori informazioni, consulta [the section called “Mappature tra account”](#).

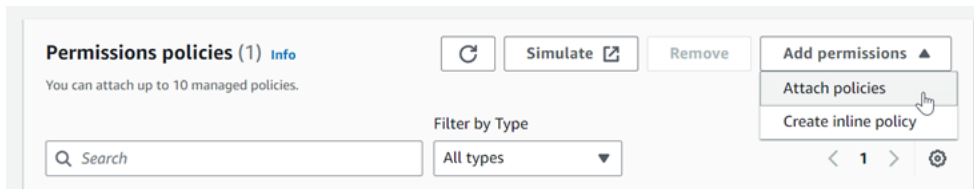
Per configurare la tua funzione per la lettura da DynamoDB Streams, collega la policy gestita di [DBExecutionDynamo Role AWSLambda](#) al tuo AWS ruolo di esecuzione e quindi crea un trigger DynamoDB.

Per aggiungere le autorizzazioni e creare un trigger

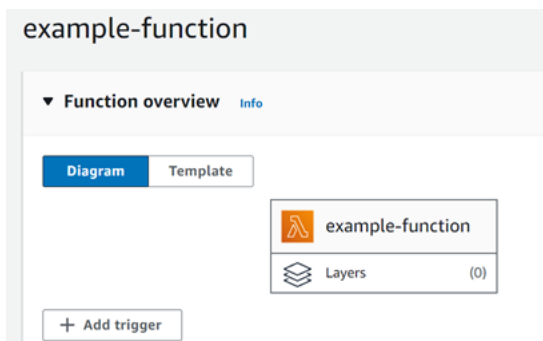
1. Aprire la pagina [Functions](#) (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Quindi, seleziona la scheda Configuration (Configurazione) e poi Permissions (Autorizzazioni).
4. In Nome del ruolo, scegli il link al tuo ruolo di esecuzione. Questo ruolo si apre nella console IAM.



5. Seleziona Aggiungi autorizzazioni, quindi seleziona Collega policy.



6. Inserisci `AWSLambdaDynamoDBExecutionRole` nel campo di ricerca. Aggiungi questa policy al tuo ruolo di esecuzione. Si tratta di una policy AWS gestita che contiene le autorizzazioni che la funzione deve leggere dal flusso DynamoDB. Per ulteriori informazioni su questa politica, consulta [AWSLambdaDynamoDBExecution Role](#) nel Managed Policy Reference.AWS
7. Torna alla funzione nella console Lambda. In Panoramica delle funzioni, scegliere Aggiungi trigger.



8. Scegliere un tipo di trigger.
9. Configurare le opzioni richieste, quindi scegliere Add (Aggiungi).

Lambda supporta le seguenti opzioni per le origini eventi DynamoDB:

#### Opzioni di origine eventi

- **DynamoDB table (Tabella DynamoDB):** la tabella DynamoDB da cui leggere i record.
- **Batch size (Dimensione batch):** il numero di record da inviare alla funzione in ogni batch, fino a 10.000. Lambda passa tutti i record del batch alla funzione in una singola chiamata, purché la dimensione totale degli eventi non superi il [limite di payload](#) per l'invocazione sincrona (6 MB).
- **Batch window (Periodo di batch):** specifica il tempo massimo in secondi per la raccolta dei record prima di richiamare la funzione.
- **Starting position (Posizione iniziale):** elabora solo i nuovi record o tutti i record esistenti.
  - **Latest (Ultimi):** consente di elaborare i nuovi record aggiunti al flusso.
  - **Trim Horizon (Orizzonte di taglio):** elabora tutti i record contenuti nel flusso.

Dopo l'elaborazione di qualsiasi record esistente, la funzione è aggiornata e continua a elaborare nuovi record.

- Destinazione in caso di errore: una coda SQS standard o un argomento SNS standard per i record che non è possibile elaborare. Quando Lambda scarta un batch di record perché è troppo datato o ha esaurito tutti i tentativi, invia i dettagli sul batch alla coda o all'argomento.
- Retry attempts (Nuovi tentativi): il numero massimo di tentativi che Lambda effettua quando la funzione restituisce un errore. Non si applica agli errori di servizio o alle limitazioni in cui il batch non ha raggiunto la funzione.
- Maximum age of record (Età massima del record): l'età massima di un record inviato da Lambda alla funzione.
- Split batch on error (Dividi batch in caso di errore): quando la funzione restituisce un errore, divide il batch in due prima di un nuovo tentativo. L'impostazione originale delle dimensioni del batch rimane invariata.
- Batch simultanei per shard: elabora più batch dallo stesso shard simultaneamente.
- Enabled (Abilitato): impostare su true per abilitare la mappatura dell'origine eventi. Impostare su false per interrompere l'elaborazione dei record. Lambda registra l'ultimo record elaborato e riprende l'elaborazione da quel punto quando la mappatura viene riabilitata.

#### Note

Non ti vengono addebitati costi per le chiamate GetRecords API richiamate da Lambda come parte dei trigger di DynamoDB.

Per gestire la configurazione dell'origine eventi in un momento successivo, scegliere il trigger nel designer.

Creazione di uno strumento di mappatura dell'origine degli eventi multi-account

[Amazon DynamoDB ora supporta policy basate sulle risorse](#). Con questa funzionalità, puoi elaborare i dati da un flusso DynamoDB in un Account AWS con una funzione Lambda in un altro account.

Per creare una mappatura delle sorgenti degli eventi per la tua funzione Lambda utilizzando un flusso DynamoDB in un Account AWS altro, devi configurare il flusso utilizzando una policy basata sulle risorse per concedere alla funzione Lambda il permesso di leggere i record. Per informazioni su come

configurare lo stream per l'accesso tra account diversi, consulta [Condividi l'accesso con le funzioni Lambda tra](#) account nella Amazon DynamoDB Developer Guide.

Dopo aver configurato lo stream con una policy basata sulle risorse che fornisce alla funzione Lambda le autorizzazioni richieste, crea la mappatura delle sorgenti degli eventi con il tuo ARN del flusso tra account. Puoi trovare l'ARN dello stream nella scheda Esportazioni e flussi della tabella nella console DynamoDB per più account.

Quando usi la console Lambda, incolla l'ARN dello stream direttamente nel campo di input della tabella DynamoDB nella pagina di creazione della mappatura delle sorgenti degli eventi.

Nota: i trigger interregionali non sono supportati.

## Configurazione della risposta batch parziale con DynamoDB e Lambda

Prima di configurare le risposte batch parziali, è necessario creare una mappatura dell'origine degli eventi DynamoDB. Per istruzioni, consulta [the section called “Creare lo strumento di mappatura”](#).

Quando si consumano ed elaborano i dati di streaming da un'origine eventi, per impostazione predefinita i Lambda imposta i checkpoint al numero di sequenza più alto di un batch solo quando il batch è riuscito completamente. Lambda tratta tutti gli altri risultati come un fallimento completo e riprova a elaborare il batch fino al limite di tentativi. Per consentire i successi parziali durante l'elaborazione di batch da un flusso, attivare `ReportBatchItemFailures`. Consentire i successi parziali può aiutare a ridurre il numero di tentativi su un record, anche se non previene completamente la possibilità che si verifichino nuovi tentativi in un record andato a buon fine.

Per attivare `ReportBatchItemFailures`, includere il valore `enumReportBatchItemFailures` nell'[FunctionResponseTypes](#) elenco. Questo elenco indica quali tipi di risposta sono abilitati per la funzione. È possibile configurare questo elenco quando si [crea](#) o si [aggiorna](#) uno strumento di mappatura dell'origine degli eventi.

### Note

Anche quando il codice della funzione restituisce risposte parziali di errore in batch, queste risposte non vengono elaborate da Lambda a meno che la `ReportBatchItemFailures` funzionalità non sia esplicitamente attivata per la mappatura dell'origine dell'evento.

## Sintassi di report

Quando si configura la creazione di report sugli errori degli elementi batch, la `StreamsEventResponse` classe viene restituita con un elenco di errori degli articoli batch. È possibile utilizzare un `StreamsEventResponse` oggetto per restituire il numero di sequenza del primo record non riuscito nel batch. È inoltre possibile creare la propria classe personalizzata utilizzando la sintassi di risposta corretta. La seguente struttura JSON mostra la sintassi di risposta richiesta:

```
{
  "batchItemFailures": [
    {
      "itemIdentifier": "<SequenceNumber>"
    }
  ]
}
```

### Note

Se l'array `batchItemFailures` contiene più elementi, Lambda utilizza il record con il numero di sequenza più basso come checkpoint. Lambda quindi riprova tutti i record a partire da quel checkpoint.

## Condizioni di successo e di errore

Lambda considera un batch come un successo completo se si restituisce uno degli elementi seguenti:

- Una `batchItemFailure` lista vuota
- Un `batchItemFailure` elenco nullo
- Un vuoto `EventResponse`
- Un valore nullo `EventResponse`

Lambda considera un batch come un fallimento completo se si restituisce uno degli elementi seguenti:

- Una stringa vuota `itemIdentifier`



- Un valore nullo `itemIdentifier`
- Un `itemIdentifier` con un nome chiave errato

Lambda esegue nuovi tentativi in seguito ai fallimenti secondo la strategia di tentativi impostata.

### Bisezione un batch

Se il richiamo fallisce ed `BisectBatchOnFunctionError` è attivato, il batch viene bisecato a prescindere dalle `ReportBatchItemFailures` impostazioni.

Quando si riceve una risposta di successo parziale del batch ed entrambi `BisectBatchOnFunctionError` e `ReportBatchItemFailures` sono attivati, il batch viene bisecato in corrispondenza del numero di sequenza restituito e Lambda ritenta solo i record rimanenti.

Per semplificare l'implementazione della logica di risposta batch parziale, prendi in considerazione l'utilizzo dell'utilità [Batch Processor](#) di Powertools for AWS Lambda, che gestisce automaticamente queste complessità per te.

Di seguito sono riportati alcuni esempi di codice di funzione che restituiscono l'elenco degli ID dei messaggi con errori nel batch:

## .NET

### SDK per .NET

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
using System.Text.Json;  
using System.Text;  
using Amazon.Lambda.Core;  
using Amazon.Lambda.DynamoDBEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
        ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
        {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
        List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
                { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

## Go

## SDK per Go V2

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
    ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
    BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
(*BatchResult, error) {
    var batchItemFailures []BatchItemFailure
    curRecordSequenceNumber := ""

    for _, record := range event.Records {
        // Process your record
        curRecordSequenceNumber = record.Change.SequenceNumber
    }

    if curRecordSequenceNumber != "" {
        batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
curRecordSequenceNumber})
    }
}
```

```

    }

    batchResult := BatchResult{
        BatchItemFailures: batchItemFailures,
    }

    return &batchResult, nil
}

func main() {
    lambda.Start(HandleRequest)
}

```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite Java.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;

import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
    StreamsEventResponse> {

    @Override

```

```
public StreamsEventResponse handleRequest(DynamodbEvent input, Context
context) {

    List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
    ArrayList<>();
    String curRecordSequenceNumber = "";

    for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
input.getRecords()) {
        try {
            //Process your record
            StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
            curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

        } catch (Exception e) {
            /* Since we are working with streams, we can return the failed
item immediately.
            Lambda will immediately begin to retry processing from this
failed item onwards. */
            batchItemFailures.add(new
StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
            return new StreamsEventResponse(batchItemFailures);
        }
    }

    return new StreamsEventResponse();
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione dei guasti degli articoli del batch DynamoDB con Lambda using. JavaScript

```

export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};

```

### Segnalazione degli errori degli elementi batch di DynamoDB con Lambda using. TypeScript

```

import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }
}

```

```
return { batchItemFailures: batchItemFailures };  
};
```

## PHP

### SDK per PHP

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite PHP.

```
<?php  
  
# using bref/bref and bref/logger for simplicity  
  
use Bref\Context\Context;  
use Bref\Event\DynamoDb\DynamoDbEvent;  
use Bref\Event\Handler as StdHandler;  
use Bref\Logger\StderrLogger;  
  
require __DIR__ . '/vendor/autoload.php';  
  
class Handler implements StdHandler  
{  
    private StderrLogger $logger;  
    public function __construct(StderrLogger $logger)  
    {  
        $this->logger = $logger;  
    }  
  
    /**  
     * @throws JsonException  
     * @throws \Bref\Event\InvalidLambdaEvent  
     */  
    public function handle(mixed $event, Context $context): array  
    {  
        $dynamoDbEvent = new DynamoDbEvent($event);
```

```
$this->logger->info("Processing records");

$records = $dynamoDbEvent->getRecords();
$failedRecords = [];
foreach ($records as $record) {
    try {
        $data = $record->getData();
        $this->logger->info(json_encode($data));
        // TODO: Do interesting work based on the new data
    } catch (Exception $e) {
        $this->logger->error($e->getMessage());
        // failed processing the record
        $failedRecords[] = $record->getSequenceNumber();
    }
}
$totalRecords = count($records);
$this->logger->info("Successfully processed $totalRecords records");

// change format for the response
$failures = array_map(
    fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
    $failedRecords
);

return [
    'batchItemFailures' => $failures
];
}

}

$logger = new StderrLogger();
return new Handler($logger);
```



## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite Ruby.

```
def lambda_handler(event:, context:)
  records = event["Records"]
  cur_record_sequence_number = ""

  records.each do |record|
    begin
      # Process your record
      cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
      # Return failed record's sequence number
      return {"batchItemFailures" => [{"itemIdentifier" =>
cur_record_sequence_number}]}
    end
  end

  {"batchItemFailures" => []}
end
```

## Rust

### SDK per Rust

#### Note

C'è altro su. [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite Rust.

```
use aws_lambda_events::{
  event::dynamodb::{Event, EventRecord, StreamRecord},
  streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
  let stream_record: &StreamRecord = &record.change;
```

```

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("").to_string(),
            });
            return Ok(response);
        }

        // Process your record here...
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
            immediately.
            Lambda will immediately begin to retry processing from this failed
            item onwards. */
            return Ok(response);
        }
    }
}

```

```

        tracing::info!("Successfully processed {} record(s)", records.len());

        Ok(response)
    }

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
        time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```

## Usare Powertools per AWS Lambda processore batch

L'utilità di elaborazione batch di Powertools per la gestione AWS Lambda automatica della logica di risposta parziale ai batch, riducendo la complessità dell'implementazione della segnalazione degli errori dei batch. Ecco alcuni esempi di utilizzo del processore batch:

### Python

#### Note

Per esempi completi e istruzioni di configurazione, consultate la documentazione del processore [batch](#).

Elaborazione dei record di flusso DynamoDB con processore AWS Lambda batch.

```

import json
from aws_lambda_powertools import Logger
from aws_lambda_powertools.utilities.batch import BatchProcessor, EventType,
    process_partial_response

```

```

from aws_lambda_powertools.utilities.data_classes import DynamoDBStreamEvent
from aws_lambda_powertools.utilities.typing import LambdaContext

processor = BatchProcessor(event_type=EventType.DynamoDBStreams)
logger = Logger()

def record_handler(record):
    logger.info(record)
    # Your business logic here
    # Raise an exception to mark this record as failed

def lambda_handler(event, context: LambdaContext):
    return process_partial_response(
        event=event,
        record_handler=record_handler,
        processor=processor,
        context=context
    )

```

## TypeScript

### Note

Per esempi completi e istruzioni di configurazione, consulta la documentazione del processore [batch](#).

Elaborazione dei record di flusso DynamoDB con processore AWS Lambda batch.

```

import { BatchProcessor, EventType, processPartialResponse } from '@aws-lambda-powertools/batch';
import { Logger } from '@aws-lambda-powertools/logger';
import type { DynamoDBStreamEvent, Context } from 'aws-lambda';

const processor = new BatchProcessor(EventType.DynamoDBStreams);
const logger = new Logger();

const recordHandler = async (record: any): Promise<void> => {
    logger.info('Processing record', { record });
    // Your business logic here
    // Throw an error to mark this record as failed
};

```

```
export const handler = async (event: DynamoDBStreamEvent, context: Context) => {
    return processPartialResponse(event, recordHandler, processor, {
        context,
    });
};
```

## Java

### Note

Per esempi completi e istruzioni di configurazione, consulta la documentazione del processore [batch](#).

Elaborazione dei record di flusso DynamoDB con processore AWS Lambda batch.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import software.amazon.lambda.powertools.batch.BatchMessageHandlerBuilder;
import software.amazon.lambda.powertools.batch.handler.BatchMessageHandler;

public class DynamoDBStreamBatchHandler implements RequestHandler<DynamodbEvent,
StreamsEventResponse> {

    private final BatchMessageHandler<DynamodbEvent, StreamsEventResponse> handler;

    public DynamoDBStreamBatchHandler() {
        handler = new BatchMessageHandlerBuilder()
            .withDynamoDbBatchHandler()
            .buildWithRawMessageHandler(this::processMessage);
    }

    @Override
    public StreamsEventResponse handleRequest(DynamodbEvent ddbEvent, Context
context) {
        return handler.processBatch(ddbEvent, context);
    }

    private void processMessage(DynamodbEvent.DynamodbStreamRecord
dynamodbStreamRecord, Context context) {
```

```

        // Process the change record
    }
}

```

## .NET

### Note

Per esempi completi e istruzioni di configurazione, consulta la documentazione del processore [batch](#).

Elaborazione dei record di flusso DynamoDB con processore AWS Lambda batch.

```

using System;
using System.Threading;
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;
using Amazon.Lambda.Serialization.SystemTextJson;
using AWS.Lambda.Powertools.BatchProcessing;

[assembly: LambdaSerializer(typeof(DefaultLambdaJsonSerializer))]

namespace HelloWorld;

public class Customer
{
    public string? CustomerId { get; set; }
    public string? Name { get; set; }
    public string? Email { get; set; }
    public DateTime CreatedAt { get; set; }
}

internal class TypedDynamoDbRecordHandler : ITypedRecordHandler<Customer>
{
    public async Task<RecordHandlerResult> HandleAsync(Customer customer,
        CancellationToken cancellationToken)
    {
        if (string.IsNullOrEmpty(customer.Email))
        {
            throw new ArgumentException("Customer email is required");
        }
    }
}

```

```
    }

    return await Task.FromResult(RecordHandlerResult.None);
}

public class Function
{
    [BatchProcessor(TypedRecordHandler = typeof(TypedDynamoDbRecordHandler))]
    public BatchItemFailuresResponse HandlerUsingTypedAttribute(DynamoDBEvent _)
    {
        return TypedDynamoDbStreamBatchProcessor.Result.BatchItemFailuresResponse;
    }
}
```

## Conservare i record scartati per un'origine eventi DynamoDB in Lambda

La gestione degli errori per gli strumenti di mappatura dell'origine degli eventi DynamoDB dipende dal fatto che l'errore si verifichi prima che la funzione venga richiamata o durante la chiamata della funzione:

- Prima della chiamata: se una mappatura dell'origine degli eventi Lambda non è in grado di richiamare la funzione a causa di limitazioni o altri problemi, riprova finché i record non scadono o superano l'età massima configurata nella mappatura dell'origine dell'evento (). [MaximumRecordAgeInSeconds](#)
- Durante la chiamata: se la funzione viene richiamata ma restituisce un errore, Lambda riprova fino alla scadenza dei record, al superamento dell'età massima () o al raggiungimento della quota di tentativi configurata ([MaximumRecordAgeInSeconds](#)). [MaximumRetryAttempts](#) Per gli errori di funzione, puoi anche configurare, che divide un batch non riuscito in due batch più piccoli [BisectBatchOnFunctionError](#), isolando i record non validi ed evitando i timeout. La divisione dei batch non consuma la quota di tentativi.

Se le misure di gestione degli errori non riescono, Lambda elimina i record e continua l'elaborazione dei batch dal flusso. Con le impostazioni predefinite, ciò significa che un record errato può bloccare l'elaborazione sullo shard interessata per un massimo di un giorno. Per evitare questa situazione, configura la mappatura dell'origine eventi della funzione con un numero ragionevole di tentativi e un'età massima dei record che sia adatta al caso d'uso.



## Configurazione delle destinazioni per le chiamate non riuscite

Per mantenere i record delle chiamate non riuscite allo strumento di mappatura dell'origine degli eventi, aggiungi una destinazione allo strumento di mappatura dell'origine degli eventi della funzione. Ogni record inviato alla destinazione è un documento JSON contenente i metadati relativi alla chiamata non riuscita. Per le destinazioni Amazon S3, Lambda invia insieme ai metadati anche l'intero record di invocazione. Puoi configurare qualsiasi argomento Amazon SNS, coda Amazon SQS, bucket Amazon S3 o Kafka come destinazione.

Con le destinazioni Amazon S3, puoi utilizzare la funzionalità [Notifiche eventi Amazon S3](#) per ricevere notifiche quando gli oggetti vengono caricati nel bucket S3 di destinazione. Puoi anche configurare Notifiche eventi S3 per richiamare un'altra funzione Lambda per eseguire l'elaborazione automatica su batch non riusciti.

Il tuo ruolo di esecuzione deve avere le autorizzazioni per la destinazione:

- [Per una destinazione SQS: sqs: SendMessage](#)
- [Per una destinazione SNS: sns:publish](#)
- [Per una destinazione S3: s3: e s3: PutObject ListBucket](#)
- [Per una destinazione Kafka: kafka-cluster: WriteData](#)

È possibile configurare un argomento di Kafka come destinazione in caso di errore per le mappature delle sorgenti degli eventi Kafka. Quando Lambda non è in grado di elaborare i record dopo estenuanti tentativi o quando i record superano l'età massima, Lambda invia i record non riusciti all'argomento Kafka specificato per un'elaborazione successiva. Fai riferimento a [the section called "Destinazione Kafka in caso di errore"](#).

[Se hai abilitato la crittografia con la tua chiave KMS per una destinazione S3, anche il ruolo di esecuzione della funzione deve avere l'autorizzazione a chiamare kms: GenerateDataKey](#) Se la chiave KMS e la destinazione del bucket S3 si trovano in un account diverso dalla funzione Lambda e dal ruolo di esecuzione, configura la chiave KMS in modo che consideri attendibile il ruolo di esecuzione da consentire. kms: GenerateDataKey

Per configurare una destinazione in caso di errore tramite la console, completa i seguenti passaggi:

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.

3. In Function overview (Panoramica delle funzioni), scegliere Add destination (Aggiungi destinazione).
4. Per Origine, scegli Chiamata allo strumento di mappatura dell'origine degli eventi.
5. Per Strumento di mappatura dell'origine degli eventi, scegli un'origine dell'evento configurata per questa funzione.
6. Per Condizione, seleziona In caso di errore. Per le chiamate allo strumento di mappatura dell'origine degli eventi, questa è l'unica condizione accettata.
7. Per Tipo di destinazione, scegli il tipo di destinazione a cui Lambda deve inviare i record di chiamata.
8. Per Destination (Destinazione), scegliere una risorsa.
9. Scegli Save (Salva).

Puoi anche configurare una destinazione in caso di errore utilizzando (). AWS Command Line Interface AWS CLI Ad esempio, il [create-event-source-mapping](#) comando seguente aggiunge una mappatura dell'origine degli eventi con una destinazione SQS in caso di errore a: MyFunction

```
aws lambda create-event-source-mapping \  
--function-name "MyFunction" \  
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/  
stream/2024-06-10T19:26:16.525 \  
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-  
east-1:123456789012:dest-queue"}}'
```

Il [update-event-source-mapping](#) comando seguente aggiorna una mappatura dell'origine degli eventi per inviare i record di chiamata non riuscita a una destinazione SNS dopo due tentativi o se i record risalgono a più di un'ora.

```
aws lambda update-event-source-mapping \  
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \  
--maximum-retry-attempts 2 \  
--maximum-record-age-in-seconds 3600 \  
--destination-config '{"OnFailure": {"Destination": "arn:aws:sns:us-  
east-1:123456789012:dest-topic"}}'
```

Le impostazioni aggiornate sono applicate in modo asincrono e non sono riflesse nell'output fino al completamento del processo. Utilizza il comando [get-event-source-mapping](#) per visualizzare lo stato corrente.

Per rimuovere una destinazione, fornisci una stringa vuota come argomento del parametro `destination-config`:

```
aws lambda update-event-source-mapping \  
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \  
--destination-config '{"OnFailure": {"Destination": ""}}'
```

### Best practice di sicurezza per destinazioni Amazon S3

L'eliminazione di un bucket S3 configurato come destinazione senza rimuovere la destinazione dalla configurazione della funzione può creare un rischio per la sicurezza. Se un altro utente conosce il nome del bucket di destinazione, può ricreare il bucket nel proprio Account AWS. I record delle invocazioni non riuscite verranno inviati al relativo bucket, esponendo potenzialmente i dati della tua funzione.

#### Warning

Per garantire che i record di invocazione della tua funzione non possano essere inviati a un bucket S3 in un altro Account AWS, aggiungi una condizione al ruolo di esecuzione della funzione che limiti le autorizzazioni ai bucket del tuo account. `s3:PutObject`

Di seguito viene illustrato un esempio di policy IAM che limita le autorizzazioni `s3:PutObject` della funzione ai bucket presenti nell'account. Questa policy fornisce inoltre a Lambda l'autorizzazione `s3:ListBucket` necessaria per utilizzare un bucket S3 come destinazione.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "S3BucketResourceAccountWrite",  
      "Effect": "Allow",  
      "Action": [  
        "s3:PutObject",  
        "s3:ListBucket"  
      ],  
      "Resource": [  
        "arn:aws:s3:::*/**",  
        "arn:aws:s3:::*"  
      ],  
      "Condition": {
```

```
        "StringEquals": {
            "s3:ResourceAccount": "111122223333"
        }
    }
}
```

Per aggiungere una politica di autorizzazioni al ruolo di esecuzione della tua funzione utilizzando Console di gestione AWS o AWS CLI, consulta le istruzioni nelle seguenti procedure:

## Console

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Seleziona la funzione Lambda di cui si desidera modificare il ruolo di esecuzione.
3. Nella scheda Configurazione scegli Autorizzazioni.
4. Nella scheda Ruolo di esecuzione, seleziona il nome del ruolo della funzione per aprire la pagina della console IAM del ruolo.
5. Aggiungi una policy di autorizzazioni di base al ruolo completando le seguenti operazioni:
  - a. Nel riquadro Policy di autorizzazioni, scegli Aggiungi autorizzazioni, poi Crea policy in linea.
  - b. Nell'editor delle policy, seleziona JSON.
  - c. Incolla la policy che desideri aggiungere nell'editor (sostituendo il codice JSON esistente), quindi scegli Avanti.
  - d. In Dettagli della policy, specifica un nome per la policy.
  - e. Scegli Crea policy.

## AWS CLI

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (CLI)

1. Crea un documento di policy JSON con le autorizzazioni richieste e salvalo in una directory locale.
2. Utilizza il comando della CLI `put-role-policy` di IAM per aggiungere le autorizzazioni per il ruolo di esecuzione di una funzione. Esegui il comando seguente dalla directory in cui hai

salvato il documento di policy JSON e sostituisci il nome del ruolo, il nome della policy e il documento di policy con i tuoi valori.

```
aws iam put-role-policy \  
--role-name my_lambda_role \  
--policy-name LambdaS3DestinationPolicy \  
--policy-document file://my_policy.json
```

Record di invocazione Amazon SNS e Amazon SQS di esempio

L'esempio seguente mostra un record di chiamata che Lambda invia a una destinazione SQS o SNS per un flusso DynamoDB.

```
{  
  "requestContext": {  
    "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",  
    "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",  
    "condition": "RetryAttemptsExhausted",  
    "approximateInvokeCount": 1  
  },  
  "responseContext": {  
    "statusCode": 200,  
    "executedVersion": "$LATEST",  
    "functionError": "Unhandled"  
  },  
  "version": "1.0",  
  "timestamp": "2019-11-14T00:13:49.717Z",  
  "DDBStreamBatchInfo": {  
    "shardId": "shardId-00000001573689847184-864758bb",  
    "startSequenceNumber": "800000000003126276362",  
    "endSequenceNumber": "800000000003126276362",  
    "approximateArrivalOfFirstRecord": "2019-11-14T00:13:19Z",  
    "approximateArrivalOfLastRecord": "2019-11-14T00:13:19Z",  
    "batchSize": 1,  
    "streamArn": "arn:aws:dynamodb:us-east-2:123456789012:table/mytable/  
stream/2019-11-14T00:04:06.388"  
  }  
}
```

È possibile utilizzare queste informazioni per recuperare i record interessati dal flusso per la risoluzione dei problemi. I record effettivi non sono inclusi, pertanto è necessario elaborare questo record e recuperarli dal flusso prima che scadano e vadano persi.

### Record di invocazione Amazon S3 di esempio

L'esempio seguente mostra un record di invocazione che Lambda invia a un bucket S3 per un flusso DynamoDB. Oltre a tutti i campi dell'esempio precedente per le destinazioni SQS e SNS, il campo `payload` contiene il record di chiamata originale come stringa JSON con escape.

```
{
  "requestContext": {
    "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
    "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",
    "condition": "RetryAttemptsExhausted",
    "approximateInvokeCount": 1
  },
  "responseContext": {
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Unhandled"
  },
  "version": "1.0",
  "timestamp": "2019-11-14T00:13:49.717Z",
  "DDBStreamBatchInfo": {
    "shardId": "shardId-00000001573689847184-864758bb",
    "startSequenceNumber": "800000000003126276362",
    "endSequenceNumber": "800000000003126276362",
    "approximateArrivalOfFirstRecord": "2019-11-14T00:13:19Z",
    "approximateArrivalOfLastRecord": "2019-11-14T00:13:19Z",
    "batchSize": 1,
    "streamArn": "arn:aws:dynamodb:us-east-2:123456789012:table/mytable/stream/2019-11-14T00:04:06.388"
  },
  "payload": "<Whole Event>" // Only available in S3
}
```

L'oggetto S3 contenente il record di invocazione utilizza la seguente convenzione di denominazione:

```
aws/lambda/<ESM-UUID>/<shardID>/YYYY/MM/DD/YYYY-MM-DDTHH.MM.SS-<Random UUID>
```

## Implementazione dell'elaborazione del flusso DynamoDB stateful in Lambda

Le funzioni Lambda possono eseguire applicazioni di elaborazione di flussi continui. Un flusso rappresenta dati illimitati che scorrono continuamente attraverso l'applicazione. Per analizzare le informazioni da questo input di aggiornamento continuo, è possibile associare i record inclusi utilizzando una finestra definita in termini di tempo.

Le finestre a cascata sono finestre temporali distinte che si aprono e si chiudono a intervalli regolari. Per impostazione predefinita, le invocazioni Lambda sono senza stato: non è possibile utilizzarle per l'elaborazione dei dati tra più invocazioni continue senza un database esterno. Tuttavia, con la finestra a cascata, è possibile mantenere il proprio stato tra le invocazioni.

Questo stato contiene il risultato aggregato dei messaggi precedentemente elaborati per la finestra corrente. Il tuo stato può essere un massimo di 1 MB per ogni shard. Se supera quella dimensione, Lambda termina la finestra in anticipo.

Ogni record in un flusso appartiene a una finestra specifica. Lambda elabora ogni record almeno una volta, ma non garantisce che ogni record venga elaborato una sola volta. In rari casi, come nel caso della gestione degli errori, alcuni record potrebbero essere elaborati più di una volta. La prima volta i record vengono sempre elaborati in ordine. Se i record vengono elaborati più di una volta, possono essere elaborati fuori ordine.

### Aggregazione ed elaborazione

La funzione gestita dall'utente viene richiamata sia per l'aggregazione che per l'elaborazione dei risultati finali di tale aggregazione. Lambda aggrega tutti i record ricevuti nella finestra. È possibile ricevere questi record in più batch, ciascuno come richiamo separato. Ogni richiamo riceve uno stato.

Pertanto, quando si utilizzano finestre a cascata, la risposta della funzione Lambda deve contenere una proprietà di `state`. Se la risposta non contiene una proprietà di `state`, Lambda la considera un'invocazione non riuscita. Per soddisfare questa condizione, la funzione può restituire un oggetto `TimeWindowEventResponse`, che presenta la seguente forma JSON:

### Example `TimeWindowEventResponse` valori

```
{
  "state": {
    "1": 282,
    "2": 715
  },
  "batchItemFailures": []
}
```

```
}
```

### Note

Per le funzioni Java, si consiglia di utilizzare una `Map<String, String>` per rappresentare lo stato.

Alla fine della finestra, il flag `isFinalInvokeForWindow` è impostato `true` per indicare che questo è lo stato finale e che è pronto per l'elaborazione. Dopo l'elaborazione, la finestra viene completata e il richiamo finale viene completato e quindi lo stato viene eliminato.

Al termine della finestra, Lambda utilizza l'elaborazione finale per le operazioni sui risultati dell'aggregazione. L'elaborazione finale viene richiamata in modo sincrono. Dopo il richiamo riuscito, la funzione controlla il numero di sequenza e l'elaborazione del flusso continua. Se il richiamo non ha esito positivo, la funzione Lambda sospende l'ulteriore elaborazione fino a quando non viene eseguito correttamente il richiamo.

### Example DynamoDbTimeWindowEvent

```
{
  "Records": [
    {
      "eventID": "1",
      "eventName": "INSERT",
      "eventVersion": "1.0",
      "eventSource": "aws:dynamodb",
      "awsRegion": "us-east-1",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        }
      }
    }
  ]
}
```



```

        }
      },
      "SequenceNumber": "111",
      "SizeBytes": 26,
      "StreamViewType": "NEW_AND_OLD_IMAGES"
    },
    "eventSourceARN": "stream-ARN"
  },
  {
    "eventID": "2",
    "eventName": "MODIFY",
    "eventVersion": "1.0",
    "eventSource": "aws:dynamodb",
    "awsRegion": "us-east-1",
    "dynamodb": {
      "Keys": {
        "Id": {
          "N": "101"
        }
      },
      "NewImage": {
        "Message": {
          "S": "This item has changed"
        },
        "Id": {
          "N": "101"
        }
      },
      "OldImage": {
        "Message": {
          "S": "New item!"
        },
        "Id": {
          "N": "101"
        }
      },
      "SequenceNumber": "222",
      "SizeBytes": 59,
      "StreamViewType": "NEW_AND_OLD_IMAGES"
    },
    "eventSourceARN": "stream-ARN"
  },
  {
    "eventID": "3",

```

```

    "eventName": "REMOVE",
    "eventVersion": "1.0",
    "eventSource": "aws:dynamodb",
    "awsRegion": "us-east-1",
    "dynamodb": {
      "Keys": {
        "Id": {
          "N": "101"
        }
      },
      "OldImage": {
        "Message": {
          "S": "This item has changed"
        },
        "Id": {
          "N": "101"
        }
      },
      "SequenceNumber": "333",
      "SizeBytes": 38,
      "StreamViewType": "NEW_AND_OLD_IMAGES"
    },
    "eventSourceARN": "stream-ARN"
  },
  ],
  "window": {
    "start": "2020-07-30T17:00:00Z",
    "end": "2020-07-30T17:05:00Z"
  },
  "state": {
    "1": "state1"
  },
  "shardId": "shard123456789",
  "eventSourceARN": "stream-ARN",
  "isFinalInvokeForWindow": false,
  "isWindowTerminatedEarly": false
}

```

## Configurazione

È possibile configurare le finestre a cascata quando si crea o si aggiorna un mapping di origini di eventi. Per configurare una finestra rotante, specificate la finestra in secondi ([TumblingWindowInSeconds](#)). Il seguente comando example AWS Command Line Interface (AWS

CLI) crea una mappatura della sorgente di un evento di streaming con una finestra rotante di 120 secondi. La funzione Lambda definita per l'aggregazione e l'elaborazione è denominata `tumbling-window-example-function`.

```
aws lambda create-event-source-mapping \  
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/  
stream/2024-06-10T19:26:16.525 \  
--function-name tumbling-window-example-function \  
--starting-position TRIM_HORIZON \  
--tumbling-window-in-seconds 120
```

Lambda determina i limiti delle finestre a cascata in base al momento in cui i record sono stati inseriti nel flusso. Tutti i record dispongono di un timestamp approssimativo che Lambda utilizza nelle determinazioni dei limiti.

Le aggregazioni delle finestre a cascata non supportano il risharding. Quando lo shard termina, Lambda considera la finestra chiusa e gli shard secondari iniziano la propria finestra in uno stato nuovo.

Le finestre a cascata supportano completamente le policy di ripetizione dei tentativi esistenti `maxRetryAttempts` e `maxRecordAge`.

#### Example Handler.py — Aggregazione ed elaborazione

La seguente funzione Python mostra come aggregare e quindi elaborare lo stato finale:

```
def lambda_handler(event, context):  
    print('Incoming event: ', event)  
    print('Incoming state: ', event['state'])  
  
    #Check if this is the end of the window to either aggregate or process.  
    if event['isFinalInvokeForWindow']:  
        # logic to handle final state of the window  
        print('Destination invoke')  
    else:  
        print('Aggregate invoke')  
  
    #Check for early terminations  
    if event['isWindowTerminatedEarly']:  
        print('Window terminated early')
```

```
#Aggregation logic
state = event['state']
for record in event['Records']:
    state[record['dynamodb']['NewImage']['Id']] = state.get(record['dynamodb']
['NewImage']['Id'], 0) + 1

print('Returning state: ', state)
return {'state': state}
```

## Parametri Lambda per gli strumenti di mappatura dell'origine degli eventi di Amazon DynamoDB

Tutti i tipi di sorgenti di eventi Lambda condividono le stesse operazioni [CreateEventSourceMapping](#) e le [UpdateEventSourceMapping](#) API. Tuttavia, solo alcuni dei parametri si applicano a DynamoDB Streams.

| Parametro                  | Obbligatorio | Predefinito | Note                                                                                                               |
|----------------------------|--------------|-------------|--------------------------------------------------------------------------------------------------------------------|
| BatchSize                  | N            | 100         | Massimo: 10.000.                                                                                                   |
| BisectBatchOnFunctionError | N            | false       | nessuno                                                                                                            |
| DestinationConfig          | N            | N/A         | Una destinazione della coda standard di Amazon SQS o di un argomento standard di Amazon SNS per i record scartati. |
| Abilitato                  | N            | true        | nessuno                                                                                                            |
| EventSourceArn             | Y            | N/A         | L'ARN del flusso dei dati o di un consumer di flusso.                                                              |
| FilterCriteria             | N            | N/A         | <a href="#">Controllare gli eventi che Lambda invia alla funzione</a>                                              |

| Parametro                      | Obbligatorio | Predefinito | Note                                                                                                                                                                                                                                                                  |
|--------------------------------|--------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FunctionName                   | Y            | N/A         | nessuno                                                                                                                                                                                                                                                               |
| FunctionResponseTypes          | N            | N/A         | Per consentire alla funzione di segnalare errori specifici in un batch, includi il valore ReportBatchItemFailures in FunctionResponseTypes. Per ulteriori informazioni, consulta <a href="#">Configurazione della risposta batch parziale con DynamoDB e Lambda</a> . |
| MaximumBatchingWindowInSeconds | N            | 0           | nessuno                                                                                                                                                                                                                                                               |
| MaximumRecordAgeInSeconds      | N            | -1          | -1 sta per infinito: i record non riusciti vengono ripetuti fino alla scadenza del record. Il <a href="#">limite di conservazione dei dati per i flussi DynamoDB</a> è di 24 ore.<br><br>Minimo: -1<br><br>Massimo: 604.800                                           |

| Parametro               | Obbligatorio | Predefinito | Note                                                                                                                               |
|-------------------------|--------------|-------------|------------------------------------------------------------------------------------------------------------------------------------|
| MaximumRetryAttempts    | N            | -1          | -1 sta per infinito: i record non riusciti vengono ripetuti fino alla scadenza del record<br><br>Minimo: 0<br><br>Massimo: 10.000. |
| ParallelizationFactor   | N            | 1           | Maximum: 10                                                                                                                        |
| StartingPosition        | Y            | N/A         | TRIM_HORIZON o LATEST                                                                                                              |
| TumblingWindowInSeconds | N            | N/A         | Minimo: 0<br><br>Massimo: 900                                                                                                      |

## Utilizzo del filtro eventi con un'origine eventi DynamoDB

Puoi utilizzare il filtraggio degli eventi per controllare quali record di un flusso o di una coda Lambda invia alla funzione. Per informazioni generali sul funzionamento del filtraggio eventi, consulta [the section called “Filtro eventi”](#).

In questa sezione viene descritto il filtraggio degli eventi per le origini eventi DynamoDB.

### Note

Le mappature delle sorgenti degli eventi DynamoDB supportano solo il filtraggio sulla chiave. dynamodb

## Argomenti

- [Evento DynamoDB](#)
- [Filtraggio con attributi di tabella](#)

- [Filtraggio con espressioni booleane](#)
- [Utilizzo dell'operatore Exists](#)
- [Formato JSON per il filtro DynamoDB](#)

## Evento DynamoDB

Supponiamo di avere una tabella DynamoDB con la chiave primaria `CustomerName` e gli attributi `AccountManager` e `PaymentTerms`. Quanto segue mostra un record di esempio dallo stream della tua tabella DynamoDB.

```
{
  "eventID": "1",
  "eventVersion": "1.0",
  "dynamodb": {
    "ApproximateCreationDateTime": "1678831218.0",
    "Keys": {
      "CustomerName": {
        "S": "AnyCompany Industries"
      }
    },
    "NewImage": {
      "AccountManager": {
        "S": "Pat Candella"
      },
      "PaymentTerms": {
        "S": "60 days"
      },
      "CustomerName": {
        "S": "AnyCompany Industries"
      }
    },
    "SequenceNumber": "111",
    "SizeBytes": 26,
    "StreamViewType": "NEW_IMAGE"
  }
}
```

Per filtrare in base ai valori della chiave e degli attributi nella tabella DynamoDB, utilizza la chiave `dynamodb` nel record. Le seguenti sezioni forniscono esempi per diversi tipi di filtri.

## Filtraggio con chiavi di tabella

Supponiamo che tu voglia che la tua funzione elabori solo i record in cui la chiave primaria `CustomerName` è "AnyCompany Industries». L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"dynamodb\" : { \"Keys\" : { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"
    }
  ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
  "dynamodb": {
    "Keys": {
      "CustomerName": {
        "S": [ "AnyCompany Industries" ]
      }
    }
  }
}
```

Puoi aggiungere il filtro utilizzando la console AWS CLI o un AWS SAM modello.

### Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in [Collegamento dei criteri di filtro a una mappatura dell'origine evento \(console\)](#) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "dynamodb" : { "Keys" : { "CustomerName" : { "S" : [ "AnyCompany Industries" ] } } } }
```

### AWS CLI

Per creare una nuova mappatura della sorgente di eventi con questi criteri di filtro utilizzando AWS Command Line Interface (AWS CLI), esegui il comando seguente.



```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"Keys\" : { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"]}]'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"Keys\" : { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"]}]'
```

## AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungete il seguente frammento al modello YAML per la fonte dell'evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "Keys" : { "CustomerName" : { "S" : [ "AnyCompany Industries" ] } } } }'
```

## Filtraggio con attributi di tabella

Con DynamoDB, puoi anche utilizzare le chiavi NewImage e OldImage per filtrare i valori degli attributi. Supponiamo di voler filtrare i record in cui l'AccountManager attributo nell'ultima immagine della tabella è «Pat Candella» o «Shirley Rodriguez». L'oggetto FilterCriteria dovrebbe avere la struttura seguente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\", \"Shirley Rodriguez\" ] } } } }"
    }
  ]
}
```

Per una maggiore chiarezza, ecco il valore del Pattern del filtro espanso in JSON semplice.

```
{
  "dynamodb": {
    "NewImage": {
      "AccountManager": {
        "S": [ "Pat Candella", "Shirley Rodriguez" ]
      }
    }
  }
}
```

Puoi aggiungere il filtro utilizzando la console o un modello. AWS CLI AWS SAM

### Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in [Collegamento dei criteri di filtro a una mappatura dell'origine evento \(console\)](#) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat Candella",
"Shirley Rodriguez" ] } } } }
```

### AWS CLI

Per creare una nuova mappatura della sorgente di eventi con questi criteri di filtro utilizzando AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\", \"Shirley Rodriguez\" ] } } } }"]}]'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
```

```
--filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\", \"Shirley Rodriguez\" ] } } } }"]}]'
```

## AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungete il seguente frammento al modello YAML per la fonte dell'evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat Candella", "Shirley Rodriguez" ] } } } }'
```

## Filtraggio con espressioni booleane

È inoltre possibile creare filtri utilizzando espressioni booleane AND. Queste espressioni possono includere sia i parametri chiave sia quelli degli attributi della tabella. Supponiamo che tu voglia filtrare i record in cui il valore di NewImage di AccountManager è "Pat Candella" e il valore di OldImage è "Terry Whitlock". L'oggetto FilterCriteria dovrebbe avere la struttura seguente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" : { \"OldImage\" : { \"AccountManager\" : { \"S\" : [ \"Terry Whitlock\" ] } } } }"
    }
  ]
}
```

Per una maggiore chiarezza, ecco il valore del Pattern del filtro espanso in JSON semplice.

```
{
  "dynamodb" : {
    "NewImage" : {
      "AccountManager" : {
        "S" : [
          "Pat Candella"
        ]
      }
    }
  }
```

```

    }
  }
},
"dynamodb": {
  "OldImage": {
    "AccountManager": {
      "S": [
        "Terry Whitlock"
      ]
    }
  }
}
}
}
}

```

Puoi aggiungere il filtro usando la console o un modello. AWS CLI AWS SAM

### Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in [Collegamento dei criteri di filtro a una mappatura dell'origine evento \(console\)](#) e inserisci la seguente stringa per i criteri di filtraggio.

```

{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" :
[ "Terry Whitlock" ] } } } } }

```

### AWS CLI

Per creare una nuova mappatura della sorgente di eventi con questi criteri di filtro utilizzando AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```

aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" : { \"OldImage\" : { \"AccountManager\" : { \"S\" : [ \"Terry Whitlock\" ] } } } } }"]}]'

```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" : { \"OldImage\" : { \"AccountManager\" : { \"S\" : [ \"Terry Whitlock\" ] } } } } }']'
```

## AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungete il seguente frammento al modello YAML per la fonte dell'evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" : [ "Terry Whitlock" ] } } } }'
```

### Note

Il filtro degli eventi di DynamoDB non supporta l'uso di operatori numerici (valori numerici uguali e intervallo numerico). Anche se gli elementi della tabella sono memorizzati come numeri, questi parametri vengono convertiti in stringhe nell'oggetto record JSON.

## Utilizzo dell'operatore Exists

A causa del modo in cui sono strutturati gli oggetti evento JSON di DynamoDB, l'utilizzo dell'operatore Exists richiede particolare attenzione. L'operatore Exists funziona solo sui nodi foglia nel JSON dell'evento, quindi se il modello di filtro utilizza Exists per testare un nodo intermedio, non funzionerà. Considera il seguente elemento della tabella DynamoDB:

```
{
  "UserID": {"S": "12345"},
  "Name": {"S": "John Doe"},
  "Organizations": {"L": [
    {"S": "Sales"},
    {"S": "Marketing"},
    {"S": "Support"}
  ]}
}
```

```

    ]
  }
}
```

Potresti voler creare uno schema di filtro come il seguente per verificare la presenza di eventi contenenti "Organizations":

```
{ "dynamodb" : { "NewImage" : { "Organizations" : [ { "exists": true } ] } } }
```

Tuttavia, questo modello di filtro non restituirebbe mai una corrispondenza perché "Organizations" non è un nodo foglia. L'esempio seguente mostra come utilizzare correttamente l'operatore Exists per creare lo schema di filtro desiderato:

```
{ "dynamodb" : { "NewImage" : { "Organizations": { "L": { "S": [ { "exists":
true } ] } } } } }
```

## Formato JSON per il filtro DynamoDB

Per filtrare correttamente gli eventi da origini DynamoDB, sia il campo dati sia i criteri di filtraggio per il campo dati (dynamodb) devono essere in un formato JSON valido. Se uno dei due campi non è in un formato JSON valido, Lambda rilascia il messaggio o genera un'eccezione. La tabella seguente riepiloga il comportamento specifico:

| Formato dei dati in entrata | Formato del modello di filtro per le proprietà di dati | Operazione risultante                                                                                                                       |
|-----------------------------|--------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| JSON valido                 | JSON valido                                            | Filtri Lambda in base ai criteri di filtro.                                                                                                 |
| JSON valido                 | Nessun modello di filtro per le proprietà dei dati     | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.                                                       |
| JSON valido                 | Non-JSON                                               | Lambda genera un'eccezione al momento della creazione o dell'aggiornamento della mappatura dell'origine evento. Il modello di filtro per le |

| Formato dei dati in entrata | Formato del modello di filtro per le proprietà di dati | Operazione risultante                                                                                                                                                                                 |
|-----------------------------|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                             |                                                        | proprietà dei dati deve essere in un formato JSON valido.                                                                                                                                             |
| Non-JSON                    | JSON valido                                            | Lambda rilascia il registro.                                                                                                                                                                          |
| Non-JSON                    | Nessun modello di filtro per le proprietà dei dati     | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.                                                                                                                 |
| Non-JSON                    | Non-JSON                                               | Lambda genera un'eccezione al momento della creazione o dell'aggiornamento della mappatura dell'origine evento. Il modello di filtro per le proprietà dei dati deve essere in un formato JSON valido. |

## Tutorial: Utilizzo AWS Lambda con stream Amazon DynamoDB

In questo tutorial creerai una funzione Lambda per consumare eventi da un flusso Amazon DynamoDB.

### Prerequisiti

#### Installa AWS Command Line Interface

Se non hai ancora installato il AWS Command Line Interface, segui i passaggi in [Installazione o aggiornamento della versione più recente AWS CLI](#) per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

#### Note

Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere

una Windows-integrated versione di Ubuntu e Bash, [installa il sottosistema Windows per Linux](#).

## Creazione del ruolo di esecuzione

Crea il ruolo di [esecuzione](#) che autorizza la tua funzione ad accedere AWS alle risorse.

Per creare un ruolo di esecuzione

1. Apri la pagina [Ruoli](#) nella console IAM.
2. Scegliere Crea ruolo.
3. Creare un ruolo con le seguenti proprietà.
  - Entità attendibile – Lambda.
  - Autorizzazioni — AWSLambdaDynamoDBExecutionRole.
  - Nome ruolo – **lambda-dynamodb-role**.

AWSLambdaDynamoDBExecutionRoleHa le autorizzazioni necessarie alla funzione per leggere gli elementi da DynamoDB e scrivere i log nei log. CloudWatch

## Creazione della funzione

Crea una funzione Lambda che elabora gli eventi DynamoDB. Il codice della funzione scrive alcuni dei dati degli eventi in entrata nei Logs. CloudWatch

.NET

SDK per .NET

### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento DynamoDB con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0
```



```
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Go

SDK per Go V2

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Utilizzo di un evento DynamoDB con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-lambda-go/events"
    "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
    error) {
    if len(event.Records) == 0 {
        return nil, fmt.Errorf("received empty event")
    }

    for _, record := range event.Records {
        LogDynamoDBRecord(record)
    }

    message := fmt.Sprintf("Records processed: %d", len(event.Records))
    return &message, nil
}

func main() {
    lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
    fmt.Println(record.EventID)
    fmt.Println(record.EventName)
    fmt.Printf("%+v\n", record.Change)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Utilizzo di un evento DynamoDB con Lambda tramite Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import
    com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
        GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
            GSON.toJson(record.getDynamodb()));
    }
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Utilizzo di un evento DynamoDB utilizzando Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));
  event.Records.forEach(record => {
    logDynamoDBRecord(record);
  });
};

const logDynamoDBRecord = (record) => {
  console.log(record.eventID);
  console.log(record.eventName);
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

### Consumo di un evento DynamoDB con Lambda utilizzando. TypeScript

```
export const handler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));
  event.Records.forEach(record => {
    logDynamoDBRecord(record);
  });
}

const logDynamoDBRecord = (record) => {
  console.log(record.eventID);
  console.log(record.eventName);
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

## PHP

### SDK per PHP

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento DynamoDB con Lambda tramite PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
        }
    }
}
```

```

        $keys = $record->getKeys();
        $old = $record->getOldImage();
        $new = $record->getNewImage();

        $this->logger->info("Event Name:".$eventName."\n");
        $this->logger->info("Keys:". json_encode($keys)."\n");
        $this->logger->info("Old Image:". json_encode($old)."\n");
        $this->logger->info("New Image:". json_encode($new));

        // TODO: Do interesting work based on the new data

        // Any exception thrown will be logged and the invocation will be
        marked as failed
    }

    $totalRecords = count($records);
    $this->logger->info("Successfully processed $totalRecords items");
}
}

$logger = new StderrLogger();
return new Handler($logger);

```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento DynamoDB con Lambda tramite Python.

```

import json

def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:

```

```
log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento DynamoDB con Lambda tramite Ruby.

```
def lambda_handler(event:, context:)
    return 'received empty event' if event['Records'].empty?

    event['Records'].each do |record|
        log_dynamodb_record(record)
    end

    "Records processed: #{event['Records'].length}"
end

def log_dynamodb_record(record)
    puts record['eventID']
    puts record['eventName']
    puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
end
```

## Rust

### SDK per Rust

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento DynamoDB con Lambda tramite Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
};

// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features = ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) -> Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}", records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records {
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");
}
```



```

    // Prepare the response
    Ok(())

}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())
}

```

Per creare la funzione

1. Copiare il codice di esempio in un file denominato `example.js`.
2. Crea un pacchetto di implementazione.

```
zip function.zip example.js
```

3. Creare una funzione Lambda con il comando `create-function`.

```

aws lambda create-function --function-name ProcessDynamoDBRecords \
    --zip-file fileb://function.zip --handler example.handler --runtime nodejs24.x \
    --role arn:aws:iam::111122223333:role/lambda-dynamodb-role

```

## Test della funzione Lambda

In questo passaggio, si richiama la funzione Lambda manualmente utilizzando il comando `invoke` AWS Lambda CLI e il seguente evento DynamoDB di esempio. Copia quanto riportato di seguito in un file denominato `input.txt`.

### Example input.txt

```
{
  "Records": [
    {
      "eventID": "1",
      "eventName": "INSERT",
      "eventVersion": "1.0",
      "eventSource": "aws:dynamodb",
      "awsRegion": "us-east-1",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
        "SequenceNumber": "111",
        "SizeBytes": 26,
        "StreamViewType": "NEW_AND_OLD_IMAGES"
      },
      "eventSourceARN": "stream-ARN"
    },
    {
      "eventID": "2",
      "eventName": "MODIFY",
      "eventVersion": "1.0",
      "eventSource": "aws:dynamodb",
      "awsRegion": "us-east-1",
      "dynamodb": {
```

```

    "Keys":{
      "Id":{
        "N":"101"
      }
    },
    "NewImage":{
      "Message":{
        "S":"This item has changed"
      },
      "Id":{
        "N":"101"
      }
    },
    "OldImage":{
      "Message":{
        "S":"New item!"
      },
      "Id":{
        "N":"101"
      }
    },
    "SequenceNumber":"222",
    "SizeBytes":59,
    "StreamViewType":"NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN":"stream-ARN"
},
{
  "eventID":"3",
  "eventName":"REMOVE",
  "eventVersion":"1.0",
  "eventSource":"aws:dynamodb",
  "awsRegion":"us-east-1",
  "dynamodb":{
    "Keys":{
      "Id":{
        "N":"101"
      }
    },
    "OldImage":{
      "Message":{
        "S":"This item has changed"
      },
      "Id":{

```

```

        "N": "101"
      }
    },
    "SequenceNumber": "333",
    "SizeBytes": 38,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "stream-ARN"
}
]
}

```

Eseguire il seguente comando `invoke`.

```

aws lambda invoke --function-name ProcessDynamoDBRecords \
  --cli-binary-format raw-in-base64-out \
  --payload file://input.txt outputfile.txt

```

L'opzione `--cli-binary-format` è obbligatoria se stai usando la versione 2. AWS CLI. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

La funzione restituisce la stringa `message` nel corpo della risposta.

Verificare l'output nel file `outputfile.txt`.

Creazione di una tabella DynamoDB con un flusso abilitato

Crea una tabella Amazon DynamoDB con un flusso abilitato.

Per creare una tabella DynamoDB

1. Aprire la [console DynamoDB](#).
2. Scegliere `Create table` (Crea tabella).
3. Creare una tabella con le impostazioni seguenti.
  - Table name (Nome tabella) – **lambda-dynamodb-stream**
  - Primary key (Chiave primaria) – **id** (string)
4. Scegliere `Create` (Crea).

## Per abilitare i flussi

1. Aprire la [console DynamoDB](#).
2. Scegliere Tables (Tabelle).
3. Scegliere la tabella lambda-dynamodb-stream.
4. In Exports and streams (Esportazioni e flussi), scegliere DynamoDB stream details (Dettagli del flusso di Dynamo DB).
5. Scegliere Turn On (Attiva).
6. Per Tipo di visualizzazione, scegli Solo attributi chiave.
7. Scegli Attiva il flusso.

Prendi nota dell'ARN del flusso. Questa operazione è necessaria nella fase successiva quando si associa il flusso alla funzione Lambda. Per ulteriori informazioni sull'attivazione dei flussi, consultare [Acquisizione dell'attività sulla tabella tramite DynamoDB Streams](#).

## Aggiungi una fonte di eventi in AWS Lambda

Creare una mappatura dell'origine eventi in AWS Lambda. Questa mappatura dell'origine eventi associa il flusso DynamoDB alla funzione Lambda. Dopo aver creato questa mappatura della sorgente di eventi, AWS Lambda avvia il polling dello stream.

Eseguire il seguente comando AWS CLI `create-event-source-mapping`. Dopo l'esecuzione del comando, annotare l'UUID. Questo UUID è necessario per fare riferimento alla mappatura delle origini eventi nei comandi, ad esempio quando si elimina tale mappatura.

```
aws lambda create-event-source-mapping --function-name ProcessDynamoDBRecords \  
--batch-size 100 --starting-position LATEST --event-source DynamoDB-stream-arn
```

Questa procedura crea una mappatura tra il flusso DynamoDB specificato e la funzione Lambda. È possibile associare un flusso DynamoDB a più funzioni Lambda e associare la stessa funzione Lambda a più flussi. Tuttavia, le funzioni Lambda condividono il throughput di lettura dello stream che condividono.

Mediante l'esecuzione del comando riportato di seguito è possibile ottenere l'elenco di mappature delle origini eventi.

```
aws lambda list-event-source-mappings
```

L'elenco restituisce tutte le mappature delle origini eventi create e per ciascuna mappatura mostra, tra l'altro, il valore `LastProcessingResult`. Questo campo è utilizzato per fornire un messaggio informativo nel caso in cui vengano riscontrati problemi. Valori come `No records processed` (indica che non AWS Lambda è stato avviato il polling o che non ci sono record nello stream) e `OK` (indica che i record sono stati letti AWS Lambda correttamente dallo stream e ha richiamato la funzione Lambda) indicano che non ci sono problemi. Se vengono riscontrati problemi, si riceve un messaggio di errore.

Se disponi di una grande quantità di mappature delle origini eventi, utilizza il parametro `funzione-nome` per restringere i risultati.

```
aws lambda list-event-source-mappings --function-name ProcessDynamoDBRecords
```

Eseguire il test della configurazione

Eseguire il test dell'esperienza end-to-end. Quando vengono eseguiti gli aggiornamenti delle tabelle, DynamoDB scrive record di eventi nel flusso. Quando AWS Lambda esegue il polling del flusso, rileva nuovi record nel flusso e richiama la funzione Lambda per proprio conto passando eventi alla funzione.

1. Nella console di DynamoDB, aggiungi, aggiorna ed elimina elementi dalla tabella. DynamoDB scrive record di queste operazioni nel flusso.
2. AWS Lambda esegue il polling dello stream e quando rileva gli aggiornamenti allo stream, richiama la funzione Lambda trasmettendo i dati dell'evento che trova nello stream.
3. La tua funzione viene eseguita e crea log in Amazon. CloudWatch Puoi verificare i log riportati nella CloudWatch console Amazon.

Fasi successive

Questo tutorial ti ha mostrato le basi dell'elaborazione degli eventi di streaming DynamoDB con Lambda. Per i carichi di lavoro di produzione, prendi in considerazione l'implementazione di una logica di risposta parziale in batch per gestire gli errori dei singoli record in modo più efficiente. L'utilità per processori [batch](#) di Powertools for AWS Lambda è disponibile in Python TypeScript, .NET e Java e fornisce una soluzione affidabile a tale scopo, gestendo automaticamente la complessità delle risposte batch parziali e riducendo il numero di tentativi per i record elaborati correttamente.

## Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti inutili addebiti al tuo Account AWS

Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **confirm** nel campo di immissione del testo, quindi scegli Elimina.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina [Ruoli](#) della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare la tabella DynamoDB

1. Aprire la pagina [Tables \(Tabelle\)](#) della console DynamoDB.
2. Selezionare la tabella creata.
3. Scegliere Delete (Elimina).
4. Immettere **delete** nella casella di testo.
5. Seleziona Delete Table (Elimina tabella).

## Elaborare gli eventi del ciclo di vita di Amazon EC2 con una funzione Lambda

Puoi utilizzarla AWS Lambda per elaborare gli eventi del ciclo di vita da Amazon Elastic Compute Cloud e gestire le risorse Amazon EC2. Amazon EC2 invia eventi ad [Amazon EventBridge \(Events\)](#) per CloudWatch eventi del [ciclo di vita](#), ad esempio quando un'istanza cambia stato, quando viene completato uno snapshot di un volume di Amazon Elastic Block Store o quando è pianificata la chiusura di un'istanza Spot. È possibile configurare EventBridge (CloudWatch Events) per inoltrare tali eventi a una funzione Lambda per l'elaborazione.

EventBridge (CloudWatch Events) richiama la funzione Lambda in modo asincrono con il documento dell'evento di Amazon EC2.

Example Evento del ciclo di vita dell'istanza

```
{
  "version": "0",
  "id": "b6ba298a-7732-2226-xmpl-976312c1a050",
  "detail-type": "EC2 Instance State-change Notification",
  "source": "aws.ec2",
  "account": "111122223333",
  "time": "2019-10-02T17:59:30Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ec2:us-east-1:111122223333:instance/i-0c314xmplcd5b8173"
  ],
  "detail": {
    "instance-id": "i-0c314xmplcd5b8173",
    "state": "running"
  }
}
```

Per informazioni dettagliate sulla configurazione degli eventi, consulta [Richiamo di una funzione Lambda su una pianificazione](#). Per un esempio di funzione che elabora le notifiche istantanee di Amazon EBS, consulta Scheduler for Amazon EBS. [EventBridge](#)

Puoi anche utilizzare l' AWS SDK per gestire istanze e altre risorse con l'API Amazon EC2.



## Concessione delle autorizzazioni a (Eventi) EventBridge CloudWatch

Per elaborare gli eventi del ciclo di vita da Amazon EC2, EventBridge (CloudWatch Events) necessita dell'autorizzazione per richiamare la funzione. Questa autorizzazione proviene dalla [policy basata sulle risorse](#) della funzione. Se utilizzi la console EventBridge (CloudWatch Events) per configurare un trigger di eventi, la console aggiorna la policy basata sulle risorse per tuo conto. In caso contrario, aggiungere un'istruzione come la seguente:

Example Dichiarazione della policy basata sulle risorse per le notifiche del ciclo di vita di Amazon EC2

```
{
  "Sid": "ec2-events",
  "Effect": "Allow",
  "Principal": {
    "Service": "events.amazonaws.com"
  },
  "Action": "lambda:InvokeFunction",
  "Resource": "arn:aws:lambda:us-east-1:12456789012:function:my-function",
  "Condition": {
    "ArnLike": {
      "AWS:SourceArn": "arn:aws:events:us-east-1:12456789012:rule/*"
    }
  }
}
```

Per aggiungere una dichiarazione, usa il comando. add-permission AWS CLI

```
aws lambda add-permission --action lambda:InvokeFunction --statement-id ec2-events \
--principal events.amazonaws.com --function-name my-function --source-arn
'arn:aws:events:us-east-1:12456789012:rule/*'
```

Se la tua funzione utilizza l' AWS SDK per gestire le risorse Amazon EC2, aggiungi le autorizzazioni Amazon EC2 al ruolo di esecuzione della funzione. [???](#)

## Elaborare le richieste di Application Load Balancer con Lambda

È possibile utilizzare una funzione Lambda per elaborare le richieste di un Application Load Balancer. Elastic Load Balancing supporta le funzioni Lambda come target per un Application Load Balancer. Utilizza le regole per il sistema di bilanciamento del carico per indirizzare le richieste HTTP a una funzione in base ai valori del percorso o dell'intestazione. Elabora la richiesta e restituisce una risposta HTTP dalla funzione Lambda.

Elastic Load Balancing richiama la funzione Lambda in modo sincrono con un evento contenente il corpo della richiesta e i metadati.

### Example Evento di richiesta Application Load Balancer

```
{
  "requestContext": {
    "elb": {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/lambda-279XGJDqGZ5rsrHC2Fjr/49e9d65c45c6791a"
    }
  },
  "httpMethod": "GET",
  "path": "/lambda",
  "queryStringParameters": {
    "query": "1234ABCD"
  },
  "headers": {
    "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "accept-encoding": "gzip",
    "accept-language": "en-US,en;q=0.9",
    "connection": "keep-alive",
    "host": "lambda-alb-123578498.us-east-1.elb.amazonaws.com",
    "upgrade-insecure-requests": "1",
    "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36",
    "x-amzn-trace-id": "Root=1-5c536348-3d683b8b04734faae651f476",
    "x-forwarded-for": "72.12.164.125",
    "x-forwarded-port": "80",
    "x-forwarded-proto": "http",
    "x-imforwards": "20"
  },
  "body": "",
  "isBase64Encoded": False
}
```

```
}
```

La funzione elabora l'evento e restituisce un documento di risposta al sistema di bilanciamento del carico in JSON. Elastic Load Balancing converte il documento in una risposta HTTP di esito positivo o negativo e la restituisce all'utente.

#### Example Formato del documento di risposta

```
{
  "statusCode": 200,
  "statusDescription": "200 OK",
  "isBase64Encoded": False,
  "headers": {
    "Content-Type": "text/html"
  },
  "body": "<h1>Hello from Lambda!</h1>"
}
```

Per configurare un Application Load Balancer come trigger di funzione, concedi l'autorizzazione Elastic Load Balancing per eseguire la funzione, crea un gruppo di destinazioni che instrada le richieste alla funzione e aggiungi una regola al sistema di bilanciamento del carico che invia le richieste al gruppo di destinazioni.

Utilizza il comando `add-permission` per aggiungere un'istruzione di autorizzazione alla policy basata sulle risorse della funzione.

```
aws lambda add-permission --function-name alb-function \
--statement-id load-balancer --action "lambda:InvokeFunction" \
--principal elasticloadbalancing.amazonaws.com
```

Verrà visualizzato l'output seguente:

```
{
  "Statement": "{\"Sid\": \"load-balancer\", \"Effect\": \"Allow\", \"Principal\": \"Service\": \"elasticloadbalancing.amazonaws.com\", \"Action\": \"lambda:InvokeFunction\", \"Resource\": \"arn:aws:lambda:us-west-2:123456789012:function:alb-function\"}"
}
```

Per istruzioni su come configurare il gruppo di destinazione e il listener di Application Load Balancer, consulta [funzioni Lambda come target](#) nella Guida per l'utente dei sistemi Application Load Balancer.

## Gestore di eventi di Powertools per AWS Lambda

Il gestore di eventi del toolkit Powertools for AWS Lambda fornisce routing, middleware, configurazione CORS, generazione di specifiche OpenAPI, convalida delle richieste, gestione degli errori e altre utili funzionalità per la scrittura di funzioni Lambda richiamate da un Application Load Balancer. L'utilità Event Handler è disponibile per Python. Per ulteriori informazioni, vedere [Event Handler REST API](#) nella documentazione di Powertools for AWS Lambda (Python).

### Python

```
import requests
from requests import Response

from aws_lambda_powertools import Logger, Tracer
from aws_lambda_powertools.event_handler import ALBResolver
from aws_lambda_powertools.logging import correlation_paths
from aws_lambda_powertools.utilities.typing import LambdaContext

tracer = Tracer()
logger = Logger()
app = ALBResolver()

@app.get("/todos")
@tracer.capture_method
def get_todos():
    todos: Response = requests.get("https://jsonplaceholder.typicode.com/todos")
    todos.raise_for_status()

    # for brevity, we'll limit to the first 10 only
    return {"todos": todos.json()[:10]}

# You can continue to use other utilities just as before
@logger.inject_lambda_context(correlation_id_path=correlation_paths.APPLICATION_LOAD_BALANCER)
@tracer.capture_lambda_handler
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    return app.resolve(event, context)
```

## Richiamo di una funzione Lambda su una pianificazione

[Amazon EventBridge Scheduler](#) è uno scheduler serverless che puoi utilizzare per creare, eseguire e gestire attività da un unico servizio gestito centralizzato. Con EventBridge Scheduler, puoi creare pianificazioni utilizzando espressioni cron e rate per modelli ricorrenti o configurare invocazioni una tantum. Puoi configurare finestre temporali flessibili per la consegna, definire limiti per nuovi tentativi e impostare il tempo massimo di conservazione per eventi non elaborati.

Quando configuri Scheduler con Lambda, EventBridge Scheduler richiama la funzione Lambda in modo asincrono. EventBridge Puoi creare una pianificazione dalla console Lambda (utilizzando il flusso di trigger Add), dalla console Scheduler, da, o. EventBridge AWS CLI CloudFormation Questa pagina spiega come utilizzare EventBridge Scheduler per richiamare una funzione Lambda su una pianificazione.

### Configurare il ruolo di esecuzione

Quando crei una nuova EventBridge pianificazione, Scheduler deve disporre dell'autorizzazione per richiamare l'operazione dell'API di destinazione per tuo conto. Concedi queste autorizzazioni a EventBridge Scheduler utilizzando un ruolo di esecuzione. La policy di autorizzazione collegata al ruolo di esecuzione della pianificazione definisce le autorizzazioni necessarie. Queste autorizzazioni dipendono dall'API di destinazione che desideri che EventBridge Scheduler richiami.

Quando crei una pianificazione utilizzando la console Lambda o la console EventBridge Scheduler, EventBridge Scheduler imposta automaticamente un ruolo di esecuzione in base al target selezionato. Se desideri creare una pianificazione utilizzando uno degli SDK EventBridge Scheduler, o AWS CLI CloudFormation, devi disporre di un ruolo di esecuzione esistente che conceda le autorizzazioni necessarie EventBridge allo Scheduler per richiamare un target. Per ulteriori informazioni sull'impostazione manuale di un ruolo di esecuzione per la pianificazione, vedere [Impostazione di un ruolo di esecuzione nella Guida per l'utente di Scheduler](#). EventBridge

### Crea una pianificazione dalla console Lambda

Usa il flusso di trigger Add nella console Lambda per creare una EventBridge pianificazione Scheduler senza uscire dalla console.

Per creare un trigger Scheduler utilizzando la console Lambda

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione che desideri programmare.

3. Nel riquadro Panoramica della funzione, scegli Aggiungi trigger.
4. Dal menu a discesa della fonte del trigger, scegli Scheduler.
5. Configura le impostazioni di pianificazione:
  - a. Nome del programma: inserisci un nome per il tuo programma.
  - b. Gruppo di pianificazione: scegli un gruppo di pianificazione o utilizza `default`.
  - c. Tipo di pianificazione: scegli una delle seguenti opzioni:
    - One-time pianificazione: richiama la funzione una volta alla data e all'ora specificate.
    - Pianificazione ricorrente: richiama la funzione ripetutamente utilizzando un'espressione cron o rate.
  - d. (Facoltativo) Inserisci un payload JSON. Se non si inserisce un payload, EventBridge Scheduler utilizza un evento vuoto per richiamare la funzione.
  - e. Configura il ruolo di esecuzione. Per fare in modo che EventBridge Scheduler crei un nuovo ruolo di esecuzione per te, scegli Crea nuovo ruolo per questa pianificazione. Inserisci, quindi, un nome per Nome ruolo. Se scegli questa opzione, EventBridge Scheduler crea una policy con le autorizzazioni richieste e la allega al ruolo.
6. Scegliere Aggiungi.

 Tip

Puoi anche cercare **schedule**, **crontime**, o per **recurring** trovare questo trigger nel flusso Aggiungi trigger.

Puoi anche utilizzare la console [EventBridge Scheduler](#) per creare una pianificazione. Con la console EventBridge Scheduler, puoi configurare opzioni aggiuntive come finestre temporali flessibili e code per le lettere morte.

## Visualizzazione EventBridge dei trigger di Scheduler

Tutte le EventBridge pianificazioni di Scheduler destinate alla tua funzione vengono visualizzate nella sezione Trigger della pagina di configurazione della funzione sotto l'etichetta Scheduler. Ciò include le pianificazioni create da una delle seguenti fonti:

- La console Lambda (Aggiungi flusso di trigger)

- La console EventBridge Scheduler
- L' AWS CLI SDK, CloudFormation, AWS SAM (utilizzando il tipo di `ScheduleV2` evento) o AWS CDK (utilizzando il `aws-scheduler` modulo)

#### Note

Se una pianificazione utilizza una chiave gestita dal cliente per la crittografia con un target Universal Target Invocation (UTI), potrebbe non apparire nell'elenco dei trigger. Per visualizzare queste pianificazioni, utilizza la console Scheduler [EventBridge](#) .

Per confermare che EventBridge Scheduler abbia richiamato la funzione, [controlla i log Amazon della funzione. CloudWatch](#)

## Risorse correlate

Per ulteriori informazioni su EventBridge Scheduler, consulta quanto segue:

- [EventBridge Guida per l'utente di Scheduler](#)
- [EventBridge Riferimento all'API Scheduler](#)
- [EventBridge Prezzi Scheduler](#)

## Utilizzo AWS Lambda con AWS IoT

AWS IoT fornisce una comunicazione sicura tra i dispositivi connessi a Internet (come i sensori) e il cloud. AWS Questo consente raccogliere, archiviare e analizzare i dati di telemetria da più dispositivi.

Puoi creare AWS IoT regole con cui i tuoi dispositivi possono interagire. Servizi AWS Il AWS IoT [Rules Engine](#) fornisce un SQL-based linguaggio per selezionare i dati dai payload dei messaggi e inviarli ad altri servizi, come Amazon S3, Amazon DynamoDB e. AWS Lambda Si definisce una regola per richiamare una funzione Lambda quando si desidera richiamare un AWS altro servizio o un servizio di terze parti.

Quando un messaggio IoT in arrivo attiva la regola, AWS IoT richiama la funzione Lambda in modo asincrono e passa i dati dal messaggio IoT alla [funzione](#).

L'esempio seguente mostra una lettura dell'umidità da un sensore serra. I valori di riga e pos identificano la posizione del sensore. Questo evento di esempio si basa sul tipo serra in [Tutorial sulle regole AWS IoT](#).

Example AWS IoT evento di messaggio

```
{
  "row" : "10",
  "pos" : "23",
  "moisture" : "75"
}
```

Per le chiamate asincrone, Lambda inserisce in una coda i messaggi e i [tentativi](#) se la funzione restituisce un errore. Configura la tua funzione con una [destination](#) per mantenere gli eventi che la tua funzione non è in grado di elaborare.

È necessario concedere l'autorizzazione al AWS IoT servizio per richiamare la funzione Lambda. Utilizza il comando `add-permission` per aggiungere un'istruzione di autorizzazione alla policy basata sulle risorse della funzione.

### Important

Se aggiungi un'autorizzazione per un' AWS IoT entità principale senza fornire `source-arn` o `source-account`, chiunque Account AWS crei una regola con la tua AWS Lambda azione



può attivare regole da cui richiamare la tua AWS Lambda funzione. AWS IoT Per ulteriori informazioni, consulta [Autorizzazioni di AWS Lambda](#).

```
aws lambda add-permission --function-name my-function \  
--statement-id iot-events --action "lambda:InvokeFunction" --principal  
iot.amazonaws.com \  
--source-arn arn:aws:iot:us-east-1:123456789012:rule/my-rule \  
--source-account 123456789012
```

Verrà visualizzato l'output seguente:

```
{  
  "Statement": "{\\"Sid\\":\\"iot-events\\",\\"Effect\\":\\"Allow\\",\\"Principal\\":  
{\\"Service\\":\\"iot.amazonaws.com\\"},\\"Action\\":\\"lambda:InvokeFunction\\",\\"Resource  
\\":\\"arn:aws:lambda:us-east-1:123456789012:function:my-function\\",\\"Condition\\":  
{\\"ArnLike\\":{\\"AWS:SourceArn\\":\\"arn:aws:iot:us-east-1:123456789012:rule/my-rule\\"},  
\\"StringEquals\\":{\\"AWS:SourceAccount\\":\\"123456789012\\"}}}"  
}
```

Per ulteriori informazioni su come usare Lambda con AWS IoT, vedi [Creazione di una AWS Lambda regola](#).

# Utilizzo di Lambda per elaborare i record da Amazon Kinesis Data Streams

È possibile usare una funzione Lambda per elaborare i record in un [flusso di dati Amazon Kinesis](#). È possibile mappare una funzione Lambda a un consumer a throughput condiviso (iteratore standard) del flusso di dati Kinesis o a un consumer di throughput dedicato con [fan-out avanzato](#). Per gli iteratori standard, Lambda interroga ogni shard nel flusso Kinesis alla ricerca di record utilizzando il protocollo HTTP. La mappatura dell'origine eventi condivide il throughput di lettura con altri utenti dello shard.

Per informazioni dettagliate sui flussi di dati Kinesis, consulta [Lettura dei dati da Flusso di dati Amazon Kinesis](#).

Per iniziare, consulta [the section called “Creare lo strumento di mappatura”](#). Puoi configurare i parametri di mappatura delle sorgenti [degli eventi](#), utilizzare [finestre scorrevoli](#) per l'aggregazione e visualizzare un esempio di codice funzionale. [???](#)

## Note

Kinesis addebita dei costi per ogni shard, per il fan-out avanzato e per i dati letti dal flusso. Per i dettagli sui prezzi, consulta [Prezzi di Amazon Kinesis](#).

## Flussi di polling e batching

Lambda legge i record dal flusso di dati e richiama la funzione [in modo sincrono](#) con un evento che contiene record di flusso. Lambda legge i record in batch e richiama la funzione per elaborare i record dal batch. Ogni batch contiene i record di un singolo stream. shard/data

La funzione Lambda è un'applicazione consumer per il flusso di dati. Elabora un batch di record alla volta da ciascuno shard. È possibile mappare una funzione Lambda a un consumer a throughput condiviso (iteratore standard) o a un consumer di throughput dedicato con fan-out avanzato.

- **Iteratori standard:** Lambda esegue il polling per ogni shard nel flusso Kinesis per i record a una velocità di base di una volta al secondo. Quando sono disponibili più record, Lambda continua l'elaborazione dei batch fino a quando la funzione raggiunge il flusso. La mappatura dell'origine eventi condivide il throughput di lettura con altri utenti dello shard.
- **Fan-out avanzato:** per ridurre al minimo la latenza e massimizzare il throughput di lettura, è necessario creare un consumer del flusso di dati con [fan-out avanzato](#). I consumatori del fan-out

avanzato ottengono una connessione dedicata a ciascuno shard che non ha conseguenze su altre applicazioni che leggono dal flusso. Gli utenti di streaming utilizzano lo streaming HTTP/2 per ridurre la latenza inviando i record a Lambda tramite una connessione di lunga durata e comprimendo le intestazioni delle richieste. Puoi creare uno stream consumer con l'API Kinesis.

[RegisterStreamConsumer](#)

```
aws kinesis register-stream-consumer \  
--consumer-name con1 \  
--stream-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
```

Verrà visualizzato l'output seguente:

```
{  
  "Consumer": {  
    "ConsumerName": "con1",  
    "ConsumerARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream/  
consumer/con1:1540591608",  
    "ConsumerStatus": "CREATING",  
    "ConsumerCreationTimestamp": 1540591608.0  
  }  
}
```

Per aumentare la velocità con cui la funzione elabora i record, [aggiungere gli shard al flusso di dati](#). Lambda elabora i record in ogni shard in ordine. Interrompe l'elaborazione di record aggiuntivi in uno shard se la funzione restituisce un errore. Con più partizioni, vengono elaborati più batch contemporaneamente e l'impatto di errori in simultanea viene ridotto.

Se la funzione non è in grado di aumentare le dimensioni fino a gestire il numero totale di batch simultanei, [richiedere un aumento della quota](#) o [riservare la simultaneità](#) per la funzione.

Per impostazione predefinita, Lambda richiama la funzione non appena i record sono disponibili. Se il batch che Lambda legge dall'origine eventi contiene un solo record, Lambda invia solo un record alla funzione. Per evitare di richiamare la funzione con pochi record è possibile, configurando un periodo di batch, chiedere all'origine eventi di memorizzare nel buffer i registri per un massimo di 5 minuti. Prima di richiamare la funzione, Lambda continua a leggere i registri dall'origine eventi fino a quando non ha raccolto un batch completo, fino alla scadenza del periodo di batch o fino a quando il batch non ha raggiunto il limite del payload di 6 MB. Per ulteriori informazioni, consulta [Comportamento di batching](#).

**⚠ Warning**

Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per saperne di più, consulta [How do I make my Lambda function idempotent](#) nel Knowledge Center. AWS

Lambda non attende il completamento di alcuna [estensione](#) prima di inviare il batch successivo per l'elaborazione. In altre parole, le estensioni possono continuare a funzionare mentre Lambda elabora il successivo batch di record. Ciò può causare problemi di limitazione in caso di violazione delle impostazioni o dei limiti di [simultaneità](#). Per rilevare se si tratta di un potenziale problema, monitora le tue funzioni e verifica se i [parametri di simultaneità](#) per lo strumento di mappatura dell'origine degli eventi sono superiori al previsto. A causa degli intervalli ridotti tra le invocazioni, Lambda potrebbe segnalare brevemente un utilizzo della simultaneità maggiore rispetto al numero di partizioni. Ciò può essere vero anche per le funzioni Lambda senza estensioni.

Configura l'[ParallelizationFactor](#) impostazione per elaborare un frammento di un flusso di dati Kinesis con più di una chiamata Lambda contemporaneamente. È possibile specificare il numero di batch simultanei che Lambda esegue il polling da uno shard utilizzando un fattore di parallelizzazione compreso tra 1 (predefinito) e 10. Ad esempio, se `ParallelizationFactor` imposti su 2, puoi avere un massimo di 200 chiamate Lambda simultanee per elaborare 100 frammenti di dati Kinesis (anche se in pratica potresti vedere valori diversi per la metrica). `ConcurrentExecutions` Ciò permette di dimensionare verso l'alto il throughput di elaborazione quando il volume dei dati è volatile e l'`IteratorAge` è alta. Se si aumenta il numero di batch simultanei per shard, Lambda garantisce comunque l'ordine di elaborazione a livello di partizione-chiave.

Puoi anche utilizzare `ParallelizationFactor` con l'aggregazione Kinesis. Il comportamento dello strumento di mappatura dell'origine degli eventi dipende dall'utilizzo o meno di un [fan-out avanzato](#):

- Senza fan-out avanzato: tutti gli eventi all'interno di un evento aggregato devono avere la stessa chiave di partizione. La chiave di partizione deve inoltre corrispondere a quella dell'evento aggregato. Se gli eventi all'interno dell'evento aggregato hanno chiavi di partizione diverse, Lambda non può garantire l'elaborazione ordinata degli eventi per chiave di partizione.
- Con fan-out migliorato: innanzitutto, Lambda decodifica l'evento aggregato nei suoi singoli eventi. L'evento aggregato può avere una chiave di partizione diversa dagli eventi che contiene. Tuttavia,

gli eventi che non corrispondono alla chiave di partizione vengono [eliminati e persi](#). Lambda non elabora questi eventi e non li invia a una destinazione di errore configurata.

## Esempio di evento

### Example

```
{
  "Records": [
    {
      "kinesis": {
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber":
"49590338271490256608559692538361571095921575989136588898",
        "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
        "approximateArrivalTimestamp": 1545084650.987
      },
      "eventSource": "aws:kinesis",
      "eventVersion": "1.0",
      "eventID":
"shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
      "eventName": "aws:kinesis:record",
      "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
      "awsRegion": "us-east-2",
      "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-
stream"
    },
    {
      "kinesis": {
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber":
"49590338271490256608559692540925702759324208523137515618",
        "data": "VGhpcyBpcyBvbmx5IGVgdGVzdC4=",
        "approximateArrivalTimestamp": 1545084711.166
      },
      "eventSource": "aws:kinesis",
      "eventVersion": "1.0",
      "eventID":
"shardId-000000000006:49590338271490256608559692540925702759324208523137515618",
      "eventName": "aws:kinesis:record",
      "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
```

```
        "awsRegion": "us-east-2",
        "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-
stream"
    }
  ]
}
```

## Elaborare i record del flusso di dati Amazon Kinesis con Lambda

Per elaborare i record di Amazon Kinesis Data Streams con Lambda, crea una mappatura dell'origine degli eventi Lambda. Puoi mappare una funzione Lambda su un iteratore standard o su un consumer fan-out avanzato. Per ulteriori informazioni, consulta [Flussi di polling e batching](#).

Crea una mappatura della sorgente degli eventi Kinesis

Per richiamare la funzione Lambda con i record dal flusso di dati, crea uno [strumento di mappatura dell'origine degli eventi](#). È possibile creare più mappature delle origini eventi per elaborare gli stessi dati con più funzioni Lambda o per elaborare elementi da più flussi di dati con una singola funzione. Quando si elaborano elementi da più flussi, ogni batch conterrà i record di un solo shard o di un solo flusso.

È possibile configurare gli strumenti di mappatura dell'origine degli eventi per elaborare i record da un flusso in un altro Account AWS. Per ulteriori informazioni, consulta [the section called “Cross-account mappature”](#).

Prima di creare uno strumento di mappatura dell'origine degli eventi, devi autorizzare la funzione Lambda a leggere da un flusso di dati Kinesis. Lambda richiede le seguenti autorizzazioni per gestire le risorse correlate al flusso di dati Kinesis:

- [kinesis: DescribeStream](#)
- [cinesi: DescribeStreamSummary](#)
- [cinesi: GetRecords](#)
- [cinesi: GetShardIterator](#)
- [cinesi: ListShards](#)
- [cinesi: SubscribeToShard](#)

La policy AWS gestita [AWSLambdaKinesisExecutionRole](#) include queste autorizzazioni. Aggiungi questa policy gestita alla funzione come descritto nella seguente procedura.

 Note

- Non è necessaria l'`kinesis:ListStreams` autorizzazione per creare e gestire le mappature delle sorgenti degli eventi per Kinesis. Tuttavia, se crei una mappatura delle sorgenti di eventi nella console e non disponi di questa autorizzazione, non sarai in grado di selezionare uno stream Kinesis da un elenco a discesa e la console visualizzerà un errore. Per creare la mappatura della sorgente degli eventi, dovrai inserire manualmente l'Amazon Resource Name (ARN) del tuo stream.
- Lambda effettua `kinesis:GetRecords` una chiamata `kinesis:GetShardIterator` API quando si riprovano le chiamate non riuscite.

## Console di gestione AWS

Per aggiungere le autorizzazioni Kinesis alla tua funzione

1. Apri la [pagina Funzioni](#) della console Lambda e scegli la tua funzione.
2. Nella scheda Configurazione scegli Autorizzazioni.
3. Nel riquadro del ruolo di esecuzione, in Nome del ruolo, scegli il link al ruolo di esecuzione della tua funzione. Questo link apre la pagina del ruolo nella console IAM.
4. Nel riquadro Policy di autorizzazioni, seleziona Aggiungi autorizzazioni, quindi Collega policy.
5. Inserisci **AWSLambdaKinesisExecutionRole** nel campo di ricerca.
6. Seleziona la casella di controllo accanto al nome della policy, quindi scegli Aggiungi autorizzazione.

## AWS CLI

Per aggiungere le autorizzazioni Kinesis alla tua funzione

- Esegui il seguente comando CLI per aggiungere la `AWSLambdaKinesisExecutionRole` policy al ruolo di esecuzione della tua funzione:

```
aws iam attach-role-policy \  
--role-name MyFunctionRole \  
--policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaKinesisExecutionRole
```

## AWS SAM

Per aggiungere le autorizzazioni Kinesis alla tua funzione

- Nella definizione della funzione, aggiungete la `Policies` proprietà come mostrato nell'esempio seguente:

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./my-function/
      Handler: index.handler
      Runtime: nodejs24.x
      Policies:
        - AWSLambdaKinesisExecutionRole
```

Dopo aver configurato le autorizzazioni richieste, crea lo strumento di mappatura dell'origine degli eventi.

### Console di gestione AWS

Per creare uno strumento di mappatura dell'origine degli eventi Kinesis

1. Apri la [pagina Funzioni](#) della console Lambda e scegli la tua funzione.
2. Nel riquadro Panoramica della funzione, scegli Aggiungi trigger.
3. In Configurazione del trigger, per l'origine seleziona Kinesis.
4. Seleziona il flusso Kinesis per il quale desideri creare lo strumento di mappatura dell'origine degli eventi e, facoltativamente, un consumer del tuo flusso.
5. (Facoltativo) Modifica la dimensione del batch, la posizione iniziale e la finestra batch per lo strumento di mappatura dell'origine degli eventi.
6. Scegliere Aggiungi.

Quando crei la mappatura della sorgente degli eventi dalla console, il tuo ruolo IAM deve avere le autorizzazioni [kinesis: ListStreams](#) e [kinesis: ListStreamConsumers](#)



## AWS CLI

Per creare uno strumento di mappatura dell'origine degli eventi Kinesis

- Per creare uno strumento di mappatura dell'origine degli eventi Kinesis, eseguire il comando della CLI sotto riportato. Scegli la dimensione del batch e la posizione iniziale in base al tuo caso d'uso.

```
aws lambda create-event-source-mapping \  
--function-name MyFunction \  
--event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream \  
--starting-position LATEST \  
--batch-size 100
```

Per specificare una finestra di batch, aggiungi l'opzione `--maximum-batching-window-in-seconds`. Per ulteriori informazioni sull'utilizzo di questo e di altri parametri, consulta [create-event-source-mapping](#) in Riferimento ai comandi della AWS CLI .

## AWS SAM

Per creare uno strumento di mappatura dell'origine degli eventi Kinesis

- Nella definizione della funzione, aggiungete la `KinesisEvent` proprietà come mostrato nell'esempio seguente:

```
Resources:  
  MyFunction:  
    Type: AWS::Serverless::Function  
    Properties:  
      CodeUri: ./my-function/  
      Handler: index.handler  
      Runtime: nodejs24.x  
      Policies:  
        - AWSLambdaKinesisExecutionRole  
      Events:  
        KinesisEvent:  
          Type: Kinesis  
          Properties:  
            Stream: !GetAtt MyKinesisStream.Arn  
            StartingPosition: LATEST  
            BatchSize: 100
```

```
MyKinesisStream:
  Type: AWS::Kinesis::Stream
  Properties:
    ShardCount: 1
```

Per saperne di più sulla creazione di una mappatura delle sorgenti di eventi per Kinesis Data Streams in AWS SAM, consulta [Kinesis](#) nella Guida per gli sviluppatori. AWS Serverless Application Model

## Posizioni di partenza di polling e flussi

Tieni presente che il polling dei flussi durante la creazione e gli aggiornamenti dello strumento di mappatura dell'origine degli eventi alla fine è coerente.

- Durante la creazione dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.
- Durante gli aggiornamenti dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.

Questo comportamento implica che se specifichi LATEST come posizione iniziale del flusso, lo strumento di mappatura dell'origine degli eventi potrebbe perdere eventi durante la creazione o gli aggiornamenti. Per non perdere alcun evento, specifica la posizione iniziale del flusso come TRIM\_HORIZON o AT\_TIMESTAMP.

## Creazione di uno strumento di mappatura dell'origine degli eventi multi-account

Flusso di dati Amazon Kinesis supporta le [policy basate sulle risorse](#). Per questo motivo, puoi elaborare i dati inseriti in uno stream in uno Account AWS con una funzione Lambda in un altro account.

Per creare una mappatura dell'origine degli eventi per la tua funzione Lambda utilizzando uno stream Kinesis in un altro Account AWS, devi configurare lo stream utilizzando una policy basata sulle risorse per autorizzare la funzione Lambda a leggere gli elementi. Per informazioni su come configurare lo stream per consentire l'accesso con più account, consulta [Condivisione dell'accesso con funzioni tra account nella guida per sviluppatori di Amazon Kinesis Streams AWS Lambda](#).

Dopo aver configurato lo stream con una policy basata sulle risorse che conferisce alla funzione Lambda le autorizzazioni richieste, crea la mappatura della sorgente degli eventi utilizzando uno dei metodi descritti nella sezione precedente.

Se scegli di creare lo strumento di mappatura dell'origine degli eventi utilizzando la console Lambda, incolla l'ARN del tuo flusso direttamente nel campo di input. Se desideri specificare un consumer per il tuo flusso, incollando l'ARN del consumer viene compilato automaticamente il campo del flusso.

## Configurazione della risposta batch parziale con il flusso di dati Kinesis e Lambda

Quando si consumano ed elaborano i dati di streaming da un'origine eventi, per impostazione predefinita i Lambda imposta i checkpoint al numero di sequenza più alto di un batch solo quando il batch è riuscito completamente. Lambda tratta tutti gli altri risultati come un fallimento completo e riprova a elaborare il batch fino al limite di tentativi. Per consentire i successi parziali durante l'elaborazione di batch da un flusso, attivare `ReportBatchItemFailures`. Consentire i successi parziali può aiutare a ridurre il numero di tentativi su un record, anche se non previene completamente la possibilità che si verifichino nuovi tentativi in un record riuscito.

Per attivare `ReportBatchItemFailures`, includere il valore enum **`ReportBatchItemFailures`** nell'[FunctionResponseTypes](#) elenco. Questo elenco indica quali tipi di risposta sono abilitati per la funzione. È possibile configurare questo elenco quando si [crea](#) o si [aggiorna](#) uno strumento di mappatura dell'origine degli eventi.

### Note

Anche quando il codice della funzione restituisce risposte parziali di errore in batch, queste risposte non vengono elaborate da Lambda a meno che la `ReportBatchItemFailures` funzionalità non sia esplicitamente attivata per la mappatura dell'origine dell'evento.

## Sintassi di report

Quando si configura la creazione di report sugli errori degli elementi batch, la `StreamsEventResponse` classe viene restituita con un elenco di errori degli articoli batch. È possibile utilizzare un `StreamsEventResponse` oggetto per restituire il numero di sequenza del primo record non riuscito nel batch. È inoltre possibile creare la propria classe personalizzata utilizzando la sintassi di risposta corretta. La seguente struttura JSON mostra la sintassi di risposta richiesta:

```
{
  "batchItemFailures": [
    {
      "itemIdentifier": "<SequenceNumber>"
    }
  ]
}
```

### Note

Se l'array `batchItemFailures` contiene più elementi, Lambda utilizza il record con il numero di sequenza più basso come checkpoint. Lambda quindi riprova tutti i record a partire da quel checkpoint.

## Condizioni di successo e di errore

Lambda considera un batch come un successo completo se si restituisce uno degli elementi seguenti:

- Una `batchItemFailure` lista vuota
- Un `batchItemFailure` elenco nullo
- Un vuoto `EventResponse`
- Un valore nullo `EventResponse`

Lambda considera un batch come un fallimento completo se si restituisce uno degli elementi seguenti:

- Una stringa vuota `itemIdentifier`
- Un valore nullo `itemIdentifier`
- Un `itemIdentifier` con un nome chiave errato

Lambda esegue nuovi tentativi in seguito ai fallimenti secondo la strategia di tentativi impostata.

## Bisezione un batch

Se il richiamo fallisce ed `BisectBatchOnFunctionError` è attivato, il batch viene bisecato a prescindere dalle `ReportBatchItemFailures` impostazioni.

Quando si riceve una risposta di successo parziale del batch ed entrambi `BisectBatchOnFunctionError` e `ReportBatchItemFailures` sono attivati, il batch viene bisecato in corrispondenza del numero di sequenza restituito e Lambda ritenta solo i record rimanenti.

Per semplificare l'implementazione della logica di risposta batch parziale, prendi in considerazione l'utilizzo dell'utilità [Batch Processor](#) di Powertools for AWS Lambda, che gestisce automaticamente queste complessità per te.

Di seguito sono riportati alcuni esempi di codice di funzione che restituiscono l'elenco degli ID dei messaggi con errori nel batch:

## .NET

### SDK per .NET

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
```

```

// POWERTOOLS_SERVICE_NAME
[Logging(LogEvent = true)]
public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
ILambdaContext context)
{
    if (evnt.Records.Count == 0)
    {
        Logger.LogInformation("Empty Kinesis Event received");
        return new StreamsEventResponse();
    }

    foreach (var record in evnt.Records)
    {
        try
        {
            Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
            string data = await GetRecordDataAsync(record.Kinesis, context);
            Logger.LogInformation($"Data: {data}");
            // TODO: Do interesting work based on the new data
        }
        catch (Exception ex)
        {
            Logger.LogError($"An error occurred {ex.Message}");
            /* Since we are working with streams, we can return the failed
item immediately.
            Lambda will immediately begin to retry processing from this
failed item onwards. */
            return new StreamsEventResponse
            {
                BatchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>
                {
                    new StreamsEventResponse.BatchItemFailure
{ ItemIdentifier = record.Kinesis.SequenceNumber }
                }
            };
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
        return new StreamsEventResponse();
    }
}

```

```

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
        ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}

```

Go

## SDK per Go V2

### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Go.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"

```

```
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
(map[string]interface{}, error) {
    batchItemFailures := []map[string]interface{}{}

    for _, record := range kinesisEvent.Records {
        curRecordSequenceNumber := ""

        // Process your record
        if /* Your record processing condition here */ {
            curRecordSequenceNumber = record.Kinesis.SequenceNumber
        }

        // Add a condition to check if the record processing failed
        if curRecordSequenceNumber != "" {
            batchItemFailures = append(batchItemFailures, map[string]interface{}{
                "itemIdentifier": curRecordSequenceNumber})
        }
    }

    kinesisBatchResponse := map[string]interface{}{
        "batchItemFailures": batchItemFailures,
    }
    return kinesisBatchResponse, nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).



## Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent input, Context
context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
ArrayList<>();
        String curRecordSequenceNumber = "";

        for (KinesisEvent.KinesisEventRecord kinesisEventRecord :
input.getRecords()) {
            try {
                //Process your record
                KinesisEvent.Record kinesisRecord =
kinesisEventRecord.getKinesis();
                curRecordSequenceNumber = kinesisRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
item immediately.
                Lambda will immediately begin to retry processing from this
failed item onwards. */
                batchItemFailures.add(new
StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

        return new StreamsEventResponse(batchItemFailures);
    }
}
```

```
}
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Javascript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
      immediately.
      Lambda will immediately begin to retry processing from this failed
      item onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
```

```

    await Promise.resolve(1); //Placeholder for actual async work
    return data;
}

```

## Segnalazione dei guasti relativi agli articoli del batch Kinesis con Lambda using. TypeScript

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
    KinesisStreamEvent,
    Context,
    KinesisStreamHandler,
    KinesisStreamRecordPayload,
    KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
    logLevel: "INFO",
    serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
    event: KinesisStreamEvent,
    context: Context
): Promise<KinesisStreamBatchResponse> => {
    for (const record of event.Records) {
        try {
            logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
            const recordData = await getRecordDataAsync(record.kinesis);
            logger.info(`Record Data: ${recordData}`);
            // TODO: Do interesting work based on the new data
        } catch (err) {
            logger.error(`An error occurred ${err}`);
            /* Since we are working with streams, we can return the failed item
            immediately.
            Lambda will immediately begin to retry processing from this failed
            item onwards. */
            return {
                batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
            };
        }
    }
}

```

```

    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}

```

## PHP

### SDK per PHP

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite PHP.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{

```

```

private StderrLogger $logger;
public function __construct(StderrLogger $logger)
{
    $this->logger = $logger;
}

/**
 * @throws JsonException
 * @throws \Bref\Event\InvalidLambdaEvent
 */
public function handle(mixed $event, Context $context): array
{
    $kinesisEvent = new KinesisEvent($event);
    $this->logger->info("Processing records");
    $records = $kinesisEvent->getRecords();

    $failedRecords = [];
    foreach ($records as $record) {
        try {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data
        } catch (Exception $e) {
            $this->logger->error($e->getMessage());
            // failed processing the record
            $failedRecords[] = $record->getSequenceNumber();
        }
    }
    $totalRecords = count($records);
    $this->logger->info("Successfully processed $totalRecords records");

    // change format for the response
    $failures = array_map(
        fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
        $failedRecords
    );

    return [
        'batchItemFailures' => $failures
    ];
}

$logger = new StderrLogger();

```

```
return new Handler($logger);
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["kinesis"]["sequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  batch_item_failures = []

  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue StandardError => err
      puts "An error occurred #{err}"
      # Since we are working with streams, we can return the failed item
      # immediately.
      # Lambda will immediately begin to retry processing from this failed item
      # onwards.
      return { batchItemFailures: [{ itemIdentifier: record['kinesis']
['sequenceNumber'] }] }
    end
  end

  puts "Successfully processed #{event['Records'].length} records."
  { batchItemFailures: batch_item_failures }
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('utf-8')
  # Placeholder for actual async work
```

```
sleep(1)
data
end
```

## Rust

### SDK per Rust

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
    Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in &event.payload.records {
        tracing::info!(
            "EventId: {}",
            record.event_id.as_deref().unwrap_or_default()
        );
    }
}
```



```

        let record_processing_result = process_record(record);

        if record_processing_result.is_err() {
            response.batch_item_failures.push(KinesisBatchItemFailure {
                item_identifier: record.kinesis.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
            immediately.
            Lambda will immediately begin to retry processing from this failed
            item onwards. */
            return Ok(response);
        }

        tracing::info!(
            "Successfully processed {} records",
            event.payload.records.len()
        );

        Ok(response)
    }

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)

```

```

        // disabling time is handy because CloudWatch will add the ingestion
        time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```

## Usare Powertools per AWS Lambda processore batch

L'utilità di elaborazione batch di Powertools per la gestione AWS Lambda automatica della logica di risposta parziale ai batch, riducendo la complessità dell'implementazione della segnalazione degli errori dei batch. Ecco alcuni esempi di utilizzo del processore batch:

### Python

#### Note

Per esempi completi e istruzioni di configurazione, consultate la documentazione del processore [batch](#).

Elaborazione dei record di flusso di Kinesis Data Streams con processore AWS Lambda batch.

```

import json
from aws_lambda_powertools import Logger
from aws_lambda_powertools.utilities.batch import BatchProcessor, EventType,
    process_partial_response
from aws_lambda_powertools.utilities.data_classes import KinesisEvent
from aws_lambda_powertools.utilities.typing import LambdaContext

processor = BatchProcessor(event_type=EventType.KinesisDataStreams)
logger = Logger()

def record_handler(record):
    logger.info(record)
    # Your business logic here
    # Raise an exception to mark this record as failed

def lambda_handler(event, context: LambdaContext):
    return process_partial_response(

```

```
    event=event,  
    record_handler=record_handler,  
    processor=processor,  
    context=context  
  )
```

## TypeScript

### Note

Per esempi completi e istruzioni di configurazione, consultate la documentazione del processore [batch](#).

Elaborazione dei record di flusso di Kinesis Data Streams con processore AWS Lambda batch.

```
import { BatchProcessor, EventType, processPartialResponse } from '@aws-lambda-  
powertools/batch';  
import { Logger } from '@aws-lambda-powertools/logger';  
import type { KinesisEvent, Context } from 'aws-lambda';  
  
const processor = new BatchProcessor(EventType.KinesisDataStreams);  
const logger = new Logger();  
  
const recordHandler = async (record: any): Promise<void> => {  
  logger.info('Processing record', { record });  
  // Your business logic here  
  // Throw an error to mark this record as failed  
};  
  
export const handler = async (event: KinesisEvent, context: Context) => {  
  return processPartialResponse(event, recordHandler, processor, {  
    context,  
  });  
};
```

## Java

 Note

Per esempi completi e istruzioni di configurazione, consultate la documentazione del processore [batch](#).

Elaborazione dei record di flusso di Kinesis Data Streams con processore AWS Lambda batch.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import software.amazon.lambda.powertools.batch.BatchMessageHandlerBuilder;
import software.amazon.lambda.powertools.batch.handler.BatchMessageHandler;

public class KinesisStreamBatchHandler implements RequestHandler<KinesisEvent,
StreamsEventResponse> {

    private final BatchMessageHandler<KinesisEvent, StreamsEventResponse> handler;

    public KinesisStreamBatchHandler() {
        handler = new BatchMessageHandlerBuilder()
            .withKinesisBatchHandler()
            .buildWithRawMessageHandler(this::processMessage);
    }

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent kinesisEvent, Context
context) {
        return handler.processBatch(kinesisEvent, context);
    }

    private void processMessage(KinesisEvent.KinesisEventRecord kinesisEventRecord,
Context context) {
        // Process the stream record
    }
}
```

## .NET

### Note

Per esempi completi e istruzioni di configurazione, consultate la documentazione del processore [batch](#).

Elaborazione dei record di flusso di Kinesis Data Streams con processore AWS Lambda batch.

```
using System;
using System.Threading;
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using Amazon.Lambda.Serialization.SystemTextJson;
using AWS.Lambda.Powertools.BatchProcessing;

[assembly: LambdaSerializer(typeof(DefaultLambdaJsonSerializer))]

namespace HelloWorld;

public class OrderEvent
{
    public string? OrderId { get; set; }
    public string? CustomerId { get; set; }
    public decimal Amount { get; set; }
    public DateTime OrderDate { get; set; }
}

internal class TypedKinesisRecordHandler : ITypedRecordHandler<OrderEvent>
{
    public async Task<RecordHandlerResult> HandleAsync(OrderEvent orderEvent,
        CancellationToken cancellation_token)
    {
        if (string.IsNullOrEmpty(orderEvent.OrderId))
        {
            throw new ArgumentException("Order ID is required");
        }

        return await Task.FromResult(RecordHandlerResult.None);
    }
}
```

```
public class Function
{
    [BatchProcessor(TypedRecordHandler = typeof(TypedKinesisRecordHandler))]
    public BatchItemFailuresResponse HandlerUsingTypedAttribute(KinesisEvent _)
    {
        return TypedKinesisStreamBatchProcessor.Result.BatchItemFailuresResponse;
    }
}
```

## Conservare i record batch scartati per un'origine di eventi del flusso di dati Kinesis in Lambda

La gestione degli errori per gli strumenti di mappatura dell'origine degli eventi Kinesis dipende dal fatto che l'errore si verifichi prima che la funzione venga richiamata o durante la chiamata della funzione:

- Prima della chiamata: se una mappatura dell'origine degli eventi Lambda non è in grado di richiamare la funzione a causa di limitazioni o altri problemi, riprova finché i record non scadono o superano l'età massima configurata nella mappatura dell'origine dell'evento (). [MaximumRecordAgeInSeconds](#)
- Durante la chiamata: se la funzione viene richiamata ma restituisce un errore, Lambda riprova fino alla scadenza dei record, al superamento dell'età massima () o al raggiungimento della quota di tentativi configurata ([MaximumRecordAgeInSeconds](#)). [MaximumRetryAttempts](#) Per gli errori di funzione, puoi anche configurare, che divide un batch non riuscito in due batch più piccoli [BisectBatchOnFunctionError](#), isolando i record non validi ed evitando i timeout. La divisione dei batch non consuma la quota di tentativi.

Se le misure di gestione degli errori non riescono, Lambda elimina i record e continua l'elaborazione dei batch dal flusso. Con le impostazioni predefinite, ciò significa che un record errato può bloccare l'elaborazione sullo shard interessata per un massimo di una settimana. Per evitare questa situazione, configura la mappatura dell'origine eventi della funzione con un numero ragionevole di tentativi e un'età massima dei record che sia adatta al caso d'uso.

### Configurazione delle destinazioni per le chiamate non riuscite

Per mantenere i record delle chiamate non riuscite allo strumento di mappatura dell'origine degli eventi, aggiungi una destinazione allo strumento di mappatura dell'origine degli eventi della funzione.

Ogni record inviato alla destinazione è un documento JSON contenente i metadati relativi alla chiamata non riuscita. Per le destinazioni Amazon S3, Lambda invia insieme ai metadati anche l'intero record di invocazione. Puoi configurare qualsiasi argomento Amazon SNS, coda Amazon SQS, bucket Amazon S3 o Kafka come destinazione.

Con le destinazioni Amazon S3, puoi utilizzare la funzionalità [Notifiche eventi Amazon S3](#) per ricevere notifiche quando gli oggetti vengono caricati nel bucket S3 di destinazione. Puoi anche configurare Notifiche eventi S3 per richiamare un'altra funzione Lambda per eseguire l'elaborazione automatica su batch non riusciti.

Il tuo ruolo di esecuzione deve avere le autorizzazioni per la destinazione:

- [Per una destinazione SQS: sqs: SendMessage](#)
- [Per una destinazione SNS: sns:publish](#)
- [Per una destinazione S3: s3: e s3: PutObject ListBucket](#)
- [Per una destinazione Kafka: kafka-cluster: WriteData](#)

È possibile configurare un argomento di Kafka come destinazione in caso di errore per le mappature delle sorgenti degli eventi Kafka. Quando Lambda non è in grado di elaborare i record dopo estenuanti tentativi o quando i record superano l'età massima, Lambda invia i record non riusciti all'argomento Kafka specificato per un'elaborazione successiva. Fai riferimento a [the section called "Destinazione Kafka in caso di errore"](#).

[Se hai abilitato la crittografia con la tua chiave KMS per una destinazione S3, anche il ruolo di esecuzione della funzione deve avere l'autorizzazione a chiamare kms: GenerateDataKey](#) Se la chiave KMS e la destinazione del bucket S3 si trovano in un account diverso dalla funzione Lambda e dal ruolo di esecuzione, configura la chiave KMS in modo che consideri attendibile il ruolo di esecuzione da consentire. kms: GenerateDataKey

Per configurare una destinazione in caso di errore tramite la console, completa i seguenti passaggi:

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. In Function overview (Panoramica delle funzioni), scegliere Add destination (Aggiungi destinazione).
4. Per Origine, scegli Chiamata allo strumento di mappatura dell'origine degli eventi.

5. Per Strumento di mappatura dell'origine degli eventi, scegli un'origine dell'evento configurata per questa funzione.
6. Per Condizione, seleziona In caso di errore. Per le chiamate allo strumento di mappatura dell'origine degli eventi, questa è l'unica condizione accettata.
7. Per Tipo di destinazione, scegli il tipo di destinazione a cui Lambda deve inviare i record di chiamata.
8. Per Destination (Destinazione), scegliere una risorsa.
9. Scegli Save (Salva).

Puoi anche configurare una destinazione in caso di errore utilizzando (). AWS Command Line Interface AWS CLI Ad esempio, il [create-event-source-mapping](#) comando seguente aggiunge una mappatura dell'origine degli eventi con una destinazione SQS in caso di errore a: MyFunction

```
aws lambda create-event-source-mapping \  
--function-name "MyFunction" \  
--event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream \  
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-  
east-1:123456789012:dest-queue"}}'
```

Il [update-event-source-mapping](#) comando seguente aggiorna una mappatura dell'origine degli eventi per inviare i record di chiamata non riuscita a una destinazione SNS dopo due tentativi o se i record risalgono a più di un'ora.

```
aws lambda update-event-source-mapping \  
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \  
--maximum-retry-attempts 2 \  
--maximum-record-age-in-seconds 3600 \  
--destination-config '{"OnFailure": {"Destination": "arn:aws:sns:us-  
east-1:123456789012:dest-topic"}}'
```

Le impostazioni aggiornate sono applicate in modo asincrono e non sono riflesse nell'output fino al completamento del processo. Utilizza il comando [get-event-source-mapping](#) per visualizzare lo stato corrente.

Per rimuovere una destinazione, fornisci una stringa vuota come argomento del parametro `destination-config`:

```
aws lambda update-event-source-mapping \  

```



```
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

## Best practice di sicurezza per destinazioni Amazon S3

L'eliminazione di un bucket S3 configurato come destinazione senza rimuovere la destinazione dalla configurazione della funzione può creare un rischio per la sicurezza. Se un altro utente conosce il nome del bucket di destinazione, può ricreare il bucket nel proprio Account AWS. I record delle invocazioni non riuscite verranno inviati al relativo bucket, esponendo potenzialmente i dati della tua funzione.

### Warning

Per garantire che i record di invocazione della tua funzione non possano essere inviati a un bucket S3 in un altro Account AWS, aggiungi una condizione al ruolo di esecuzione della funzione che limiti le autorizzazioni ai bucket del tuo account. `s3:PutObject`

Di seguito viene illustrato un esempio di policy IAM che limita le autorizzazioni `s3:PutObject` della funzione ai bucket presenti nell'account. Questa policy fornisce inoltre a Lambda l'autorizzazione `s3:ListBucket` necessaria per utilizzare un bucket S3 come destinazione.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "S3BucketResourceAccountWrite",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::*/*",
        "arn:aws:s3:::*"
      ],
      "Condition": {
        "StringEquals": {
          "s3:ResourceAccount": "111122223333"
        }
      }
    }
  ]
}
```

```
}  
]  
}
```

Per aggiungere una politica di autorizzazioni al ruolo di esecuzione della tua funzione utilizzando Console di gestione AWS o AWS CLI, consulta le istruzioni nelle seguenti procedure:

## Console

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Seleziona la funzione Lambda di cui si desidera modificare il ruolo di esecuzione.
3. Nella scheda Configurazione scegli Autorizzazioni.
4. Nella scheda Ruolo di esecuzione, seleziona il nome del ruolo della funzione per aprire la pagina della console IAM del ruolo.
5. Aggiungi una policy di autorizzazioni di base al ruolo completando le seguenti operazioni:
  - a. Nel riquadro Policy di autorizzazioni, scegli Aggiungi autorizzazioni, poi Crea policy in linea.
  - b. Nell'editor delle policy, seleziona JSON.
  - c. Incolla la policy che desideri aggiungere nell'editor (sostituendo il codice JSON esistente), quindi scegli Avanti.
  - d. In Dettagli della policy, specifica un nome per la policy.
  - e. Scegli Crea policy.

## AWS CLI

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (CLI)

1. Crea un documento di policy JSON con le autorizzazioni richieste e salvalo in una directory locale.
2. Utilizza il comando della CLI `put-role-policy` di IAM per aggiungere le autorizzazioni per il ruolo di esecuzione di una funzione. Esegui il comando seguente dalla directory in cui hai salvato il documento di policy JSON e sostituisci il nome del ruolo, il nome della policy e il documento di policy con i tuoi valori.

```
aws iam put-role-policy \  
--role-name my_lambda_role \  
--policy-name LambdaS3DestinationPolicy \  
--policy-document file://my_policy.json
```

## Record di invocazione Amazon SNS e Amazon SQS di esempio

L'esempio seguente mostra ciò che Lambda invia a un argomento SNS o una coda SQS di destinazione per una chiamata non riuscita all'origine dell'evento Kinesis. Poiché Lambda invia solo i metadati per questi tipi di destinazione, utilizza i campi `streamArn`, `shardId`, `startSequenceNumber` e `endSequenceNumber` per ottenere il record originale completo. Tutti i campi mostrati nella proprietà `KinesisBatchInfo` saranno sempre presenti.

```
{  
  "requestContext": {  
    "requestId": "c9b8fa9f-5a7f-xmpl-af9c-0c604cde93a5",  
    "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",  
    "condition": "RetryAttemptsExhausted",  
    "approximateInvokeCount": 1  
  },  
  "responseContext": {  
    "statusCode": 200,  
    "executedVersion": "$LATEST",  
    "functionError": "Unhandled"  
  },  
  "version": "1.0",  
  "timestamp": "2019-11-14T00:38:06.021Z",  
  "KinesisBatchInfo": {  
    "shardId": "shardId-000000000001",  
    "startSequenceNumber":  
"49601189658422359378836298521827638475320189012309704722",  
    "endSequenceNumber":  
"49601189658422359378836298522902373528957594348623495186",  
    "approximateArrivalOfFirstRecord": "2019-11-14T00:38:04.835Z",  
    "approximateArrivalOfLastRecord": "2019-11-14T00:38:05.580Z",  
    "batchSize": 500,  
    "streamArn": "arn:aws:kinesis:us-east-2:123456789012:stream/mystream"  
  }  
}
```

È possibile utilizzare queste informazioni per recuperare i record interessati dal flusso per la risoluzione dei problemi. I record effettivi non sono inclusi, pertanto è necessario elaborare questo record e recuperarli dal flusso prima che scadano e vadano persi.

### Record di invocazione Amazon S3 di esempio

L'esempio seguente mostra ciò che Lambda invia a un bucket Amazon S3 per una chiamata non riuscita all'origine eventi Kinesis. Oltre a tutti i campi dell'esempio precedente per le destinazioni SQS e SNS, il campo `payload` contiene il record di chiamata originale come stringa JSON con escape.

```
{
  "requestContext": {
    "requestId": "c9b8fa9f-5a7f-xmpl-af9c-0c604cde93a5",
    "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",
    "condition": "RetryAttemptsExhausted",
    "approximateInvokeCount": 1
  },
  "responseContext": {
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Unhandled"
  },
  "version": "1.0",
  "timestamp": "2019-11-14T00:38:06.021Z",
  "KinesisBatchInfo": {
    "shardId": "shardId-000000000001",
    "startSequenceNumber":
"49601189658422359378836298521827638475320189012309704722",
    "endSequenceNumber":
"49601189658422359378836298522902373528957594348623495186",
    "approximateArrivalOfFirstRecord": "2019-11-14T00:38:04.835Z",
    "approximateArrivalOfLastRecord": "2019-11-14T00:38:05.580Z",
    "batchSize": 500,
    "streamArn": "arn:aws:kinesis:us-east-2:123456789012:stream/mystream"
  },
  "payload": "<Whole Event>" // Only available in S3
}
```

L'oggetto S3 contenente il record di invocazione utilizza la seguente convenzione di denominazione:

```
aws/lambda/<ESM-UUID>/<shardID>/YYYY/MM/DD/YYYY-MM-DDTHH.MM.SS-<Random UUID>
```

## Implementazione dell'elaborazione di Flusso di dati Kinesis stateful in Lambda

Le funzioni Lambda possono eseguire applicazioni di elaborazione di flussi continui. Un flusso rappresenta dati illimitati che scorrono continuamente attraverso l'applicazione. Per analizzare le informazioni da questo input di aggiornamento continuo, è possibile associare i record inclusi utilizzando una finestra definita in termini di tempo.

Le finestre a cascata sono finestre temporali distinte che si aprono e si chiudono a intervalli regolari. Per impostazione predefinita, le invocazioni Lambda sono senza stato: non è possibile utilizzarle per l'elaborazione dei dati tra più invocazioni continue senza un database esterno. Tuttavia, con la finestra a cascata, è possibile mantenere il proprio stato tra le invocazioni.

Questo stato contiene il risultato aggregato dei messaggi precedentemente elaborati per la finestra corrente. Il tuo stato può avere un massimo di 1 MB per ogni frammento. Se supera quella dimensione, Lambda termina la finestra in anticipo.

Ogni record in un flusso appartiene a una finestra specifica. Lambda elabora ogni record almeno una volta, ma non garantisce che ogni record venga elaborato una sola volta. In rari casi, come nel caso della gestione degli errori, alcuni record potrebbero essere elaborati più di una volta. La prima volta i record vengono sempre elaborati in ordine. Se i record vengono elaborati più di una volta, possono essere elaborati fuori ordine.

### Aggregazione ed elaborazione

La funzione gestita dall'utente viene richiamata sia per l'aggregazione che per l'elaborazione dei risultati finali di tale aggregazione. Lambda aggrega tutti i record ricevuti nella finestra. È possibile ricevere questi record in più batch, ciascuno come richiamo separato. Ogni richiamo riceve uno stato.

Pertanto, quando si utilizzano finestre a cascata, la risposta della funzione Lambda deve contenere una proprietà di `state`. Se la risposta non contiene una proprietà di `state`, Lambda la considera un'invocazione non riuscita. Per soddisfare questa condizione, la funzione può restituire un oggetto `TimeWindowEventResponse`, che presenta la seguente forma JSON:

### Example **TimeWindowEventResponse** valori

```
{
  "state": {
    "1": 282,
    "2": 715
  },
  "batchItemFailures": []
}
```

}

**Note**

Per le funzioni Java, si consiglia di utilizzare una `Map<String, String>` per rappresentare lo stato.

Alla fine della finestra, il flag `isFinalInvokeForWindow` è impostato `true` per indicare che questo è lo stato finale e che è pronto per l'elaborazione. Dopo l'elaborazione, la finestra viene completata e il richiamo finale viene completato e quindi lo stato viene eliminato.

Al termine della finestra, Lambda utilizza l'elaborazione finale per le operazioni sui risultati dell'aggregazione. L'elaborazione finale viene richiamata in modo sincrono. Dopo il richiamo riuscito, la funzione controlla il numero di sequenza e l'elaborazione del flusso continua. Se il richiamo non ha esito positivo, la funzione Lambda sospende l'ulteriore elaborazione fino a quando non viene eseguito correttamente il richiamo.

**Example KinesisTimeWindowEvent**

```
{
  "Records": [
    {
      "kinesis": {
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber":
"49590338271490256608559692538361571095921575989136588898",
        "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
        "approximateArrivalTimestamp": 1607497475.000
      },
      "eventSource": "aws:kinesis",
      "eventVersion": "1.0",
      "eventID":
"shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
      "eventName": "aws:kinesis:record",
      "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-kinesis-role",
      "awsRegion": "us-east-1",
      "eventSourceARN": "arn:aws:kinesis:us-east-1:123456789012:stream/lambda-
stream"
    }
  ]
}
```

```
    }
  ],
  "window": {
    "start": "2020-12-09T07:04:00Z",
    "end": "2020-12-09T07:06:00Z"
  },
  "state": {
    "1": 282,
    "2": 715
  },
  "shardId": "shardId-0000000000006",
  "eventSourceARN": "arn:aws:kinesis:us-east-1:123456789012:stream/lambda-stream",
  "isFinalInvokeForWindow": false,
  "isWindowTerminatedEarly": false
}
```

## Configurazione

È possibile configurare le finestre a cascata quando si crea o si aggiorna un mapping di origini di eventi. Per configurare una finestra rotante, specificate la finestra in secondi ([TumblingWindowInSeconds](#)). Il seguente comando example AWS Command Line Interface (AWS CLI) crea una mappatura della sorgente di un evento di streaming con una finestra rotante di 120 secondi. La funzione Lambda definita per l'aggregazione e l'elaborazione è denominata `tumbling-window-example-function`.

```
aws lambda create-event-source-mapping \  
--event-source-arn arn:aws:kinesis:us-east-1:123456789012:stream/lambda-stream \  
--function-name tumbling-window-example-function \  
--starting-position TRIM_HORIZON \  
--tumbling-window-in-seconds 120
```

Lambda determina i limiti delle finestre a cascata in base al momento in cui i record sono stati inseriti nel flusso. Tutti i record dispongono di un timestamp approssimativo che Lambda utilizza nelle determinazioni dei limiti.

Le aggregazioni delle finestre a cascata non supportano il risharding. Quando uno shard termina, Lambda considera chiusa la finestra corrente e tutti i frammenti secondari iniziano la propria finestra in un nuovo stato. Quando non vengono aggiunti nuovi record alla finestra corrente, Lambda attende fino a due minuti prima di presumere che la finestra sia terminata. Ciò aiuta a garantire che la funzione legga tutti i record nella finestra corrente, anche se i record vengono aggiunti a intermittenza.

Le finestre a cascata supportano completamente le policy di ripetizione dei tentativi esistenti `maxRetryAttempts` e `maxRecordAge`.

### Example Handler.py — Aggregazione ed elaborazione

La seguente funzione Python mostra come aggregare e quindi elaborare lo stato finale:

```
def lambda_handler(event, context):
    print('Incoming event: ', event)
    print('Incoming state: ', event['state'])

    #Check if this is the end of the window to either aggregate or process.
    if event['isFinalInvokeForWindow']:
        # logic to handle final state of the window
        print('Destination invoke')
    else:
        print('Aggregate invoke')

    #Check for early terminations
    if event['isWindowTerminatedEarly']:
        print('Window terminated early')

    #Aggregation logic
    state = event['state']
    for record in event['Records']:
        state[record['kinesis']['partitionKey']] = state.get(record['kinesis']
['partitionKey'], 0) + 1

    print('Returning state: ', state)
    return {'state': state}
```

## Parametri Lambda per gli strumenti di mappatura dell'origine degli eventi di un flusso di dati Amazon Kinesis

Tutti gli strumenti di mappatura dell'origine degli eventi Lambda condividono le stesse operazioni API [CreateEventSourceMapping](#) e [UpdateEventSourceMapping](#). Tuttavia, solo alcuni dei parametri si applicano a Kinesis.

| Parametro                 | Obbligatorio | Predefinito | Note             |
|---------------------------|--------------|-------------|------------------|
| <a href="#">BatchSize</a> | N            | 100         | Massimo: 10.000. |



| Parametro                                  | Obbligatorio | Predefinito | Note                                                                                                                                                                                                                                                                                   |
|--------------------------------------------|--------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">BisectBatchOnFunctionError</a> | N            | false       | nessuno                                                                                                                                                                                                                                                                                |
| <a href="#">DestinationConfig</a>          | N            | N/D         | Una destinazione coda Amazon SQS o argomento Amazon SNS per i record scartati. Per ulteriori informazioni, consulta <a href="#">Configurazione delle destinazioni per le chiamate non riuscite</a> .                                                                                   |
| <a href="#">Abilitazione di</a>            | N            | true        | nessuno                                                                                                                                                                                                                                                                                |
| <a href="#">EventSourceArn</a>             | Y            | N/D         | L'ARN del flusso dei dati o di un consumer di flusso.                                                                                                                                                                                                                                  |
| <a href="#">FunctionName</a>               | Y            | N/D         | nessuno                                                                                                                                                                                                                                                                                |
| <a href="#">FunctionResponseTypes</a>      | N            | N/D         | Per consentire alla funzione di segnalare errori specifici in un batch, includi il valore ReportBatchItemFailures in FunctionResponseTypes. Per ulteriori informazioni, consulta <a href="#">Configurazione della risposta batch parziale con il flusso di dati Kinesis e Lambda</a> . |

| Parametro                                      | Obbligatorio | Predefinito | Note                                                                                                                                                                                                       |
|------------------------------------------------|--------------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">MaximumBatchingWindowInSeconds</a> | N            | 0           | nessuno                                                                                                                                                                                                    |
| <a href="#">MaximumRecordAgeInSeconds</a>      | N            | -1          | <p>-1 sta per infinito: Lambda non scarta i record (le <a href="#">impostazioni di conservazione dei dati del flusso di dati Kinesis</a> sono ancora valide)</p> <p>Minimo: -1</p> <p>Massimo: 604.800</p> |
| <a href="#">MaximumRetryAttempts</a>           | N            | -1          | <p>-1 sta per infinito: i record non riusciti vengono ripetuti fino alla scadenza del record</p> <p>Minimo: -1</p> <p>Massimo: 10.000.</p>                                                                 |
| <a href="#">ParallelizationFactor</a>          | N            | 1           | Maximum: 10                                                                                                                                                                                                |
| <a href="#">StartingPosition</a>               | Y            | N/D         | AT_TIMESTAMP, TRIM_HORIZON o LATEST                                                                                                                                                                        |

| Parametro                                 | Obbligatorio | Predefinito | Note                                                                                                                       |
|-------------------------------------------|--------------|-------------|----------------------------------------------------------------------------------------------------------------------------|
| <a href="#">StartingPositionTimestamp</a> | N            | N/D         | Valido solo se StartingPosition è impostato su AT_TIMESTAMP. Il tempo da cui avviare la lettura, in secondi di tempo Unix. |
| <a href="#">TumblingWindowInSeconds</a>   | N            | N/D         | Minimo: 0<br>Massimo: 900                                                                                                  |

## Utilizzo del filtro eventi con un'origine eventi Kinesis

Puoi utilizzare il filtraggio degli eventi per controllare quali record di un flusso o di una coda Lambda invia alla funzione. Per informazioni generali sul funzionamento del filtraggio eventi, consulta [the section called “Filtro eventi”](#).

In questa sezione viene descritto il filtraggio degli eventi per le origini eventi Kinesis.

### Note

Le mappature delle sorgenti degli eventi Kinesis supportano solo il filtraggio sulla chiave. data

## Argomenti

- [Nozioni di base sul filtro di eventi Kinesis](#)
- [Filtraggio dei record aggregati Kinesis](#)

## Nozioni di base sul filtro di eventi Kinesis

Supponiamo che un produttore stia inserendo dati in formato JSON nel flusso di dati Kinesis. Un record di esempio sarebbe simile al seguente, con i dati JSON convertiti in una stringa codificata Base64 nel campo data.

```
{
  "kinesis": {
    "kinesisSchemaVersion": "1.0",
    "partitionKey": "1",
    "sequenceNumber": "49590338271490256608559692538361571095921575989136588898",
    "data":
      "eyJJSZWNvcnR0dW1iZXIiOiAiMDAwMSIsICJUaW1lU3RhbnAiOiAiAieXl5eS1tbS1kZFRoRDptbTpczyIsICJSZXN0Q",
    "approximateArrivalTimestamp": 1545084650.987
  },
  "eventSource": "aws:kinesis",
  "eventVersion": "1.0",
  "eventID":
    "shardId-0000000000006:49590338271490256608559692538361571095921575989136588898",
  "eventName": "aws:kinesis:record",
  "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
  "awsRegion": "us-east-2",
  "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream"
}
```

Fintantoché i dati che il produttore inserisce nel flusso sono JSON validi, puoi utilizzare il filtraggio degli eventi per filtrare i record utilizzando la chiave data. Supponiamo che un produttore stia inserendo dati in formato JSON nel flusso di dati Kinesis.

```
{
  "record": 12345,
  "order": {
    "type": "buy",
    "stock": "ANYCO",
    "quantity": 1000
  }
}
```

Per filtrare solo i record in cui il tipo di ordine è «acquisto», l'`FilterCriteria` oggetto sarebbe il seguente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"data\" : { \"order\" : { \"type\" : [ \"buy\" ] } } }"
    }
  ]
}
```

```
}
```

Per una maggiore chiarezza, ecco il valore del Pattern del filtro espanso in JSON semplice.

```
{
  "data": {
    "order": {
      "type": [ "buy" ]
    }
  }
}
```

Puoi aggiungere il filtro utilizzando la console AWS CLI o un AWS SAM modello.

### Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in [Collegamento dei criteri di filtro a una mappatura dell'origine evento \(console\)](#) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "data" : { "order" : { "type" : [ "buy" ] } } }
```

### AWS CLI

Per creare una nuova mappatura della sorgente di eventi con questi criteri di filtro utilizzando AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/my-stream \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"order\" : { \"type\" : [ \"buy\" ] } } }"}]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"order\" : { \"type\" : [ \"buy\" ] } } }"}]}'
```

## AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungete il seguente frammento al modello YAML per la fonte dell'evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : { "order" : { "type" : [ "buy" ] } } }'
```

Per filtrare correttamente gli eventi da origini Kinesis, sia il campo dati sia i criteri di filtraggio per il campo dati devono essere in un formato JSON valido. Se uno dei due campi non è in un formato JSON valido, Lambda rilascia il messaggio o genera un'eccezione. La tabella seguente riepiloga il comportamento specifico:

| Formato dei dati in entrata | Formato del modello di filtro per le proprietà di dati | Operazione risultante                                                                                                                                                                                 |
|-----------------------------|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| JSON valido                 | JSON valido                                            | Filtri Lambda in base ai criteri di filtro.                                                                                                                                                           |
| JSON valido                 | Nessun modello di filtro per le proprietà dei dati     | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.                                                                                                                 |
| JSON valido                 | Non-JSON                                               | Lambda genera un'eccezione al momento della creazione o dell'aggiornamento della mappatura dell'origine evento. Il modello di filtro per le proprietà dei dati deve essere in un formato JSON valido. |
| Non-JSON                    | JSON valido                                            | Lambda rilascia il registro.                                                                                                                                                                          |
| Non-JSON                    | Nessun modello di filtro per le proprietà dei dati     | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.                                                                                                                 |

| Formato dei dati in entrata | Formato del modello di filtro per le proprietà di dati | Operazione risultante                                                                                                                                                                                 |
|-----------------------------|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Non-JSON                    | Non-JSON                                               | Lambda genera un'eccezione al momento della creazione o dell'aggiornamento della mappatura dell'origine evento. Il modello di filtro per le proprietà dei dati deve essere in un formato JSON valido. |

### Filtraggio dei record aggregati Kinesis

Con Kinesis, puoi aggregare più record in un unico record flusso di dati Kinesis per aumentare il throughput dei dati. Lambda può applicare criteri di filtraggio ai record aggregati solo quando si utilizza il [fan-out avanzato](#) di Kinesis. Il filtraggio dei record aggregati con Kinesis standard non è supportato. Quando utilizzi il fan-out avanzato, configuri un consumatore Kinesis a throughput dedicato che funga da trigger per la funzione Lambda. Lambda filtra quindi i record aggregati e passa solo i record che soddisfano i criteri di filtraggio.

Per ulteriori informazioni sull'aggregazione dei record Kinesis, consulta la sezione [Aggregazione](#) nella pagina Concetti chiave di Kinesis Producer Library (KPL). Per saperne di più sull'utilizzo di Lambda con Kinesis Enhanced fan-out, consulta [Increasing real-time stream processing performance with Amazon Kinesis Data Streams enhanced fan-out e Lambda nel blog di calcolo. AWS](#) AWS

### Tutorial: Utilizzo di Lambda con Flusso di dati Kinesis

In questo tutorial, viene creata una funzione Lambda per utilizzare gli eventi da un flusso di dati Amazon Kinesis.

1. L'app personalizzata scrive record nel flusso.
2. AWS Lambda esegue il polling dello stream e, quando rileva nuovi record nello stream, richiama la funzione Lambda.
3. AWS Lambda esegue la funzione Lambda assumendo il ruolo di esecuzione specificato al momento della creazione della funzione Lambda.

## Prerequisiti

### Installa AWS Command Line Interface

Se non hai ancora installato il AWS Command Line Interface, segui i passaggi in [Installazione o aggiornamento della versione più recente AWS CLI](#) per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

#### Note

Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere una Windows-integrated versione di Ubuntu e Bash, [installa il sottosistema Windows per Linux](#).

### Creazione del ruolo di esecuzione

Crea il ruolo di [esecuzione](#) che autorizza la tua funzione ad accedere AWS alle risorse.

Per creare un ruolo di esecuzione

1. Apri la pagina [Ruoli](#) nella console IAM.
2. Scegliere Crea ruolo.
3. Creare un ruolo con le seguenti proprietà.
  - Trusted entity (Entità attendibile) – AWS Lambda.
  - Autorizzazioni — `AWSLambdaKinesisExecutionRole`.
  - Nome ruolo – **lambda-kinesis-role**.

La `AWSLambdaKinesisExecutionRole` policy dispone delle autorizzazioni necessarie alla funzione per leggere gli elementi da Kinesis e scrivere i log nei log. CloudWatch

### Creazione della funzione

Crea una funzione Lambda che elabora i messaggi Kinesis. Il codice della funzione registra l'ID dell'evento e i dati dell'evento del record Kinesis in Logs. CloudWatch



Questo tutorial utilizza il runtime Node.js 24, ma abbiamo anche fornito codice di esempio in altri linguaggi di runtime. Per visualizzare il codice per il runtime che ti interessa, seleziona la scheda corrispondente nella casella seguente. Il JavaScript codice che utilizzerai in questo passaggio si trova nel primo esempio mostrato nella [JavaScript](#) scheda.

## .NET

### SDK per .NET

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }
    }
}
```

```

    }

    foreach (var record in evnt.Records)
    {
        try
        {
            Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
            string data = await GetRecordDataAsync(record.Kinesis, context);
            Logger.LogInformation($"Data: {data}");
            // TODO: Do interesting work based on the new data
        }
        catch (Exception ex)
        {
            Logger.LogError($"An error occurred {ex.Message}");
            throw;
        }
    }
    Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
}

private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
{
    byte[] bytes = record.Data.ToArray();
    string data = Encoding.UTF8.GetString(bytes);
    await Task.CompletedTask; //Placeholder for actual async work
    return data;
}
}

```

Go

SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Utilizzo di un evento Kinesis con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "log"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
    if len(kinesisEvent.Records) == 0 {
        log.Printf("empty Kinesis event received")
        return nil
    }

    for _, record := range kinesisEvent.Records {
        log.Printf("processed Kinesis event with EventId: %v", record.EventID)
        recordDataBytes := record.Kinesis.Data
        recordDataText := string(recordDataBytes)
        log.Printf("record data: %v", recordDataText)
        // TODO: Do interesting work based on the new data
    }
    log.Printf("successfully processed %v records", len(kinesisEvent.Records))
    return nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Utilizzo di un evento Kinesis con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

public class Handler implements RequestHandler<KinesisEvent, Void> {
    @Override
    public Void handleRequest(final KinesisEvent event, final Context context) {
        LambdaLogger logger = context.getLogger();
        if (event.getRecords().isEmpty()) {
            logger.log("Empty Kinesis Event received");
            return null;
        }
        for (KinesisEvent.KinesisEventRecord record : event.getRecords()) {
            try {
                logger.log("Processed Event with EventId: "+record.getEventID());
                String data = new String(record.getKinesis().getData().array());
                logger.log("Data:"+ data);
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex) {
                logger.log("An error occurred:"+ex.getMessage());
                throw ex;
            }
        }
        logger.log("Successfully processed:"+event.getRecords().size()+"
records");
    }
}
```

```
        return null;
    }

}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Consumare un evento Kinesis utilizzando Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    for (const record of event.Records) {
        try {
            console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
            const recordData = await getRecordDataAsync(record.kinesis);
            console.log(`Record Data: ${recordData}`);
            // TODO: Do interesting work based on the new data
        } catch (err) {
            console.error(`An error occurred ${err}`);
            throw err;
        }
    }
    console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
    var data = Buffer.from(payload.data, "base64").toString("utf-8");
    await Promise.resolve(1); //Placeholder for actual async work
    return data;
}
```

## Utilizzo di un evento Kinesis con Lambda utilizzando TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
    KinesisStreamEvent,
    Context,
    KinesisStreamHandler,
    KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
    logLevel: "INFO",
    serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
    event: KinesisStreamEvent,
    context: Context
): Promise<void> => {
    for (const record of event.Records) {
        try {
            logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
            const recordData = await getRecordDataAsync(record.kinesis);
            logger.info(`Record Data: ${recordData}`);
            // TODO: Do interesting work based on the new data
        } catch (err) {
            logger.error(`An error occurred ${err}`);
            throw err;
        }
        logger.info(`Successfully processed ${event.Records.length} records.`);
    }
};

async function getRecordDataAsync(
    payload: KinesisStreamRecordPayload
): Promise<string> {
    var data = Buffer.from(payload.data, "base64").toString("utf-8");
    await Promise.resolve(1); //Placeholder for actual async work
    return data;
}
```

## PHP

### SDK per PHP

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
    }
}
```

```

        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
            marked as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);

```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite Python.

```

# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import base64
def lambda_handler(event, context):

    for record in event['Records']:
        try:
            print(f"Processed Kinesis Event - EventID: {record['eventID']}")
            record_data = base64.b64decode(record['kinesis']
['data']).decode('utf-8')
            print(f"Record Data: {record_data}")
            # TODO: Do interesting work based on the new data

```



```

    except Exception as e:
        print(f"An error occurred {e}")
        raise e
print(f"Successfully processed {len(event['Records'])} records.")

```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite Ruby.

```

# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue => err
      $stderr.puts "An error occurred #{err}"
      raise err
    end
  end
  puts "Successfully processed #{event['Records'].length} records."
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('UTF-8')
  # Placeholder for actual async work
  # You can use Ruby's asynchronous programming tools like async/await or fibers
  here.

```

```
    return data
end
```

## Rust

### SDK per Rust

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error>
{
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
{}", record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    })
}
```

```

    });

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
        time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```

## Creazione della funzione

1. Crea una directory per il progetto, quindi passa a quella directory.

```

mkdir kinesis-tutorial
cd kinesis-tutorial

```

2. Copia il JavaScript codice di esempio in un nuovo file denominato `index.js`.
3. Crea un pacchetto di implementazione.

```

zip function.zip index.js

```

4. Creare una funzione Lambda con il comando `create-function`.

```

aws lambda create-function --function-name ProcessKinesisRecords \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs24.x \
--role arn:aws:iam::111122223333:role/lambda-kinesis-role

```

## Test della funzione Lambda

Richiamate manualmente la funzione Lambda utilizzando il comando `invoke` AWS Lambda CLI e un evento Kinesis di esempio.

### Verifica della funzione Lambda

1. Copiare il codice JSON seguente in un file e salvarlo con nome `input.txt`.

```
{
  "Records": [
    {
      "kinesis": {
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber":
"49590338271490256608559692538361571095921575989136588898",
        "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
        "approximateArrivalTimestamp": 1545084650.987
      },
      "eventSource": "aws:kinesis",
      "eventVersion": "1.0",
      "eventID":
"shardId-0000000000006:49590338271490256608559692538361571095921575989136588898",
      "eventName": "aws:kinesis:record",
      "invokeIdentityArn": "arn:aws:iam::111122223333:role/lambda-kinesis-
role",
      "awsRegion": "us-east-2",
      "eventSourceARN": "arn:aws:kinesis:us-east-2:111122223333:stream/
lambda-stream"
    }
  ]
}
```

2. Utilizzare il comando `invoke` per inviare l'evento alla funzione.

```
aws lambda invoke --function-name ProcessKinesisRecords \
--cli-binary-format raw-in-base64-out \
--payload file://input.txt outputfile.txt
```

L'opzione `cli-binary-format` è obbligatoria se si utilizza la versione 2. AWS CLI Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-`

in-base64-out. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

La risposta viene salvata in `out.txt`.

## Creare un flusso Kinesis

Utilizzare il comando `create-stream` per creare un flusso.

```
aws kinesis create-stream --stream-name lambda-stream --shard-count 1
```

Eeguire il seguente comando `describe-stream` per ottenere il flusso ARN.

```
aws kinesis describe-stream --stream-name lambda-stream
```

Verrà visualizzato l'output seguente:

```
{
  "StreamDescription": {
    "Shards": [
      {
        "ShardId": "shardId-000000000000",
        "HashKeyRange": {
          "StartingHashKey": "0",
          "EndingHashKey": "340282366920746074317682119384634633455"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
"49591073947768692513481539594623130411957558361251844610"
        }
      }
    ],
    "StreamARN": "arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream",
    "StreamName": "lambda-stream",
    "StreamStatus": "ACTIVE",
    "RetentionPeriodHours": 24,
    "EnhancedMonitoring": [
      {
        "ShardLevelMetrics": []
      }
    ]
  },
}
```

```
"EncryptionType": "NONE",  
"KeyId": null,  
"StreamCreationTimestamp": 1544828156.0  
}  
}
```

Nella fase successiva utilizzare l'ARN del flusso per associare il flusso alla funzione Lambda.

Aggiungi una fonte di eventi in AWS Lambda

Eseguire il seguente comando AWS CLI `add-event-source`.

```
aws lambda create-event-source-mapping --function-name ProcessKinesisRecords \  
--event-source arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream \  
--batch-size 100 --starting-position LATEST
```

Annotare l'ID della mappatura per utilizzarlo in futuro. Mediante l'esecuzione del comando `list-event-source-mappings` è possibile ottenere un elenco di mappature delle origini eventi.

```
aws lambda list-event-source-mappings --function-name ProcessKinesisRecords \  
--event-source arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream
```

Nella risposta è possibile verificare che il valore dello stato sia `enabled` (attivato). Le mappature dell'origine eventi possono essere disabilitate per sospendere il polling temporaneamente senza perdere alcuni record.

Eseguire il test della configurazione

Per testare la mappatura dell'origine eventi, aggiungere i record dell'evento al flusso Kinesis. Il `--data` a valore è una stringa che la CLI codifica in base64 prima di inviarla a Kinesis. È possibile eseguire lo stesso comando più volte per aggiungere più record al flusso.

```
aws kinesis put-record --stream-name lambda-stream --partition-key 1 \  
--data "Hello, this is a test."
```

Lambda usa il ruolo di esecuzione per leggere i record dal flusso. Quindi richiama la funzione Lambda, passando ai batch dei record. La funzione decodifica i dati di ogni record e li registra, inviando l'output a Logs. CloudWatch Visualizza i log nella [console CloudWatch](#).

## Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili al tuo Account AWS

### Come eliminare il ruolo di esecuzione

1. Aprire la pagina [Ruoli](#) della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

### Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **confirm** nel campo di immissione testo e scegli Delete (Elimina).

### Per eliminare il flusso Kinesis

1. Accedi Console di gestione AWS e apri la console Kinesis all'indirizzo. <https://console.aws.amazon.com/kinesis>
2. Selezionare il flusso creato.
3. Scegli Operazioni > Elimina.
4. Inserisci **delete** nel campo di immissione del testo.
5. Scegli Delete (Elimina).

## Utilizzo di Lambda con Kubernetes

Puoi implementare e gestire funzioni Lambda con l'API Kubernetes utilizzando i [controller AWS per Kubernetes \(ACK\)](#) o [Crossplane](#).

### Controller AWS per Kubernetes (ACK)

Puoi utilizzare ACK per implementare e gestire risorse AWS dall'API Kubernetes. Tramite ACK, AWS fornisce controller personalizzati open source per servizi AWS, ad esempio Lambda, Amazon Elastic Container Registry (Amazon ECR), Amazon Simple Storage Service (Amazon S3) e Amazon SageMaker AI. Ogni servizio AWS supportato ha un proprio controller personalizzato. Nel cluster Kubernetes installa un controller per ogni servizio AWS da utilizzare. Crea, quindi, una [definizione risorsa personalizzata \(CRD\)](#) per definire le risorse AWS.

È preferibile utilizzare [Helm 3.8 o versioni successive](#) per installare i controller ACK. Ogni controller ACK è dotato di un proprio grafico Helm, che installa il controller, le CRD e le regole RBAC di Kubernetes. Per ulteriori informazioni, consulta [Installare un ACK Controller](#) nella documentazione di ACK.

Dopo aver creato la risorsa personalizzata ACK, puoi utilizzarla come qualunque altro oggetto Kubernetes integrato. Ad esempio, puoi implementare e gestire funzioni Lambda con le tue toolchain Kubernetes preferite, inclusa [kubectl](#).

Di seguito sono riportati alcuni esempi di casi d'uso per il provisioning di funzioni Lambda tramite ACK:

- La tua organizzazione utilizza il [controllo degli accessi basato sui ruoli \(RBAC\)](#) e [ruoli IAM per gli account di servizio](#) per creare limiti delle autorizzazioni. Con ACK puoi riutilizzare questo modello di sicurezza per Lambda senza bisogno di creare nuovi utenti e policy.
- La tua organizzazione utilizza un processo DevOps per implementare le risorse in un cluster Amazon Elastic Kubernetes Service (Amazon EKS) utilizzando manifesti Kubernetes. Con ACK puoi utilizzare un manifesto per eseguire il provisioning delle funzioni Lambda senza creare un'infrastruttura separata come modelli di codice.

Per ulteriori informazioni sull'utilizzo di ACK, consulta il [tutorial Lambda nella documentazione di ACK](#).



## Crossplane

[Crossplane](#) è un progetto CNCF (Cloud Native Computing Foundation) che utilizza Kubernetes per gestire risorse dell'infrastruttura cloud. Con Crossplane gli sviluppatori possono richiedere l'infrastruttura senza doverne comprendere le complessità. I team della piattaforma mantengono il controllo sulle modalità di provisioning e gestione dell'infrastruttura.

Utilizzando Crossplane puoi implementare e gestire funzioni Lambda con le tue toolchain Kubernetes preferite, ad esempio [kubectrl](#), e qualunque pipeline CI/CD in grado di implementare manifesti su Kubernetes. Di seguito sono riportati alcuni esempi di casi d'uso per il provisioning di funzioni Lambda tramite Crossplane:

- La tua organizzazione desidera attenersi alla conformità accertandosi che le funzioni Lambda abbiano [tag](#) corretti. I team della piattaforma possono utilizzare [composizioni Crossplane](#) per definire questa policy tramite astrazioni API. Gli sviluppatori, quindi, possono utilizzare tali astrazioni per implementare funzioni Lambda con tag.
- Il tuo progetto utilizza GitOps con Kubernetes. In questo modello, Kubernetes riconcilia continuamente il repository git (stato desiderato) con le risorse in esecuzione all'interno del cluster (stato corrente). Se esistono differenze, il processo GitOps apporta automaticamente modifiche al cluster. Puoi utilizzare GitOps con Kubernetes per implementare e gestire funzioni Lambda tramite Crossplane, adoperando strumenti e concetti familiari di Kubernetes, ad esempio [CRD](#) e [controller](#).

Per ulteriori informazioni sull'utilizzo di Crossplane con Lambda, consulta:

- [Schemi AWS per Crossplane](#): questo repository include esempi di utilizzo di Crossplane per l'implementazione di risorse AWS, incluse le funzioni Lambda.

### Note

Gli schemi AWS per Crossplane sono in fase di sviluppo attivo e non devono essere utilizzati in produzione.

- [Implementazione di Lambda con Amazon EKS e Crossplane](#): questo video mostra un esempio avanzato di implementazione di un'architettura serverless AWS con Crossplane, esplorando la progettazione dal punto di vista dello sviluppatore e della piattaforma.

## Utilizzo di Lambda con Amazon MQ

### Note

Se desideri inviare dati a una destinazione diversa da una funzione Lambda o arricchire i dati prima di inviarli, consulta [Amazon EventBridge Pipes](#).

Amazon MQ è un servizio gestito di broker dei messaggi per [Apache ActiveMQ](#) e [RabbitMQ](#). Un broker di messaggi consente alle applicazioni e ai componenti software di comunicare utilizzando vari linguaggi di programmazione, sistemi operativi e protocolli di messaggistica formali tramite destinazioni eventi di tipo argomento o coda.

Amazon MQ può anche gestire automaticamente istanze di Amazon Elastic Compute Cloud (Amazon EC2) con l'installazione di ActiveMQ o RabbitMQ e la fornitura di diverse topologie di rete e altri requisiti infrastrutturali.

Puoi utilizzare una funzione Lambda per elaborare i record da un broker di messaggi di Amazon MQ. Lambda richiama la funzione tramite uno [strumento di mappatura dell'origine degli eventi](#), una risorsa Lambda che legge i messaggi dal broker e richiama la funzione [in maniera sincrona](#). Per le opzioni di configurazione, consulta [the section called "Parametri"](#). Per gli errori più comuni, vedere [the section called "Risoluzione dei problemi"](#).

### Warning

Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per saperne di più, vedi [Come posso rendere idempotente la mia funzione Lambda](#) nel Knowledge Center. AWS

Lo strumento di mappatura dell'origine degli eventi di Amazon MQ presenta le seguenti restrizioni di configurazione:

- **Simultaneità:** le funzioni Lambda che utilizzano uno strumento di mappatura dell'origine degli eventi Amazon MQ hanno un'impostazione di [simultaneità](#) massima predefinita. Per ActiveMQ, il servizio Lambda limita il numero di ambienti di esecuzione simultanei a cinque per lo strumento

di mappatura dell'origine degli eventi di Amazon MQ. Per RabbitMQ, il numero di ambienti di esecuzione simultanei è limitato a 1 per lo strumento di mappatura dell'origine degli eventi di Amazon MQ.

Anche se modifichi le impostazioni di simultaneità sottoposta a provisioning o riservata, il servizio Lambda non renderà disponibili altri ambienti di esecuzione. Per richiedere un aumento della concorrenza massima predefinita per una singola mappatura della sorgente di eventi Amazon MQ, contatta l'UUID di Supporto mappatura della sorgente dell'evento e la regione. Poiché gli aumenti vengono applicati al livello dello strumento di mappatura dell'origine degli eventi specifico, non a livello di account o regione, è necessario richiedere manualmente un aumento di scalabilità per ogni strumento di mappatura dell'origine degli eventi.

- Più account: Lambda non supporta l'elaborazione tra più account. Non puoi utilizzare Lambda per elaborare i record da un broker di messaggi di Amazon MQ incluso in un Account AWS diverso.
- Autenticazione: per ActiveMQ, è supportato solo ActiveMQ. [SimpleAuthenticationPlugin](#) Per RabbitMQ è supportato solo il meccanismo di autenticazione [PLAIN](#). Gli utenti devono utilizzare per gestire le proprie AWS Secrets Manager credenziali. Per ulteriori informazioni sull'autenticazione di ActiveMQ, consulta [Integrazione di broker ActiveMQ con LDAP](#) nella Guida per gli sviluppatori di Amazon MQ.
- Quota di connessione: i broker hanno un numero massimo di connessioni consentite per protocollo a livello di collegamento. Questa quota si basa sul tipo di istanza del broker. Per ulteriori informazioni, consulta la sezione [Broker](#) di Quote in Amazon MQ nella Guida per gli sviluppatori di Amazon MQ.
- Connettività: puoi creare broker in un VPC (Virtual Private Cloud) pubblico o privato. Per i VPC privati, la funzione Lambda deve accedere al VPC per ricevere i messaggi. Per ulteriori informazioni, consulta [the section called “Configurare la sicurezza della rete”](#) di seguito in questa sezione.
- Destinazioni eventi: sono supportate solo le destinazioni coda. Tuttavia, puoi utilizzare un argomento virtuale che si comporta internamente come un argomento mentre interagisce con Lambda come una coda. Per ulteriori informazioni, consulta [Destinazioni virtuali](#) sul sito web di Apache ActiveMQ e [Host virtuali](#) sul sito Web di RabbitMQ.
- Topologia di rete: per ActiveMQ è supportato un solo broker a istanza singola o in standby per ogni strumento di mappatura dell'origine degli eventi. Per RabbitMQ, è supportata una sola distribuzione di broker o cluster a istanza singola per mappatura dell'origine degli eventi. Single-instance i broker richiedono un endpoint di failover. Per ulteriori informazioni su queste modalità di implementazione del broker, consulta [Architettura del broker MQ attiva](#) e [Architettura del broker MQ di Rabbit](#) nella Guida per gli sviluppatori di Amazon MQ.

- Protocolli: i protocolli supportati dipendono dal tipo di integrazione di Amazon MQ.
  - Per le integrazioni ActiveMQ, Lambda utilizza i messaggi utilizzando il protocollo Message Service (JMS). OpenWire/Java Non sono supportati altri protocolli per l'utilizzo dei messaggi. All'interno del protocollo JMS, sono supportati solo [TextMessage](#) e [BytesMessage](#). Lambda supporta anche le proprietà JMS personalizzate. Per ulteriori informazioni sul OpenWire protocollo, [OpenWire](#) consulta il sito Web di Apache ActiveMQ.
  - Per le integrazioni RabbitMQ, Lambda utilizza i messaggi tramite il protocollo AMQP 0-9-1. Non sono supportati altri protocolli per l'utilizzo dei messaggi. Per ulteriori informazioni sull'implementazione del protocollo AMQP 0-9-1 in RabbitMQ, consulta la [Guida di riferimento completa di AMQP 0-9-1](#) sul sito web di RabbitMQ.

Lambda supporta automaticamente le versioni più recenti di ActiveMQ e RabbitMQ supportate da Amazon MQ. Per le ultime versioni supportate, consulta le [Note di rilascio di Amazon MQ](#) nella Guida per gli sviluppatori di Amazon MQ.

#### Note

Per impostazione predefinita, Amazon MQ prevede un periodo di manutenzione settimanale per i broker. Durante quella finestra temporale, i broker non sono disponibili. Per i broker senza standby, Lambda non può elaborare alcun messaggio durante tale finestra.

## Argomenti

- [Informazioni sul gruppo di consumatori Lambda per Amazon MQ](#)
- [Configurazione dell'origine eventi Amazon MQ per Lambda](#)
- [Parametri dello strumento di mappatura dell'origine degli eventi](#)
- [Filtrare gli eventi da una origine eventi Amazon MQ](#)
- [Risolvere i problemi relativi gli errori di mappatura dell'origine eventi di Amazon MQ](#)

## Informazioni sul gruppo di consumatori Lambda per Amazon MQ

Per interagire con Amazon MQ, Lambda crea un gruppo di utenti che può leggere dai broker di Amazon MQ. Il gruppo di utenti viene creato con lo stesso ID dell'UUID dello strumento di mappatura dell'origine degli eventi.

Per le origini eventi di Amazon MQ, Lambda crea un batch dei record e li invia alla tua funzione in un singolo payload. Per controllare il comportamento, puoi configurare la finestra batch e le dimensioni del batch. Lambda estrae i messaggi finché elabora la dimensione del payload massima di 6 MB, la finestra di batch scade o il numero di record raggiunge la dimensione completa del batch. Per ulteriori informazioni, consulta [Comportamento di batching](#).

Il gruppo di utenti recupera i messaggi come BLOB di byte, li codifica in base64 in un singolo payload JSON e richiama la tua funzione. Se la funzione restituisce un errore per uno qualunque dei messaggi in un batch, Lambda ritenta l'intero batch di messaggi fino a quando l'elaborazione riesce o i messaggi scadono.

#### Note

Anche se le funzioni Lambda generalmente prevedono un timeout massimo di 15 minuti, gli strumenti di mappatura dell'origine degli eventi per Amazon MSK, Apache Kafka autogestito, Amazon DocumentDB e Amazon MQ per ActiveMQ e RabbitMQ supportano solo funzioni con timeout massimi di 14 minuti. Questa limitazione garantisce che lo strumento di mappatura dell'origine degli eventi possa gestire correttamente errori di funzioni e nuovi tentativi.

Puoi monitorare l'utilizzo della concorrenza di una determinata funzione utilizzando la metrica in Amazon. `ConcurrentExecutions` CloudWatch Per ulteriori informazioni sulla simultaneità, consulta [the section called “Configurazione della simultaneità riservata”](#).

Example Eventi record di Amazon MQ

ActiveMQ

```
{
  "eventSource": "aws:mq",
  "eventSourceArn": "arn:aws:mq:us-east-2:111122223333:broker:test:b-9bcfa592-423a-4942-879d-eb284b418fc8",
  "messages": [
    {
      "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-east-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
      "messageType": "jms/text-message",
      "deliveryMode": 1,
      "replyTo": null,
      "type": null,
      "expiration": "60000",

```

```

    "priority": 1,
    "correlationId": "myJMScoID",
    "redelivered": false,
    "destination": {
      "physicalName": "testQueue"
    },
    "data": "QUJD0kFBQUE=",
    "timestamp": 1598827811958,
    "brokerInTime": 1598827811958,
    "brokerOutTime": 1598827811959,
    "properties": {
      "index": "1",
      "doAlarm": "false",
      "myCustomProperty": "value"
    }
  },
  {
    "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-east-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
    "messageType": "jms/bytes-message",
    "deliveryMode": 1,
    "replyTo": null,
    "type": null,
    "expiration": "60000",
    "priority": 2,
    "correlationId": "myJMScoID1",
    "redelivered": false,
    "destination": {
      "physicalName": "testQueue"
    },
    "data": "LQaGQ82S48k=",
    "timestamp": 1598827811958,
    "brokerInTime": 1598827811958,
    "brokerOutTime": 1598827811959,
    "properties": {
      "index": "1",
      "doAlarm": "false",
      "myCustomProperty": "value"
    }
  }
]
}

```

## RabbitMQ

```
{
  "eventSource": "aws:rmq",
  "eventSourceArn": "arn:aws:mq:us-east-2:111122223333:broker:pizzaBroker:b-9bcfa592-423a-4942-879d-eb284b418fc8",
  "rmqMessagesByQueue": {
    "pizzaQueue::/": [
      {
        "basicProperties": {
          "contentType": "text/plain",
          "contentEncoding": null,
          "headers": {
            "header1": {
              "bytes": [
                118,
                97,
                108,
                108,
                101,
                49
              ]
            },
            "header2": {
              "bytes": [
                118,
                97,
                108,
                108,
                101,
                50
              ]
            },
            "numberInHeader": 10
          },
          "deliveryMode": 1,
          "priority": 34,
          "correlationId": null,
          "replyTo": null,
          "expiration": "60000",
          "messageId": null,
          "timestamp": "Jan 1, 1970, 12:33:41 AM",
          "type": null,
```

```
        "userId": "AIDACKCEVSQ6C2EXAMPLE",
        "appId": null,
        "clusterId": null,
        "bodySize": 80
    },
    "redelivered": false,
    "data": "eyJ0aW1lb3V0IjowLCJkYXRhIjoiQ1pybWYwR3c4T3Y0YnFMUXhENEUifQ=="
  }
}
```

### Note

Nell'esempio di RabbitMQ, `pizzaQueue` è il nome della coda RabbitMQ e `/` è il nome dell'host virtuale. Quando si ricevono messaggi, l'origine eventi elenca i messaggi in `pizzaQueue::/`.

## Configurazione dell'origine eventi Amazon MQ per Lambda

### Argomenti

- [Configurare la sicurezza della rete](#)
- [Creare lo strumento di mappatura dell'origine degli eventi](#)

### Configurare la sicurezza della rete

Per concedere a Lambda l'accesso completo ad Amazon MQ tramite lo strumento di mappatura dell'origine degli eventi, il broker deve utilizzare un endpoint pubblico (indirizzo IP pubblico) oppure deve fornire l'accesso all'Amazon VPC in cui hai creato il broker.

Quando usi Amazon MQ con Lambda, crea [endpoint VPC AWS PrivateLink](#) che forniscono alla funzione l'accesso alle risorse del tuo Amazon VPC.

### Note

AWS PrivateLink Gli endpoint VPC sono necessari per le funzioni con mappature dell'origine degli eventi che utilizzano la modalità predefinita (su richiesta) per i poller di eventi. Se



la mappatura dell'origine degli eventi utilizza la modalità [provisioned](#), non è necessario configurare gli endpoint VPC. AWS PrivateLink

Crea un endpoint per fornire l'accesso alle seguenti risorse:

- Lambda: crea un endpoint per il principale del servizio Lambda.
- AWS STS — Crea un endpoint per consentire AWS STS a un responsabile del servizio di assumere un ruolo per tuo conto.
- Secrets Manager: se il tuo broker utilizza Secrets Manager per archiviare le credenziali, crea un endpoint per Secrets Manager.

In alternativa, configura un gateway NAT su ogni sottorete pubblica in Amazon VPC. Per ulteriori informazioni, consulta [the section called “Accesso Internet per funzioni del VPC”](#).

Quando crei uno strumento di mappatura dell'origine degli eventi per Amazon MQ, Lambda verifica se le interfacce di rete elastiche (ENI) sono già presenti per le sottoreti e i gruppi di sicurezza configurati per il tuo Amazon VPC. Se Lambda trova ENI esistenti, prova a riutilizzarle. Altrimenti, Lambda crea nuove ENI per connettersi all'origine dell'evento e richiamare la tua funzione.

#### Note

Le funzioni Lambda vengono sempre eseguite all'interno di VPC di proprietà del servizio Lambda. La configurazione VPC della funzione non influisce sullo strumento di mappatura dell'origine degli eventi. Solo la configurazione di rete dell'origine dell'evento determina il modo in cui Lambda si connette all'origine dell'evento.

Configura i gruppi di sicurezza per l'Amazon VPC contenente il tuo broker. Per impostazione predefinita, Amazon MQ utilizza le seguenti porte: 61617 (Amazon MQ per ActiveMQ) e 5671 (Amazon MQ per RabbitMQ).

- Regole in ingresso: consenti tutto il traffico sulla porta del broker predefinita per il gruppo di sicurezza associato all'origine eventi. In alternativa, puoi utilizzare una regola autoreferenziale del gruppo di sicurezza per consentire l'accesso da istanze all'interno dello stesso gruppo di sicurezza.
- Regole in uscita: consenti tutto il traffico sulla porta 443 per destinazioni esterne se la tua funzione deve comunicare con i servizi. AWS In alternativa, puoi anche utilizzare una regola di gruppo di

sicurezza autoreferenziale per limitare l'accesso al broker se non hai bisogno di comunicare con altri servizi. AWS

- Regole di ingresso degli endpoint Amazon VPC: se utilizzi un endpoint Amazon VPC, il gruppo di sicurezza associato all'endpoint Amazon VPC deve consentire il traffico in entrata sulla porta 443 dal gruppo di sicurezza del broker.

Se il broker utilizza l'autenticazione, puoi anche limitare la policy degli endpoint per l'endpoint Secrets Manager. Per chiamare l'API Secrets Manager, Lambda utilizza il ruolo della funzione, non il principale del servizio Lambda.

Example Policy dell'endpoint VPC: endpoint Secrets Manager

```
{
  "Statement": [
    {
      "Action": "secretsmanager:GetSecretValue",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws::iam::123456789012:role/my-role"
        ]
      },
      "Resource": "arn:aws::secretsmanager:us-west-2:123456789012:secret:my-secret"
    }
  ]
}
```

Quando utilizzi gli endpoint Amazon VPC, AWS indirizza le tue chiamate API per richiamare la tua funzione utilizzando l'Elastic Network Interface (ENI) dell'endpoint. Il principale del servizio Lambda deve chiamare `lambda:InvokeFunction` su tutti i ruoli e le funzioni che utilizzano tali ENI.

Per impostazione predefinita, gli endpoint Amazon VPC dispongono di policy IAM aperte che consentono un ampio accesso alle risorse. La best practice consiste nel limitare queste policy per eseguire le azioni necessarie utilizzando quell'endpoint. Per garantire che lo strumento di mappatura dell'origine degli eventi sia in grado di invocare la funzione Lambda, la policy degli endpoint VPC deve consentire al principale del servizio Lambda di chiamare `sts:AssumeRole` e `lambda:InvokeFunction`. Limitare le policy degli endpoint VPC per consentire solo le chiamate

API provenienti dall'organizzazione impedisce il corretto funzionamento dello strumento di mappatura dell'origine degli eventi, pertanto "Resource": "\*" è richiesto in queste policy.

Il seguente esempio di policy degli endpoint VPC mostra come concedere l'accesso richiesto al principale del servizio Lambda per gli endpoint AWS STS e Lambda.

#### Example Politica VPC Endpoint — AWS STS endpoint

```
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

#### Example Policy dell'endpoint VPC: endpoint Lambda

```
{
  "Statement": [
    {
      "Action": "lambda:InvokeFunction",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

## Creare lo strumento di mappatura dell'origine degli eventi

Creare una [mappatura dell'origine eventi](#) per indicare a Lambda di inviare i record da un broker Amazon MQ a una funzione Lambda. È possibile creare più mappature delle origini di eventi per elaborare gli stessi dati con più funzioni o per elaborare elementi da più fonti con una singola funzione.

Per configurare la funzione per la lettura da Amazon MQ, aggiungi le autorizzazioni richieste e crea un trigger MQ nella console Lambda.

Per leggere i record da un broker Amazon MQ, la funzione Lambda richiede le seguenti autorizzazioni. Concedi a Lambda l'autorizzazione a interagire con il tuo broker Amazon MQ e le relative risorse sottostanti aggiungendo istruzioni di autorizzazione al tuo [ruolo di esecuzione](#) della funzione:

- [mq: DescribeBroker](#)
- [responsabile dei segreti: GetSecretValue](#)
- [ec2: CreateNetworkInterface](#)
- [ec2: DeleteNetworkInterface](#)
- [ec2: DescribeNetworkInterfaces](#)
- [ec2: DescribeSecurityGroups](#)
- [ec2: DescribeSubnets](#)
- [ec2: DescribeVpcs](#)
- [registri: CreateLogGroup](#)
- [registri: CreateLogStream](#)
- [registri: PutLogEvents](#)

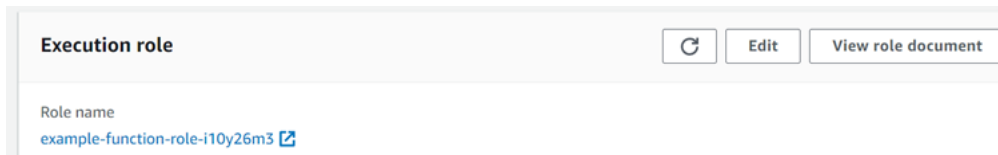
### Note

Quando utilizzi una chiave gestita dal cliente crittografata, aggiungi anche l'autorizzazione [kms:Decrypt](#).

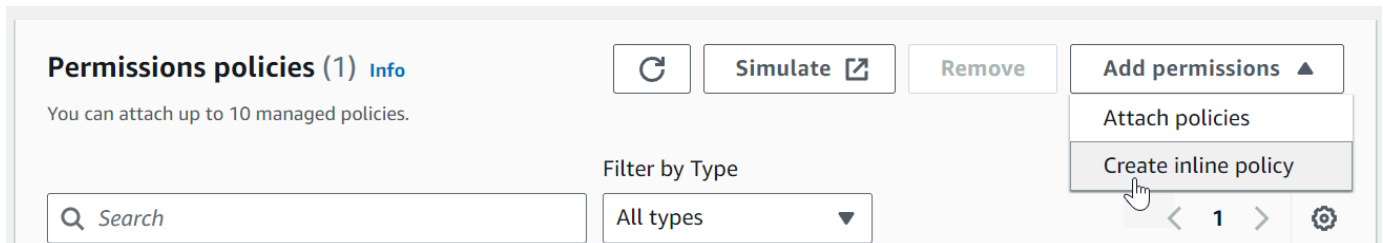
Per aggiungere le autorizzazioni e creare un trigger

1. Aprire la pagina [Functions](#) (Funzioni) della console Lambda.

- Scegliere il nome della funzione.
- Quindi, seleziona la scheda Configuration (Configurazione) e poi Permissions (Autorizzazioni).
- In Nome del ruolo, scegli il link al tuo ruolo di esecuzione. Questo ruolo si apre nella console IAM.



- Scegli Aggiungi autorizzazioni, quindi seleziona Crea policy in linea.



- Nella sezione Editor di policy, scegli JSON. Immetti la seguente policy. La tua funzione necessita di queste autorizzazioni per leggere da un broker Amazon MQ.

JSON

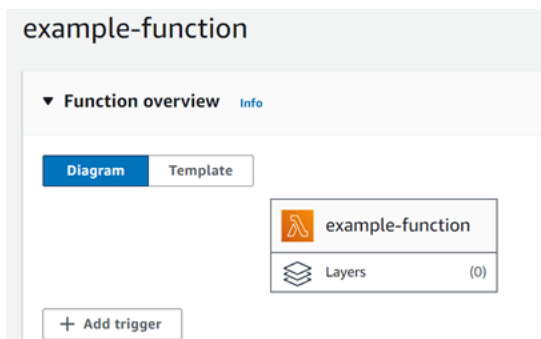
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "mq:DescribeBroker",
        "secretsmanager:GetSecretValue",
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

```
}  
]  
}
```

### Note

Quando utilizzi una chiave gestita dal cliente crittografata, è necessario aggiungere anche l'autorizzazione `kms:Decrypt`.

7. Scegli Next (Successivo). Inserire un nome per la policy, quindi seleziona Crea policy.
8. Torna alla funzione nella console Lambda. In Panoramica delle funzioni, scegliere Aggiungi trigger.



9. Seleziona il tipo di trigger MQ.
10. Configurare le opzioni richieste, quindi scegliere Add (Aggiungi).

Lambda supporta le seguenti opzioni per le origini eventi Amazon MQ:

- Broker MQ – Selezionare un broker Amazon MQ.
- Batch size (Dimensioni batch) – Impostare il numero massimo di messaggi da recuperare in un singolo batch.
- Queue name (Nome della coda) - Immettere la coda Amazon MQ da utilizzare.
- Source access configuration (Configurazione dell'accesso di origine) – Immettere le informazioni sull'host virtuale e il segreto di Secrets Manager in cui sono memorizzate le credenziali del broker.
- Abilita trigger – Disabilitare il trigger per interrompere l'elaborazione dei record.

Per attivare o disattivare il trigger (o eliminarlo), scegliere il trigger MQ nella finestra di progettazione. Per riconfigurare il trigger, utilizzare le operazioni API della mappatura dell'origine eventi.

## Parametri dello strumento di mappatura dell'origine degli eventi

Tutti i tipi di origine eventi Lambda condividono le stesse operazioni API [CreateEventSourceMapping](#) e [UpdateEventSourceMapping](#). Tuttavia, solo alcuni dei parametri si applicano ad Amazon MQ e RabbitMQ.

| Parametro                      | Obbligatorio | Predefinito | Note                                                                                                                                 |
|--------------------------------|--------------|-------------|--------------------------------------------------------------------------------------------------------------------------------------|
| BatchSize                      | N            | 100         | Massimo: 10.000                                                                                                                      |
| Abilitato                      | N            | true        | nessuno                                                                                                                              |
| FunctionName                   | Y            | N/D         | nessuno                                                                                                                              |
| FilterCriteria                 | N            | N/D         | <a href="#">Controllare gli eventi che Lambda invia alla funzione</a>                                                                |
| MaximumBatchingWindowInSeconds | N            | 500 ms      | <a href="#">Comportamento di batching</a>                                                                                            |
| Queues                         | N            | N/D         | Il nome della coda di destinazione del broker Amazon MQ da utilizzare.                                                               |
| SourceAccessConfigurations     | N            | N/D         | Per ActiveMQ, le credenziali BASIC_AUTH. Per RabbitMQ, può contenere sia le credenziali BASIC_AUTH che le informazioni VIRTUAL_HOST. |

## Filtrare gli eventi da una origine eventi Amazon MQ

Puoi utilizzare il filtraggio degli eventi per controllare quali record di un flusso o di una coda Lambda invia alla funzione. Per informazioni generali sul funzionamento del filtraggio eventi, consulta [the section called “Filtro eventi”](#).

In questa sezione viene descritto il filtraggio degli eventi per le origini di eventi Amazon MQ.

### Note

Le mappature delle sorgenti degli eventi di Amazon MQ supportano solo il filtraggio sulla chiave. data

### Argomenti

- [Nozioni di base sul filtraggio degli eventi Amazon MQ](#)

### Nozioni di base sul filtraggio degli eventi Amazon MQ

Supponiamo che la coda di messaggi di Amazon MQ contenga messaggi in formato JSON valido o come stringhe semplici. Un record di esempio sarebbe simile al seguente, con i dati convertiti in una stringa codificata Base64 nel campo data.

### ActiveMQ

```
{
  "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-east-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
  "messageType": "jms/text-message",
  "deliveryMode": 1,
  "replyTo": null,
  "type": null,
  "expiration": "60000",
  "priority": 1,
  "correlationId": "myJMScoID",
  "redelivered": false,
  "destination": {
    "physicalName": "testQueue"
  },
  "data": "QUJD0kFBQUE=",
}
```



```
"timestamp": 1598827811958,
"brokerInTime": 1598827811958,
"brokerOutTime": 1598827811959,
"properties": {
  "index": "1",
  "doAlarm": "false",
  "myCustomProperty": "value"
}
}
```

## RabbitMQ

```
{
  "basicProperties": {
    "contentType": "text/plain",
    "contentEncoding": null,
    "headers": {
      "header1": {
        "bytes": [
          118,
          97,
          108,
          117,
          101,
          49
        ]
      },
      "header2": {
        "bytes": [
          118,
          97,
          108,
          117,
          101,
          50
        ]
      },
      "numberInHeader": 10
    },
    "deliveryMode": 1,
    "priority": 34,
    "correlationId": null,
    "replyTo": null,
```

```

    "expiration": "60000",
    "messageId": null,
    "timestamp": "Jan 1, 1970, 12:33:41 AM",
    "type": null,
    "userId": "AIDACKCEVSQ6C2EXAMPLE",
    "appId": null,
    "clusterId": null,
    "bodySize": 80
  },
  "redelivered": false,
  "data": "eyJ0aW1lb3V0IjowLCJkYXRhIjoiQ1pybWYwR3c4T3Y0YnFMUXhENEUifQ=="
}

```

Per i broker sia Active MQ sia Rabbit MQ, puoi utilizzare il filtraggio degli eventi per filtrare i record utilizzando la chiave data. Supponiamo che la coda Amazon MQ contenga messaggi nel formato JSON seguente.

```

{
  "timeout": 0,
  "IPAddress": "203.0.113.254"
}

```

Per filtrare solo i record in cui il campo timeout è maggiore di 0, l'oggetto `FilterCriteria` sarebbe il seguente.

```

{
  "Filters": [
    {
      "Pattern": "{ \"data\" : { \"timeout\" : [ { \"numeric\": [ \">\",
0] } ] ] } }"
    }
  ]
}

```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```

{
  "data": {
    "timeout": [ { "numeric": [ ">", 0 ] } ]
  }
}

```

```
}
```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

## Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in [Collegamento dei criteri di filtro a una mappatura dell'origine evento \(console\)](#) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }
```

## AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
aws lambda create-event-source-mapping \  
  --function-name my-function \  
  --event-source-arn arn:aws:mq:us-east-2:123456789012:broker:my-  
broker:b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \  
  --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :  
[ { \"numeric\": [ \">\", 0 ] } ] } }"]}]'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \  
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \  
  --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :  
[ { \"numeric\": [ \">\", 0 ] } ] } }"]}]'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \  
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \  
  --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :  
[ { \"numeric\": [ \">\", 0 ] } ] } }"]}]'
```

## AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }'
```

Con Amazon MQ, puoi anche filtrare i record in cui il messaggio è una stringa semplice. Supponiamo di voler elaborare solo i record in cui il messaggio inizia con "Risultato: ". L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"data\" : [ { \"prefix\": \"Result: \" } ] }"
    }
  ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
  "data": [
    {
      "prefix": "Result: "
    }
  ]
}
```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

### Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in [Collegamento dei criteri di filtro a una mappatura dell'origine evento \(console\)](#) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "data" : [ { "prefix": "Result: " } ] }
```

## AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:mq:us-east-2:123456789012:broker:my-  
broker:b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : [ { \"prefix\":  
\"Result: \" } ] }"]}]'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : [ { \"prefix\":  
\"Result: \" } ] }"]}]'
```

## AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : [ { "prefix": "Result " } ] }'
```

I messaggi Amazon MQ devono essere stringhe codificate in UTF-8, semplici o in formato JSON. Questo perché Lambda decodifica gli array di byte Amazon MQ in UTF-8 prima di applicare i criteri di filtraggio. Se i messaggi utilizzano un'altra codifica, ad esempio UTF-16 o ASCII o se il formato del messaggio non corrisponde al formato `FilterCriteria`, Lambda elabora solo i filtri di metadati. La tabella seguente riassume il comportamento specifico:

| Formato messaggio in arrivo  | Formato del modello di filtro per le proprietà di messaggi | Operazione risultante                                                                 |
|------------------------------|------------------------------------------------------------|---------------------------------------------------------------------------------------|
| Stringa normale              | Stringa normale                                            | Filtri Lambda in base ai criteri di filtro.                                           |
| Stringa normale              | Nessun modello di filtro per le proprietà dei dati         | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro. |
| Stringa normale              | JSON valido                                                | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro. |
| JSON valido                  | Stringa normale                                            | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro. |
| JSON valido                  | Nessun modello di filtro per le proprietà dei dati         | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro. |
| JSON valido                  | JSON valido                                                | Filtri Lambda in base ai criteri di filtro.                                           |
| Stringa codificata non UTF-8 | JSON, stringa semplice o nessun modello                    | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro. |

## Risolvere i problemi relativi gli errori di mappatura dell'origine eventi di Amazon MQ

Quando una funzione Lambda rileva un errore irreversibile, il consumatore Amazon MQ arresta l'elaborazione dei record. Tutti gli altri consumatori possono continuare a elaborare, a condizione che non riscontrino lo stesso errore. Per determinare la causa potenziale di un consumatore fermato, controllare il campo `StateTransitionReason` nei dettagli di reso del `EventSourceMapping` per uno dei seguenti codici:

**ESM\_CONFIG\_NOT\_VALID**

La configurazione della mappa dell'origine eventi non è valida.

**EVENT\_SOURCE\_AUTHN\_ERROR**

Lambda non è riuscito ad autenticare l'origine eventi.

**EVENT\_SOURCE\_AUTHZ\_ERROR**

Lambda non dispone delle autorizzazioni necessarie per accedere all'origine eventi.

**FUNCTION\_CONFIG\_NOT\_VALID**

La configurazione della funzione non è valida.

I record non verranno elaborati anche se Lambda li scarta a causa delle loro dimensioni. Il limite di dimensioni per i record Lambda è di 6 MB. Per riconsegnare i messaggi in caso di errore di funzione, è possibile utilizzare una coda di messaggi non instradabili (coda DLQ). Per ulteriori informazioni, consultare [Message Redelivery and DLQ Handling](#) sul sito Web Apache ActiveMQ e [Guida all'affidabilità](#) sul sito Web RabbitMQ.

 **Note**

Lambda non supporta policy di riconsegna personalizzate. Lambda utilizza invece una policy con i valori predefiniti dalla pagina [Policy di riconsegna](#) sul sito Web Apache ActiveMQ, con `maximumRedeliveries` impostato su 6.

## Utilizzo AWS Lambda con Amazon RDS

Puoi collegare una funzione Lambda a un database Amazon Relational Database Service (Amazon RDS) direttamente e attraverso un Server proxy per Amazon RDS. Le connessioni dirette sono utili in scenari semplici, mentre i server proxy sono consigliati per la produzione. Un proxy del database gestisce un pool di connessioni a database condivisi che consente alla funzione di raggiungere elevati livelli di simultaneità senza esaurire le connessioni al database.

Per aiutarti a determinare se utilizzare Amazon RDS o DynamoDB per il tuo carico di lavoro, consulta [the section called “Amazon RDS e DynamoDB”](#)

Consigliamo di utilizzare il Server proxy per Amazon RDS per le funzioni Lambda che effettuano connessioni brevi e frequenti al database o aprono e chiudono numerose connessioni al database. Per ulteriori informazioni, consulta [Connessione automatica di una funzione Lambda e un'istanza DB](#) nella Guida per gli sviluppatori di Amazon Relational Database Service.

### Tip

Per connettere rapidamente una funzione Lambda a un database Amazon RDS, puoi utilizzare la procedura guidata integrata nella console. Per aprire la procedura guidata, esegui queste operazioni:

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Seleziona la funzione a cui desideri connettere un database.
3. Nella scheda Configurazione, seleziona Database RDS.
4. Scegli Connessione al database RDS.

Dopo aver collegato la funzione a un database, scegli Aggiungi proxy se desideri creare un proxy.

## Configurazione della funzione per l'utilizzo con le risorse RDS

Nella console Lambda, puoi eseguire il provisioning, e configurare, delle istanze di database Amazon RDS e risorse proxy. Puoi farlo accedendo ai database RDS nella scheda Configurazione. In alternativa, puoi anche creare e configurare connessioni alle funzioni Lambda nella console Amazon RDS. Quando configuri un'istanza di database RDS da utilizzare con Lambda, tieni presente i seguenti criteri:



- Per effettuare la connessione a un database, la funzione si deve trovare nello stesso Amazon VPC in cui viene eseguito il database.
- Puoi utilizzare i database Amazon RDS con motori MySQL, MariaDB, PostgreSQL o Microsoft SQL Server.
- Puoi anche utilizzare cluster Aurora DB con motori MySQL o PostgreSQL.
- Devi fornire un segreto di Secrets Manager per l'autenticazione del database.
- Un ruolo IAM deve fornire l'autorizzazione all'uso del segreto, mentre una policy di attendibilità deve consentire ad Amazon RDS di assumere il ruolo.
- Il principale IAM che utilizza la console per configurare la risorsa Amazon RDS e connetterla alla tua funzione deve disporre delle seguenti autorizzazioni:

Policy di autorizzazione di esempio

 Note

Le autorizzazioni del server proxy per Amazon RDS sono necessarie solo se configuri un server proxy per Amazon RDS per gestire un pool di connessioni al database.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateSecurityGroup",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:AuthorizeSecurityGroupEgress",
        "ec2:RevokeSecurityGroupEgress",
        "ec2:CreateNetworkInterface",
        "ec2>DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces"
      ]
    }
  ],
}
```

```
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "rds-db:connect",
      "rds:CreateDBProxy",
      "rds:CreateDBInstance",
      "rds:CreateDBSubnetGroup",
      "rds:DescribeDBClusters",
      "rds:DescribeDBInstances",
      "rds:DescribeDBSubnetGroups",
      "rds:DescribeDBProxies",
      "rds:DescribeDBProxyTargets",
      "rds:DescribeDBProxyTargetGroups",
      "rds:RegisterDBProxyTargets",
      "rds:ModifyDBInstance",
      "rds:ModifyDBProxy"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "lambda:CreateFunction",
      "lambda:ListFunctions",
      "lambda:UpdateFunctionConfiguration"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "iam:AttachRolePolicy",
      "iam:CreateRole",
      "iam:CreatePolicy"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "secretsmanager:GetResourcePolicy",
      "secretsmanager:GetSecretValue",
```

```
    "secretsmanager:DescribeSecret",
    "secretsmanager:ListSecretVersionIds",
    "secretsmanager:CreateSecret"
  ],
  "Resource": "*"
}
```

Amazon RDS addebita una tariffa oraria per i proxy in base alla dimensione dell'istanza di database, consulta [Prezzi di Server proxy per RDS](#) per conoscere i dettagli. Per ulteriori informazioni sulle connessioni proxy in generale, consulta [Utilizzo di Server proxy per Amazon RDS](#) nella Guida per l'utente di Amazon RDS.

## SSL/TLS requisiti per le connessioni Amazon RDS

Per effettuare SSL/TLS connessioni sicure a un'istanza di database Amazon RDS, la funzione Lambda deve verificare l'identità del server di database utilizzando un certificato affidabile. Lambda gestisce questi certificati in modo diverso a seconda del tipo di pacchetto di implementazione:

- [Archivi di file .zip](#): la gestione dei certificati varia in base al runtime:
  - Node.js 18 e versioni precedenti: Lambda include automaticamente certificati CA e certificati RDS.
  - Node.js 20 e versioni successive: Lambda non carica più certificati CA aggiuntivi per impostazione predefinita. Impostare la variabile di ambiente `NODE_EXTRA_CA_CERTS` su `/var/runtime/ca-cert.pem`.

Potrebbero essere necessarie fino a 4 settimane prima che i nuovi certificati Amazon RDS vengano aggiunti Regioni AWS ai runtime gestiti di Lambda.

- [Immagini del contenitore](#): le immagini AWS di base includono solo certificati CA. Se la tua funzione si connette a un'istanza del database Amazon RDS, devi includere i certificati appropriati nell'immagine container. Nel tuo Dockerfile, scarica il pacchetto di [certificati corrispondente al Regione AWS luogo in cui ospiti il tuo database](#). Esempio:

```
RUN curl https://truststore.pki.rds.amazonaws.com/us-east-1/us-east-1-bundle.pem -o /  
us-east-1-bundle.pem
```

Questo comando scarica il bundle di certificati Amazon RDS e lo salva nel percorso assoluto `/us-east-1-bundle.pem` nella directory principale del container. Quando configuri la connessione del database nel codice della funzione, devi fare riferimento a questo percorso esatto. Esempio:

## Node.js

La `readFileSync` funzione è necessaria perché i client del Node.js database necessitano del contenuto effettivo del certificato in memoria, non solo del percorso del file del certificato. Senza `readFileSync`, il client interpreta la stringa del percorso come contenuto del certificato, generando un errore di «certificato autofirmato nella catena di certificati».

Example Node.js configurazione della connessione per la funzione OCI

```
import { readFileSync } from 'fs';

// ...

let connectionConfig = {
  host: process.env.ProxyHostName,
  user: process.env.DBUserName,
  password: token,
  database: process.env.DBName,
  ssl: {
    ca: readFileSync('/us-east-1-bundle.pem') // Load RDS certificate content
    from file into memory
  }
};
```

## Python

Example Configurazione della connessione Python per la funzione OCI

```
connection = pymysql.connect(
    host=proxy_host_name,
    user=db_username,
    password=token,
    db=db_name,
    port=port,
    ssl={'ca': '/us-east-1-bundle.pem'} #Path to the certificate in container
)
```

## Java

Per le funzioni Java che utilizzano connessioni JDBC, la stringa di connessione deve includere:

- `useSSL=true`
- `requireSSL=true`
- Un parametro `sslCA` che indica la posizione del certificato Amazon RDS nell'immagine container

Example Stringa di connessione Java per la funzione OCI

```
// Define connection string
String connectionString = String.format("jdbc:mysql://%s:%s/%s?
useSSL=true&requireSSL=true&sslCA=/us-east-1-bundle.pem", // Path to the certificate
in container
    System.getenv("ProxyHostName"),
    System.getenv("Port"),
    System.getenv("DBName"));
```

## .NET

Example Stringa di connessione.NET per la connessione MySQL nella funzione OCI

```
/// Build the Connection String with the Token
string connectionString =
    $"Server={Environment.GetEnvironmentVariable("RDS_ENDPOINT")};" +
        $"Port={Environment.GetEnvironmentVariable("RDS_PORT")};" +

    $"Uid={Environment.GetEnvironmentVariable("RDS_USERNAME")};" +
        $"Pwd={authToken};" +
        "SslMode=Required;" +
        "SslCa=/us-east-1-bundle.pem"; // Path to the certificate
in container
```

## Go

Per le funzioni Go che utilizzano connessioni MySQL, carica il certificato Amazon RDS in un pool di certificati e registralo con il driver MySQL. La stringa di connessione deve quindi fare riferimento a questa configurazione utilizzando il parametro `tls`.

## Example Codice Go per la connessione MySQL nella funzione OCI

```
import (
    "crypto/tls"
    "crypto/x509"
    "os"
    "github.com/go-sql-driver/mysql"
)

...

// Create certificate pool and register TLS config
rootCertPool := x509.NewCertPool()
pem, err := os.ReadFile("/us-east-1-bundle.pem") // Path to the certificate in
container
if err != nil {
    panic("failed to read certificate file: " + err.Error())
}
if ok := rootCertPool.AppendCertsFromPEM(pem); !ok {
    panic("failed to append PEM")
}

mysql.RegisterTLSConfig("custom", &tls.Config{
    RootCAs: rootCertPool,
})

dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?allowCleartextPasswords=true&tls=custom",
    dbUser, authenticationToken, dbEndpoint, dbName,
)
```

## Ruby

### Example Configurazione della connessione Ruby per la funzione OCI

```
conn = Mysql2::Client.new(
  host: endpoint,
  username: user,
  password: token,
  port: port,
  database: db_name,
  sslca: '/us-east-1-bundle.pem', # Path to the certificate in container
  sslverify: true
)
```

## Connessione a un database Amazon RDS in una funzione Lambda

Gli esempi di codice seguenti mostrano come implementare una funzione Lambda che si connette a un database Amazon RDS. La funzione effettua una semplice richiesta al database e restituisce il risultato.

### Note

Questi esempi di codice sono validi solo per i [pacchetti di implementazione .zip](#). Se stai implementando la funzione utilizzando un'[immagine container](#), devi specificare il file del certificato Amazon RDS nel codice della funzione, come spiegato nella [sezione precedente](#).

## .NET

### SDK per .NET

### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Connessione a un database Amazon RDS in una funzione Lambda con .NET.

```
using System.Data;
using System.Text.Json;
using Amazon.Lambda.APIGatewayEvents;
using Amazon.Lambda.Core;
using MySql.Data.MySqlClient;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace aws_rds;

public class InputModel
{
    public string key1 { get; set; }
```

```

        public string key2 { get; set; }
    }

    public class Function
    {
        /// <summary>
        /// Handles the Lambda function execution for connecting to RDS using IAM
        authentication.
        /// </summary>
        /// <param name="input">The input event data passed to the Lambda function</
param>
        /// <param name="context">The Lambda execution context that provides runtime
information</param>
        /// <returns>A response object containing the execution result</returns>

        public async Task<APIGatewayProxyResponse>
FunctionHandler(APIGatewayProxyRequest request, ILambdaContext context)
        {
            // Sample Input: {"body": "{\"key1\":\"20\", \"key2\":\"25\"}"}
            var input = JsonSerializer.Deserialize<InputModel>(request.Body);

            /// Obtain authentication token
            var authToken = RDSAuthTokenGenerator.GenerateAuthToken(
                Environment.GetEnvironmentVariable("RDS_ENDPOINT"),
                Convert.ToInt32(Environment.GetEnvironmentVariable("RDS_PORT")),
                Environment.GetEnvironmentVariable("RDS_USERNAME")
            );

            /// Build the Connection String with the Token
            string connectionString =
$"Server={Environment.GetEnvironmentVariable("RDS_ENDPOINT")};" +

$"Port={Environment.GetEnvironmentVariable("RDS_PORT")};" +

$"Uid={Environment.GetEnvironmentVariable("RDS_USERNAME")};" +
                $"Pwd={authToken};";

            try
            {
                await using var connection = new MySqlConnection(connectionString);
                await connection.OpenAsync();

                const string sql = "SELECT @param1 + @param2 AS Sum";
            }
        }
    }

```



```

        await using var command = new MySqlCommand(sql, connection);
        command.Parameters.AddWithValue("@param1", int.Parse(input.key1 ??
"0"));
        command.Parameters.AddWithValue("@param2", int.Parse(input.key2 ??
"0"));

        await using var reader = await command.ExecuteReaderAsync();
        if (await reader.ReadAsync())
        {
            int result = reader.GetInt32("Sum");

            //Sample Response: {"statusCode":200,"body":{"\message\":"The
sum is: 45\"},"isBase64Encoded":false}
            return new APIGatewayProxyResponse
            {
                StatusCode = 200,
                Body = JsonSerializer.Serialize(new { message = $"The sum is:
{result}" })
            };
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }

        return new APIGatewayProxyResponse
        {
            StatusCode = 500,
            Body = JsonSerializer.Serialize(new { error = "Internal server
error" })
        };
    }
}

```

## Go

### SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite Go.

```
/*
Golang v2 code here.
*/

package main

import (
    "context"
    "database/sql"
    "encoding/json"
    "fmt"
    "os"

    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
    _ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
    Name string `json:"name"`
}

func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

    var dbName string = os.Getenv("DatabaseName")
    var dbUser string = os.Getenv("DatabaseUser")
    var dbHost string = os.Getenv("DBHost") // Add hostname without https
    var dbPort int = os.Getenv("Port")      // Add port number
    var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
    var region string = os.Getenv("AWS_REGION")
```

```
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic("configuration error: " + err.Error())
}

authenticationToken, err := auth.BuildAuthToken(
    context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
if err != nil {
    panic("failed to create authentication token: " + err.Error())
}

dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
    dbUser, authenticationToken, dbEndpoint, dbName,
)

db, err := sql.Open("mysql", dsn)
if err != nil {
    panic(err)
}

defer db.Close()

var sum int
err = db.QueryRow("SELECT ?+? AS sum", 3, 2).Scan(&sum)
if err != nil {
    panic(err)
}
s := fmt.Sprint(sum)
message := fmt.Sprintf("The selected sum is: %s", s)

messageBytes, err := json.Marshal(message)
if err != nil {
    return nil, err
}

messageString := string(messageBytes)
return map[string]interface{}{
    "statusCode": 200,
    "headers":    map[string]string{"Content-Type": "application/json"},
    "body":       messageString,
}, nil
}
```

```
func main() {  
    lambda.Start(HandleRequest)  
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite Java.

```
import com.amazonaws.services.lambda.runtime.Context;  
import com.amazonaws.services.lambda.runtime.RequestHandler;  
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;  
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;  
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;  
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.rdsdata.RdsDataClient;  
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementRequest;  
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementResponse;  
import software.amazon.awssdk.services.rdsdata.model.Field;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
  
public class RdsLambdaHandler implements  
    RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {  
  
    @Override  
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent  
event, Context context) {  
        APIGatewayProxyResponseEvent response = new  
APIGatewayProxyResponseEvent();  
    }  
}
```

```

    try {
        // Obtain auth token
        String token = createAuthToken();

        // Define connection configuration
        String connectionString = String.format("jdbc:mysql://%s:%s/%s?
useSSL=true&requireSSL=true",
            System.getenv("ProxyHostName"),
            System.getenv("Port"),
            System.getenv("DBName"));

        // Establish a connection to the database
        try (Connection connection =
            DriverManager.getConnection(connectionString, System.getenv("DBUserName"),
            token);

            PreparedStatement statement =
            connection.prepareStatement("SELECT ? + ? AS sum")) {

            statement.setInt(1, 3);
            statement.setInt(2, 2);

            try (ResultSet resultSet = statement.executeQuery()) {
                if (resultSet.next()) {
                    int sum = resultSet.getInt("sum");
                    response.setStatusCode(200);
                    response.setBody("The selected sum is: " + sum);
                }
            }
        }

    } catch (Exception e) {
        response.setStatusCode(500);
        response.setBody("Error: " + e.getMessage());
    }

    return response;
}

private String createAuthToken() {
    // Create RDS Data Service client
    RdsDataClient rdsDataClient = RdsDataClient.builder()
        .region(Region.of(System.getenv("AWS_REGION")))
        .credentialsProvider(DefaultCredentialsProvider.create())
        .build();

```

```

    // Define authentication request
    ExecuteStatementRequest request = ExecuteStatementRequest.builder()
        .resourceArn(System.getenv("ProxyHostName"))
        .secretArn(System.getenv("DBUserName"))
        .database(System.getenv("DBName"))
        .sql("SELECT 'RDS IAM Authentication'")
        .build();

    // Execute request and obtain authentication token
    ExecuteStatementResponse response =
rdsDataClient.executeStatement(request);
    Field tokenField = response.records().get(0).get(0);

    return tokenField.stringValue();
}
}

```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda utilizzando JavaScript

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
    // Define connection authentication parameters

```

```
const dbinfo = {

  hostname: process.env.ProxyHostName,
  port: process.env.Port,
  username: process.env.DBUserName,
  region: process.env.AWS_REGION,

}

// Create RDS Signer object
const signer = new Signer(dbinfo);

// Request authorization token from RDS, specifying the username
const token = await signer.getAuthToken();
return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
}
```

```
}  
};
```

## Connessione a un database Amazon RDS in una funzione Lambda utilizzando TypeScript

```
import { Signer } from "@aws-sdk/rds-signer";  
import mysql from 'mysql2/promise';  
  
// RDS settings  
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that  
// the DB settings are not null or undefined,  
const proxy_host_name = process.env.PROXY_HOST_NAME!  
const port = parseInt(process.env.PORT!)  
const db_name = process.env.DB_NAME!  
const db_user_name = process.env.DB_USER_NAME!  
const aws_region = process.env.AWS_REGION!  
  
async function createAuthToken(): Promise<string> {  
  
    // Create RDS Signer object  
    const signer = new Signer({  
        hostname: proxy_host_name,  
        port: port,  
        region: aws_region,  
        username: db_user_name  
    });  
  
    // Request authorization token from RDS, specifying the username  
    const token = await signer.getAuthToken();  
    return token;  
}  
  
async function dbOps(): Promise<mysql.QueryResult | undefined> {  
    try {  
        // Obtain auth token  
        const token = await createAuthToken();  
        const conn = await mysql.createConnection({  
            host: proxy_host_name,  
            user: db_user_name,  
            password: token,  

```



```
        database: db_name,
        ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
    });
    const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
    console.log('result:', rows);
    return rows;
}
catch (err) {
    console.log(err);
}
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number;
body: string }> => {
    // Execute database flow
    const result = await dbOps();

    // Return error if result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

## PHP

### SDK per PHP

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite PHP.

```
<?php
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    private function getAuthToken(): string {
        // Define connection authentication parameters
        $dbConnection = [
            'hostname' => getenv('DB_HOSTNAME'),
            'port' => getenv('DB_PORT'),
            'username' => getenv('DB_USERNAME'),
            'region' => getenv('AWS_REGION'),
        ];

        // Create RDS AuthTokenGenerator object
        $generator = new
AuthTokenGenerator(CredentialProvider::defaultProvider());

        // Request authorization token from RDS, specifying the username
        return $generator->createToken(
            $dbConnection['hostname'] . ':' . $dbConnection['port'],
            $dbConnection['region'],
            $dbConnection['username']
        );
    }
}
```

```

private function getQueryResults() {
    // Obtain auth token
    $token = $this->getAuthToken();

    // Define connection configuration
    $connectionConfig = [
        'host' => getenv('DB_HOSTNAME'),
        'user' => getenv('DB_USERNAME'),
        'password' => $token,
        'database' => getenv('DB_NAME'),
    ];

    // Create the connection to the DB
    $conn = new PDO(
        "mysql:host={$connectionConfig['host']};dbname={$connectionConfig['database']}",
        $connectionConfig['user'],
        $connectionConfig['password'],
        [
            PDO::MYSQL_ATTR_SSL_CA => '/path/to/rds-ca-2019-root.pem',
            PDO::MYSQL_ATTR_SSL_VERIFY_SERVER_CERT => true,
        ]
    );

    // Obtain the result of the query
    $stmt = $conn->prepare('SELECT ?+? AS sum');
    $stmt->execute([3, 2]);

    return $stmt->fetch(PDO::FETCH_ASSOC);
}

/**
 * @param mixed $event
 * @param Context $context
 * @return array
 */
public function handle(mixed $event, Context $context): array
{
    $this->logger->info("Processing query");

    // Execute database flow
    $result = $this->getQueryResults();

```

```
        return [
            'sum' => $result['sum']
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite Python.

```
import json
import os
import boto3
import pymysql

# RDS settings
proxy_host_name = os.environ['PROXY_HOST_NAME']
port = int(os.environ['PORT'])
db_name = os.environ['DB_NAME']
db_user_name = os.environ['DB_USER_NAME']
aws_region = os.environ['AWS_REGION']

# Fetch RDS Auth Token
def get_auth_token():
    client = boto3.client('rds')
    token = client.generate_db_auth_token(
        DBHostname=proxy_host_name,
        Port=port
        DBUsername=db_user_name
        Region=aws_region
```

```
)
return token

def lambda_handler(event, context):
    token = get_auth_token()
    try:
        connection = pymysql.connect(
            host=proxy_host_name,
            user=db_user_name,
            password=token,
            db=db_name,
            port=port,
            ssl={'ca': 'Amazon RDS'} # Ensure you have the CA bundle for SSL
connection
        )

        with connection.cursor() as cursor:
            cursor.execute('SELECT %s + %s AS sum', (3, 2))
            result = cursor.fetchone()

        return result

    except Exception as e:
        return (f"Error: {str(e)}") # Return an error message if an exception
occurs
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite Ruby.

```
# Ruby code here.

require 'aws-sdk-rds'
```

```
require 'json'
require 'mysql2'

def lambda_handler(event:, context:)
  endpoint = ENV['DBEndpoint'] # Add the endpoint without https"
  port = ENV['Port']           # 3306
  user = ENV['DBUser']
  region = ENV['DBRegion']     # 'us-east-1'
  db_name = ENV['DBName']

  credentials = Aws::Credentials.new(
    ENV['AWS_ACCESS_KEY_ID'],
    ENV['AWS_SECRET_ACCESS_KEY'],
    ENV['AWS_SESSION_TOKEN']
  )
  rds_client = Aws::RDS::AuthTokenGenerator.new(
    region: region,
    credentials: credentials
  )

  token = rds_client.auth_token(
    endpoint: endpoint+ ':' + port,
    user_name: user,
    region: region
  )

  begin
    conn = Mysql2::Client.new(
      host: endpoint,
      username: user,
      password: token,
      port: port,
      database: db_name,
      sslca: '/var/task/global-bundle.pem',
      sslverify: true,
      enable_cleartext_plugin: true
    )
    a = 3
    b = 2
    result = conn.query("SELECT #{a} + #{b} AS sum").first['sum']
    puts result
    conn.close
  {
    statusCode: 200,
```

```

        body: result.to_json
      }
      rescue => e
        puts "Database connection failed due to #{e}"
      end
    end
  end
end

```

## Rust

### SDK per Rust

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite Rust.

```

use aws_config::BehaviorVersion;
use aws_credential_types::provider::ProvideCredentials;
use aws_sigv4::{
    http_request::{sign, SignableBody, SignableRequest, SigningSettings},
    sign::v4,
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
use sqlx::postgres::PgConnectOptions;
use std::env;
use std::time::{Duration, SystemTime};

const RDS_CERTS: &[u8] = include_bytes!("global-bundle.pem");

async fn generate_rds_iam_token(
    db_hostname: &str,
    port: u16,
    db_username: &str,
) -> Result<String, Error> {
    let config = aws_config::load_defaults(BehaviorVersion::v2024_03_28()).await;

    let credentials = config
        .credentials_provider()

```

```

        .expect("no credentials provider found")
        .provide_credentials()
        .await
        .expect("unable to load credentials");
let identity = credentials.into();
let region = config.region().unwrap().to_string();

let mut signing_settings = SigningSettings::default();
signing_settings.expires_in = Some(Duration::from_secs(900));
signing_settings.signature_location =
aws_sigv4::http_request::SignatureLocation::QueryParams;

let signing_params = v4::SigningParams::builder()
    .identity(&identity)
    .region(&region)
    .name("rds-db")
    .time(SystemTime::now())
    .settings(signing_settings)
    .build()?;

let url = format!(
    "https://{db_hostname}:{port}/?Action=connect&DBUser={db_user}",
    db_hostname = db_hostname,
    port = port,
    db_user = db_username
);

let signable_request =
    SignableRequest::new("GET", &url, std::iter::empty(),
SignableBody::Bytes(&[]))
        .expect("signable request");

let (signing_instructions, _signature) =
    sign(signable_request, &signing_params.into())?.into_parts();

let mut url = url::Url::parse(&url).unwrap();
for (name, value) in signing_instructions.params() {
    url.query_pairs_mut().append_pair(name, &value);
}

let response = url.to_string().split_off("https://".len());

Ok(response)
}

```



```

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(handler)).await
}

async fn handler(_event: LambdaEvent<Value>) -> Result<Value, Error> {
    let db_host = env::var("DB_HOSTNAME").expect("DB_HOSTNAME must be set");
    let db_port = env::var("DB_PORT")
        .expect("DB_PORT must be set")
        .parse::<u16>()
        .expect("PORT must be a valid number");
    let db_name = env::var("DB_NAME").expect("DB_NAME must be set");
    let db_user_name = env::var("DB_USERNAME").expect("DB_USERNAME must be set");

    let token = generate_rds_iam_token(&db_host, db_port, &db_user_name).await?;

    let opts = PgConnectOptions::new()
        .host(&db_host)
        .port(db_port)
        .username(&db_user_name)
        .password(&token)
        .database(&db_name)
        .ssl_root_cert_from_pem(RDS_CERTS.to_vec())
        .ssl_mode(sqlx::postgres::PgSslMode::Require);

    let pool = sqlx::postgres::PgPoolOptions::new()
        .connect_with(opts)
        .await?;

    let result: i32 = sqlx::query_scalar("SELECT $1 + $2")
        .bind(3)
        .bind(2)
        .fetch_one(&pool)
        .await?;

    println!("Result: {:?}", result);

    Ok(json!({
        "statusCode": 200,
        "content-type": "text/plain",
        "body": format!("The selected sum is: {result}")
    })))
}

```

## Elaborazione di notifiche di eventi da Amazon RDS

Puoi utilizzare Lambda per elaborare le notifiche degli eventi da un database Amazon RDS. Amazon RDS invia le notifiche a un argomento Amazon Simple Notification Service (Amazon SNS), che puoi configurare per richiamare una funzione Lambda. Amazon SNS avvolge il messaggio proveniente da Amazon RDS nel proprio documento evento e lo invia alla tua funzione.

Per ulteriori informazioni sulla configurazione di un database Amazon RDS per l'invio di notifiche, consulta [Utilizzo delle notifiche di eventi di Amazon RDS](#).

### Example Messaggio Amazon RDS in un evento Amazon SNS

```
{
  "Records": [
    {
      "EventVersion": "1.0",
      "EventSubscriptionArn": "arn:aws:sns:us-east-2:123456789012:rds-
lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
      "EventSource": "aws:sns",
      "Sns": {
        "SignatureVersion": "1",
        "Timestamp": "2023-01-02T12:45:07.000Z",
        "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEKai6RibDsvpi
+tE/1+82j...65r==",
        "SigningCertUrl": "https://sns.us-east-2.amazonaws.com/
SimpleNotificationService-ac565b8b1a6c5d002d285f9598aa1d9b.pem",
        "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
        "Message": "{\"Event Source\":\"db-instance\",\"Event Time\":\"2023-01-02
12:45:06.000\",\"Identifier Link\":\"https://console.aws.amazon.com/rds/home?
region=eu-west-1#dbinstance:id=dbinstanceid\",\"Source ID\":\"dbinstanceid\",\"Event ID
\":\"http://docs.amazonwebservices.com/AmazonRDS/latest/UserGuide/USER_Events.html#RDS-
EVENT-0002\",\"Event Message\":\"Finished DB Instance backup\"}",
        "MessageAttributes": {},
        "Type": "Notification",
        "UnsubscribeUrl": "https://sns.us-east-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-2:123456789012:test-
lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
        "TopicArn": "arn:aws:sns:us-east-2:123456789012:sns-lambda",
        "Subject": "RDS Notification Message"
      }
    ]
  }
```

```
}  
]  
}
```

## Completare il tutorial su Lambda e Amazon RDS

- [Uso di una funzione Lambda per accedere a un database Amazon RDS](#): seguendo la Guida per l'utente di Amazon RDS, impara come utilizzare una funzione Lambda per scrivere dati su un database Amazon RDS attraverso un Server proxy per Amazon RDS. La tua funzione Lambda legge i record da una coda Amazon SQS e scrive nuovi elementi in una tabella del tuo database ogni volta che viene aggiunto un messaggio.

## Seleziona un servizio di database per Lambda-based le tue applicazioni

Molte applicazioni serverless devono archiviare e recuperare i dati. AWS offre diverse opzioni di database che funzionano con le funzioni Lambda. Due delle scelte più popolari sono Amazon DynamoDB, un servizio di database NoSQL, e Amazon RDS, una soluzione di database relazionale tradizionale. Le sezioni seguenti spiegano le principali differenze tra questi servizi quando vengono utilizzati con Lambda e ti aiutano a selezionare il servizio di database giusto per la tua applicazione serverless.

Per saperne di più sugli altri servizi di database offerti da AWS e per comprenderne i casi d'uso e i compromessi più in generale, vedi [Scelta di un AWS servizio di database](#). Tutti i servizi di AWS database sono compatibili con Lambda, ma non tutti potrebbero essere adatti al tuo caso d'uso particolare.

Quali sono le tue scelte quando selezioni un servizio di database con Lambda?

AWS offre diversi servizi di database. Per le applicazioni serverless, due delle scelte più popolari sono DynamoDB e Amazon RDS.

- DynamoDB è un servizio di database NoSQL completamente gestito ottimizzato per le applicazioni serverless. Fornisce una scalabilità ottimale e prestazioni costanti in millisecondi su qualsiasi scala.
- Amazon RDS è un servizio di database relazionale gestito che supporta più motori di database tra cui MySQL e PostgreSQL. Fornisce funzionalità SQL familiari con un'infrastruttura gestita.

Suggerimenti se conosci già le tue esigenze

Se hai già le idee chiare sulle tue esigenze, ecco i nostri suggerimenti di base:

Consigliamo [DynamoDB](#) per applicazioni serverless che richiedono prestazioni costanti a bassa latenza, scalabilità automatica e non richiedono join o transazioni complesse. È particolarmente adatto per Lambda-based le applicazioni grazie alla sua natura serverless.

[Amazon RDS](#) è la scelta migliore quando hai bisogno di query SQL complesse, join o se hai applicazioni esistenti che utilizzano database relazionali. Tuttavia, tieni presente che la connessione delle funzioni Lambda ad Amazon RDS richiede una configurazione aggiuntiva e può influire sui tempi di avvio a freddo.

Aspetti da tenere in considerazione selezioni un servizio di database

Quando scegli tra DynamoDB e Amazon RDS per le tue applicazioni Lambda, considera questi fattori:

- Gestione delle connessioni e avvii a freddo
- Modelli di accesso ai dati
- Complessità delle query
- Requisiti di coerenza dei dati
- Caratteristiche di scalabilità
- Modello di costi

Se conosci questi fattori, puoi selezionare l'opzione che soddisfa al meglio le esigenze del caso d'uso specifico.

Gestione delle connessioni e avvii a freddo

- DynamoDB utilizza un'API HTTP per tutte le operazioni. Le funzioni Lambda possono effettuare richieste immediate senza mantenere le connessioni, con conseguente miglioramento delle prestazioni di avvio a freddo. Ogni richiesta viene autenticata utilizzando le AWS credenziali senza sovraccarico di connessione.
- Amazon RDS richiede la gestione dei pool di connessioni poiché utilizza connessioni di database tradizionali. Ciò può influire sugli avvii a freddo poiché le nuove istanze Lambda devono stabilire connessioni. Dovrai implementare strategie di pool di connessioni e potenzialmente utilizzare il [Server proxy per Amazon RDS](#) per gestire le connessioni in modo efficace. Tieni presente che l'utilizzo del Server proxy per Amazon RDS comporta costi aggiuntivi.

## Modelli di accesso ai dati

- DynamoDB funziona al meglio con modelli di accesso noti e design a tabella singola. È ideale per applicazioni Lambda che richiedono un accesso coerente a bassa latenza ai dati basato su chiavi primarie o indici secondari.
- Amazon RDS offre flessibilità per query complesse e modelli di accesso in evoluzione. È più adatto quando le funzioni Lambda devono eseguire query uniche e personalizzate o join complessi su più tabelle.

## Complessità delle query

- DynamoDB eccelle nelle operazioni semplici basate su chiavi e nei modelli di accesso predefiniti. Le query complesse devono essere progettate sulla base di strutture indicizzate e i join devono essere gestiti nel codice dell'applicazione.
- Amazon RDS supporta query SQL complesse con join, sottoquery e aggregazioni. Questo può semplificare il codice della funzione Lambda quando sono necessarie operazioni complesse sui dati.

## Requisiti di coerenza dei dati

- DynamoDB offre opzioni di coerenza sia finali che avanzate, con una forte coerenza disponibile per le letture di singoli elementi. Le transazioni sono supportate ma con alcune limitazioni.
- Amazon RDS garantisce la conformità completa di atomicità, coerenza, isolamento e durabilità (ACID) e supporto per transazioni complesse. Se le tue funzioni Lambda richiedono transazioni complesse o una forte coerenza tra più record, Amazon RDS potrebbe essere più adatto.

## Caratteristiche di scalabilità

- DynamoDB si adatta automaticamente al carico di lavoro. È in grado di gestire picchi improvvisi di traffico provenienti da funzioni Lambda senza pre-provisioning. Puoi utilizzare la modalità di capacità su richiesta per pagare solo per ciò che usi, in linea con il modello di scalabilità di Lambda.
- Amazon RDS ha una capacità fissa in base alla dimensione dell'istanza scelta. Se più funzioni Lambda tentano di connettersi contemporaneamente, potresti superare la tua quota di connessione. È necessario gestire con attenzione i pool di connessioni e potenzialmente implementare la logica dei tentativi.

## Modello di costi

- I prezzi di DynamoDB si allineano bene con le applicazioni serverless. Con la capacità su richiesta, paghi solo per le letture e le scritture effettive eseguite dalle tue funzioni Lambda. Non sono previsti addebiti per i tempi di inattività.
- Amazon RDS addebita i costi per l'istanza in esecuzione indipendentemente dall'utilizzo. Questo può essere meno conveniente per i carichi di lavoro sporadici che possono essere tipici delle applicazioni serverless. Tuttavia, potrebbe essere più economico per carichi di lavoro ad alto throughput con un utilizzo costante.

## Nozioni di base sulle istanze del database da te scelte

Ora che hai letto i criteri per scegliere tra DynamoDB and Amazon RDS e le principali differenze tra questi, puoi selezionare l'opzione più adatta alle tue esigenze e utilizzare le seguenti risorse per iniziare a utilizzarla.

### DynamoDB

Iniziare a usare DynamoDB con le seguenti risorse

- Per un'introduzione al servizio DynamoDB, leggi [Cos'è DynamoDB?](#) nella Guida per gli sviluppatori di Amazon DynamoDB.
- Segui il tutorial [Uso di Lambda con API Gateway](#) per vedere un esempio di utilizzo di una funzione Lambda per eseguire operazioni CRUD su una tabella DynamoDB in risposta a una richiesta API.
- Leggi [Programming with DynamoDB and the AWS SDK](#) nella Amazon DynamoDB Developer Guide per saperne di più su come accedere a DynamoDB dall'interno della tua funzione Lambda utilizzando uno degli SDK. AWS

### Amazon RDS

Iniziare a usare Amazon RDS con le seguenti risorse

- Per un'introduzione al servizio Amazon RDS, leggi [Cos'è Amazon Relational Database Service \(Amazon RDS\)?](#) nella Guida per l'utente di Amazon Relational Database Service.
- Segui il tutorial [Uso di una funzione Lambda per accedere a un database Amazon RDS](#) nella Guida per l'utente di Amazon Relational Database Service.

- Per ulteriori informazioni sull'uso di Lambda con Amazon RDS, consulta [the section called “RDS”](#).

## Elaborare le notifiche di eventi Amazon S3 con Lambda

È possibile utilizzare Lambda per elaborare le [notifiche degli eventi](#) da Amazon Simple Storage Service. Amazon S3 può inviare un evento a una funzione Lambda quando un oggetto viene creato o eliminato. È possibile configurare le impostazioni di notifica su un bucket e concedere ad Amazon S3 l'autorizzazione a invocare una funzione sulla policy di autorizzazione basata sulle risorse della funzione.

### Warning

Se la funzione Lambda utilizza lo stesso bucket che la attiva, potrebbe causare l'esecuzione della funzione in loop. Ad esempio, se il bucket attiva una funzione ogni volta che un oggetto viene caricato e la funzione carica un oggetto nel bucket, allora la funzione indirettamente lo attiva. Per evitare questo, utilizzare due bucket, oppure configurare il trigger in modo che venga applicato solo a un prefisso utilizzato per gli oggetti in entrata.

Amazon S3 richiama la funzione [in modo asincrono](#) con un evento che contiene dettagli sull'oggetto. L'esempio seguente mostra un evento che Amazon S3 ha inviato quando un pacchetto di distribuzione è stato caricato su Amazon S3.

### Example Evento di notifica Amazon S3

```
{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-2",
      "eventTime": "2019-09-03T19:37:27.192Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AWS:AIDAINPONIXQXHT3IKHL2"
      },
      "requestParameters": {
        "sourceIPAddress": "205.255.255.255"
      },
      "responseElements": {
        "x-amz-request-id": "D82B88E5F771F645",
        "x-amz-id-2":
"v1R7PnpV2Ce81l0PRw6jlUpck7Jo5ZsQjryTjK1c5aLWGVHPZLj5NeC6qMa0emYBDX0o6QBU0Wo="
      }
    }
  ]
}
```



```

    },
    "s3": {
      "s3SchemaVersion": "1.0",
      "configurationId": "828aa6fc-f7b5-4305-8584-487c791949c1",
      "bucket": {
        "name": "amzn-s3-demo-bucket",
        "ownerIdentity": {
          "principalId": "A3I5XTEXAMAI3E"
        },
        "arn": "arn:aws:s3:::lambda-artifacts-deafc19498e3f2df"
      },
      "object": {
        "key": "b21b84d653bb07b05b1e6b33684dc11b",
        "size": 1305107,
        "eTag": "b21b84d653bb07b05b1e6b33684dc11b",
        "sequencer": "0C0F6F405D6ED209E1"
      }
    }
  }
]
}

```

Per richiamare la funzione, Amazon S3 necessita dell'autorizzazione dalla [policy basata su risorse](#) della funzione. Quando si configura un trigger Amazon S3 nella console Lambda, la console modifica la policy basata su risorse per consentire ad Amazon S3 di richiamare la funzione se il nome del bucket e l'ID account corrispondono. Se si configura la notifica in Amazon S3, si utilizza l'API Lambda per aggiornare la policy. È inoltre possibile utilizzare l'API Lambda per concedere l'autorizzazione a un altro account o limitare l'autorizzazione a un alias designato.

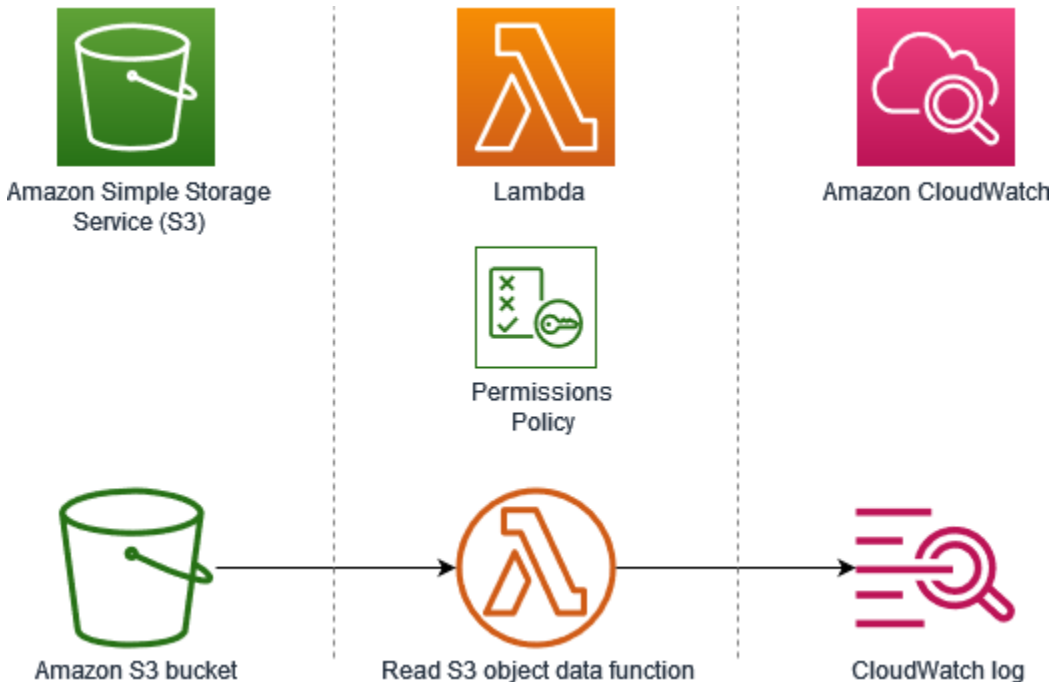
[Se la tua funzione utilizza l' AWS SDK per gestire le risorse Amazon S3, necessita anche delle autorizzazioni Amazon S3 nel suo ruolo di esecuzione.](#)

## Argomenti

- [Tutorial: uso di un trigger Amazon S3 per richiamare una funzione Lambda](#)
- [Tutorial: uso di un trigger Amazon S3 per creare immagini in miniatura](#)

## Tutorial: uso di un trigger Amazon S3 per richiamare una funzione Lambda

In questo tutorial si utilizzerà la console per creare una funzione Lambda e configurare un trigger per un bucket Amazon Simple Storage Service (Amazon S3). Ogni volta che aggiungi un oggetto al tuo bucket Amazon S3, la funzione viene eseguita e invia il tipo di oggetto ad Amazon Logs. CloudWatch

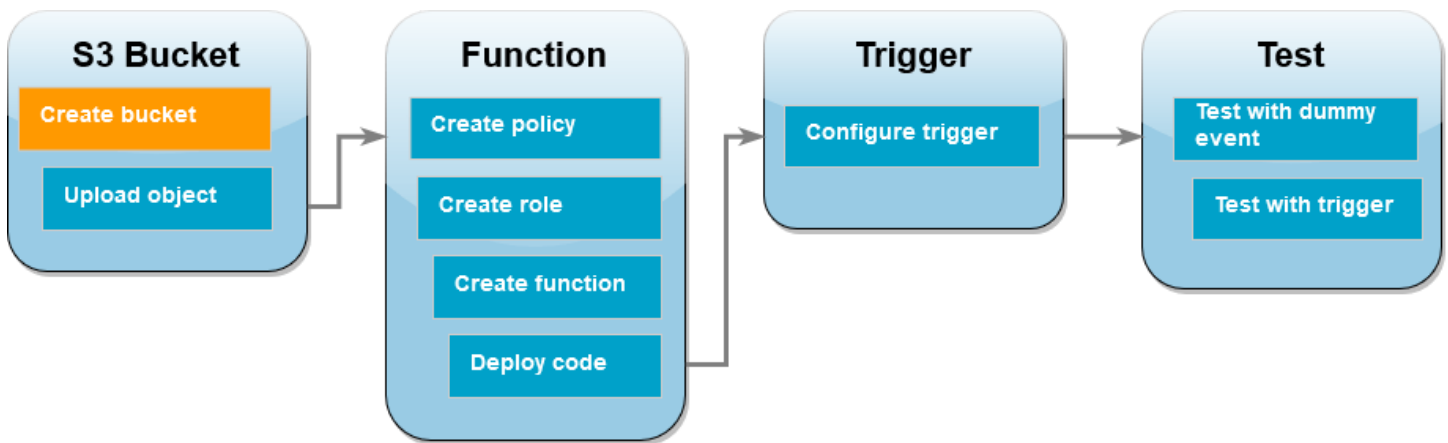


Questo tutorial dimostra come:

1. Crea un bucket Amazon S3.
2. Crea una funzione Lambda che restituisce il tipo di oggetto in un bucket Amazon S3.
3. Configura un trigger Lambda che richiami la tua funzione quando gli oggetti vengono caricati nel bucket.
4. Prova la tua funzione, prima con un evento fittizio e poi usando il trigger.

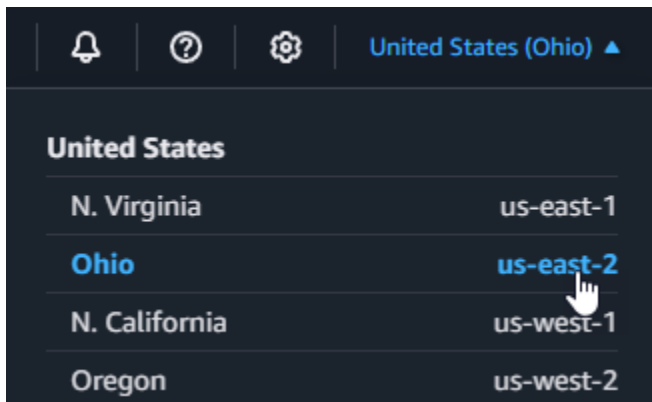
Completando questi passaggi, imparerai come configurare una funzione Lambda da eseguire ogni volta che vengono aggiunti o eliminati oggetti da un bucket Amazon S3. Puoi completare questo tutorial soltanto dalla Console di gestione AWS.

## Crea un bucket Amazon S3



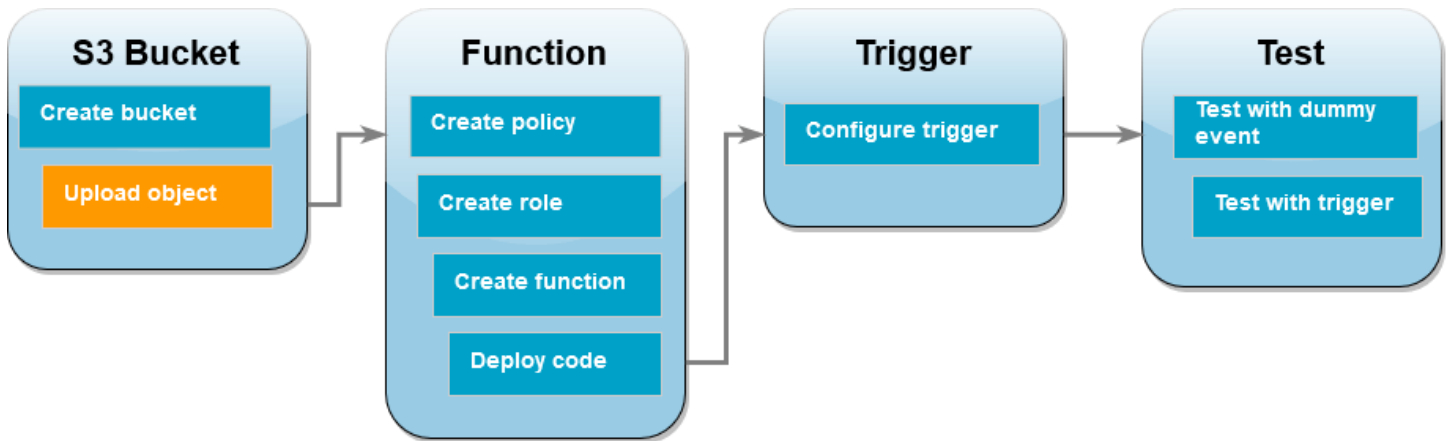
Come creare un bucket Amazon S3.

1. Apri la console [Amazon S3](#) e seleziona la pagina General purpose buckets.
2. Seleziona quella più Regione AWS vicina alla tua posizione geografica. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo. Più avanti nel tutorial, è necessario creare la funzione Lambda nella stessa regione.



3. Scegliere Create bucket (Crea bucket).
4. In General configuration (Configurazione generale), eseguire le operazioni seguenti:
  - a. Per il tipo di secchio, assicurati che sia selezionato Uso generale.
  - b. Per Nome del bucket, inserisci un nome univoco globale che soddisfi le [regole di denominazione dei bucket](#) di Amazon S3. I nomi dei bucket possono contenere solo lettere minuscole, numeri, punti (.) e trattini (-).
5. Lascia tutte le altre opzioni impostate sui valori predefiniti e scegli Crea bucket.

## Caricamento di un oggetto di test in un bucket

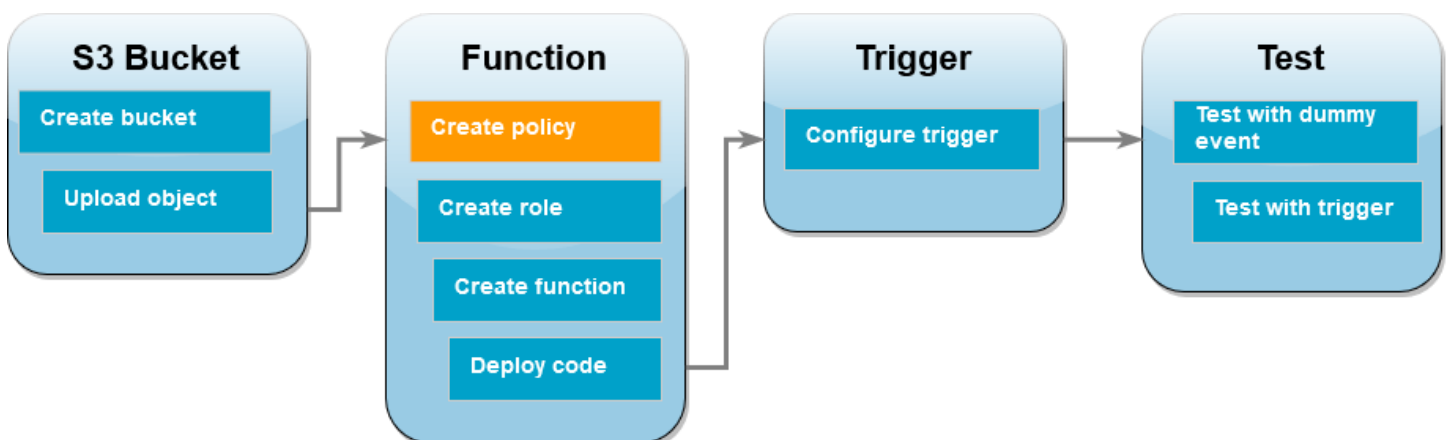


## Caricamento di un oggetto di test

1. Apri la pagina [Bucket](#) della console Amazon S3 e scegli il bucket che hai creato durante il passaggio precedente.
2. Scegli Carica.
3. Scegli Aggiungi file e seleziona l'oggetto da caricare. È possibile selezionare qualsiasi file (ad esempio, HappyFace .jpg).
4. Seleziona Apri, quindi Carica.

Più avanti nel tutorial, testerai la tua funzione Lambda usando questo oggetto.

## Creazione di una policy di autorizzazione



Crea una policy di autorizzazioni che consenta a Lambda di ottenere oggetti da un bucket Amazon S3 e di scrivere su Amazon Logs. CloudWatch

## Come creare la policy

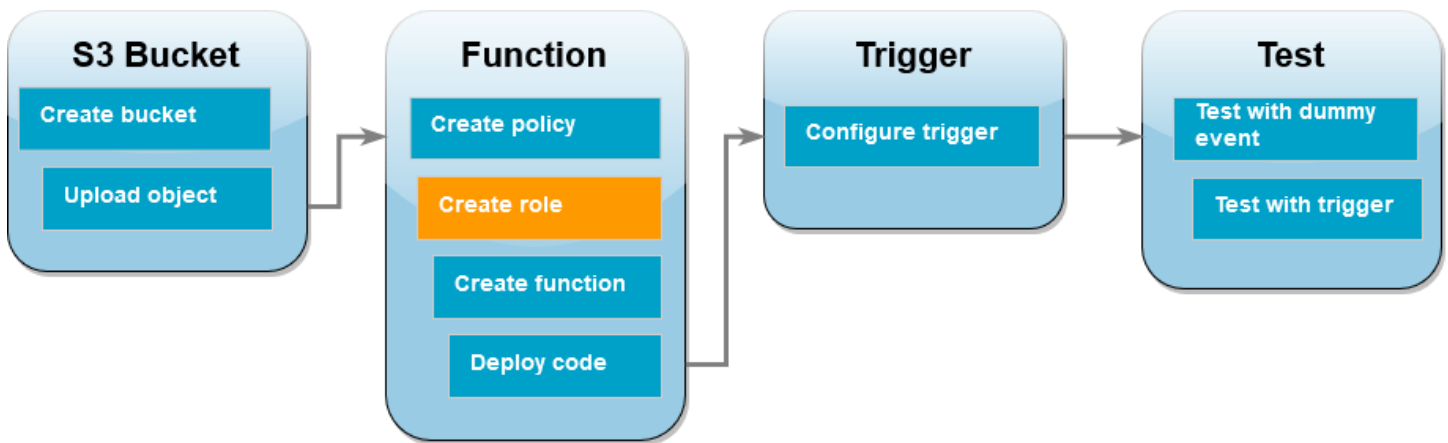
1. Apri la pagina [Policies \(Policy\)](#) nella console IAM.
2. Scegli Crea policy.
3. Scegliere la scheda JSON e quindi incollare la seguente policy personalizzata nell'editor JSON.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:CreateLogStream"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::*/*"
    }
  ]
}
```

4. Scegliere Next: Tags (Successivo: Tag).
5. Scegliere Next: Review (Successivo: Rivedi).
6. In Rivedi policy, per Nome della policy inserisci **s3-trigger-tutorial**.
7. Scegli Crea policy.

## Creazione di un ruolo di esecuzione

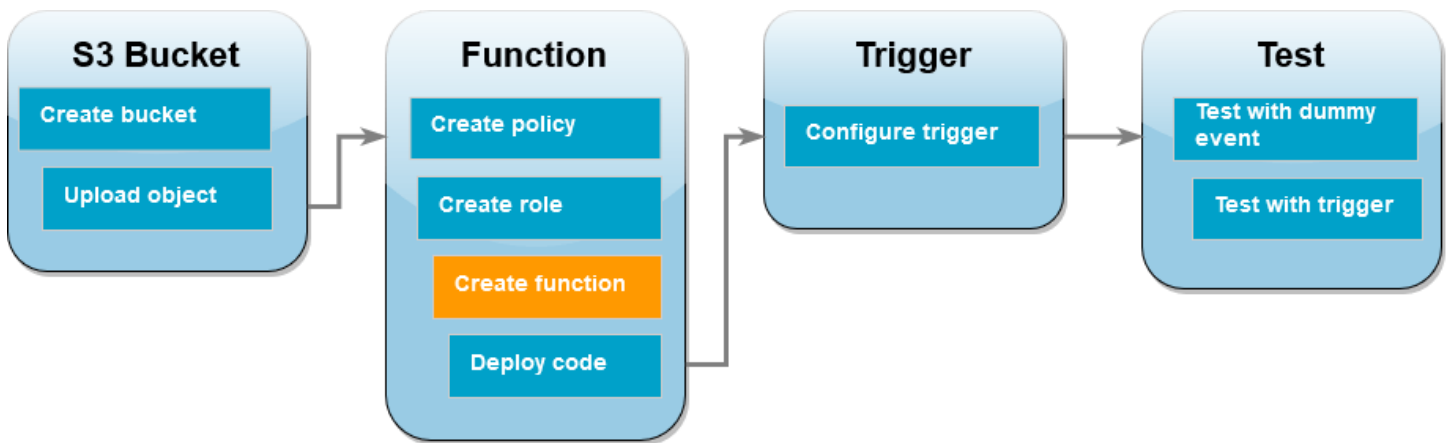


Un ruolo di [esecuzione](#) è un ruolo AWS Identity and Access Management (IAM) che concede a una funzione Lambda l'autorizzazione all'accesso Servizi AWS e alle risorse. In questa fase, creare un ruolo di esecuzione utilizzando la policy di autorizzazioni creata nel passaggio precedente.

Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione personalizzata

1. Aprire la [pagina Roles \(Ruoli\)](#) della console IAM.
2. Scegliere Crea ruolo.
3. Per il tipo di entità attendibile, scegli Servizio AWS , quindi per il caso d'uso seleziona Lambda.
4. Scegli Avanti.
5. Nella casella di ricerca delle policy, immettere **s3-trigger-tutorial**.
6. Nei risultati della ricerca, seleziona la policy creata (s3-trigger-tutorial), quindi scegli Next (Successivo).
7. In Role details (Dettagli del ruolo), per Role name (Nome del ruolo), specifica **lambda-s3-trigger-role**, quindi scegli Create role (Crea ruolo).

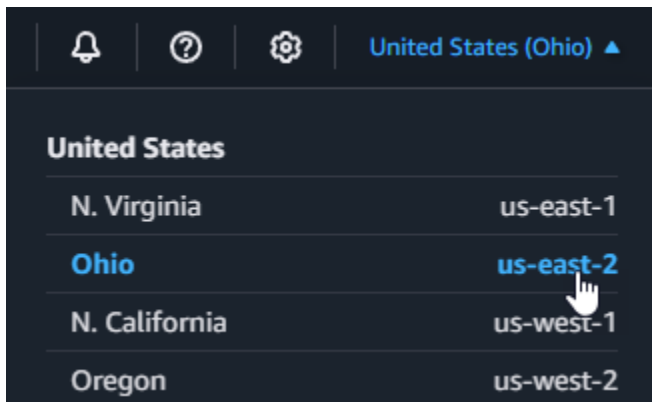
## Creazione della funzione Lambda



Crea una funzione Lambda nella console usando il runtime Python 3.14.

### Creazione della funzione Lambda

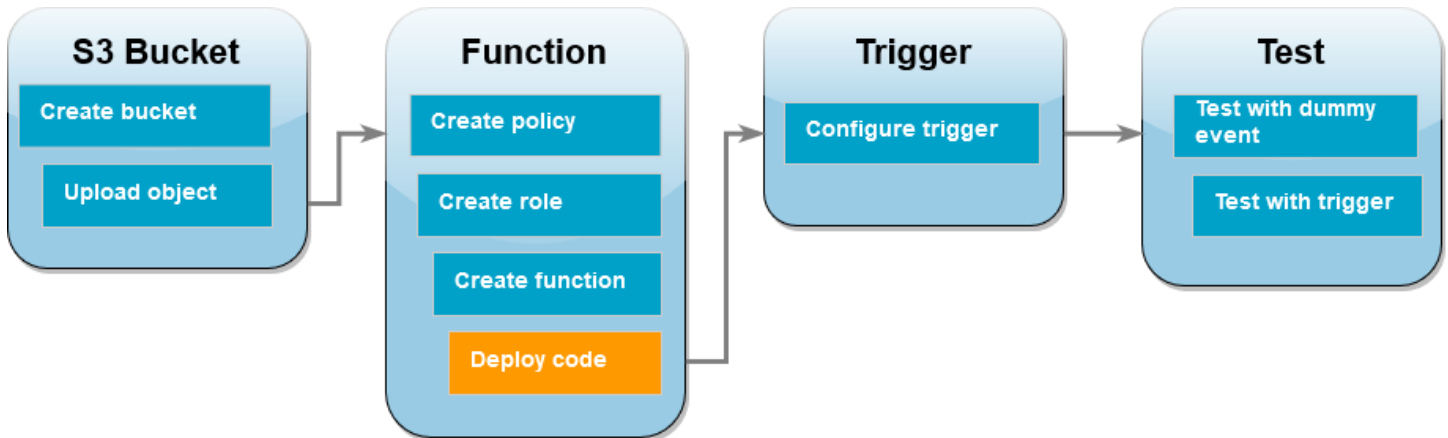
1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Assicurati di lavorare nello stesso ambiente in cui Regione AWS hai creato il bucket Amazon S3. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo.



3. Scegli Crea funzione.
4. Scegli Crea da zero.
5. In Basic information (Informazioni di base) eseguire queste operazioni:
  - a. Nel campo Nome funzione, inserisci `s3-trigger-tutorial`.
  - b. Per Runtime, scegli Python 3.14.
  - c. In Architecture (Architettura), scegli `x86_64`.
6. Nella scheda Modifica ruolo di esecuzione predefinito, effettua le seguenti operazioni:

- a. Espandi la scheda, quindi scegli Utilizza un ruolo esistente.
  - b. Seleziona il `lambda-s3-trigger-role` che hai creato in precedenza.
7. Scegli Crea funzione.

#### Implementazione del codice della funzione



Questo tutorial utilizza il runtime Python 3.14, ma abbiamo anche fornito file di codice di esempio per altri runtime. Puoi selezionare la scheda nella casella seguente per visualizzare il codice per il runtime che ti interessa.

La funzione Lambda recupera il nome della chiave dell'oggetto caricato e il nome del bucket dal parametro event che riceve da Amazon S3. La funzione utilizza quindi il [metodo](#) `get_object` di AWS SDK per Python (Boto3) per recuperare i metadati dell'oggetto, incluso il tipo di contenuto (tipo MIME) dell'oggetto caricato.

#### Implementazione del codice della funzione

1. Scegli la scheda Python nella casella seguente e copia il codice.

.NET

SDK per .NET

#### Note

C'è altro su. [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).



## Utilizzo di un evento S3 con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be
// converted into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
            }
        }
    }
}
```

```
        var key =
            HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

        context.Logger.LogLine($"Request is for {bucket} and {key}");

        var objectResult = await _s3Client.GetObjectAsync(bucket,
            key);

        context.Logger.LogLine($"Returning {objectResult.Key}");

        return objectResult.Key;
    }
    catch (Exception e)
    {
        context.Logger.LogLine($"Error processing request -
            {e.Message}");

        return string.Empty;
    }
}
```

Go

SDK per Go V2

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
```

```
"log"

"github.com/aws/aws-lambda-go/events"
"github.com/aws/aws-lambda-go/lambda"
"github.com/aws/aws-sdk-go-v2/config"
"github.com/aws/aws-sdk-go-v2/service/s3"
)

func handler(ctx context.Context, s3Event events.S3Event) error {
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        log.Printf("failed to load default config: %s", err)
        return err
    }
    s3Client := s3.NewFromConfig(sdkConfig)

    for _, record := range s3Event.Records {
        bucket := record.S3.Bucket.Name
        key := record.S3.Object.URLDecodedKey
        headOutput, err := s3Client.HeadObject(ctx, &s3.HeadObjectInput{
            Bucket: &bucket,
            Key:     &key,
        })
        if err != nil {
            log.Printf("error getting head of object %s/%s: %s", bucket, key, err)
            return err
        }
        log.Printf("successfully retrieved %s/%s of type %s", bucket, key,
            *headOutput.ContentType)
    }

    return nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Utilizzo di un evento S3 con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import software.amazon.awssdk.services.s3.model.HeadObjectRequest;
import software.amazon.awssdk.services.s3.model.HeadObjectResponse;
import software.amazon.awssdk.services.s3.S3Client;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;
import
    com.amazonaws.services.lambda.runtime.events.models.s3.S3EventNotification.S3EventNo

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Handler implements RequestHandler<S3Event, String> {
    private static final Logger logger =
        LoggerFactory.getLogger(Handler.class);
    @Override
    public String handleRequest(S3Event s3event, Context context) {
        try {
            S3EventNotificationRecord record = s3event.getRecords().get(0);
            String srcBucket = record.getS3().getBucket().getName();
            String srcKey = record.getS3().getObject().getUrlDecodedKey();

            S3Client s3Client = S3Client.builder().build();
```

```

        HeadObjectResponse headObject = getHeadObject(s3Client, srcBucket,
srcKey);

        logger.info("Successfully retrieved " + srcBucket + "/" + srcKey +
" of type " + headObject.contentType());

        return "Ok";
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}

private HeadObjectResponse getHeadObject(S3Client s3Client, String
bucket, String key) {
    HeadObjectRequest headObjectRequest = HeadObjectRequest.builder()
        .bucket(bucket)
        .key(key)
        .build();
    return s3Client.headObject(headObjectRequest);
}
}

```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Consumare un evento S3 utilizzando Lambda. JavaScript

```

import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

export const handler = async (event, context) => {

    // Get the object from the event and show its content type

```

```

    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));

    try {
      const { ContentType } = await client.send(new HeadObjectCommand({
        Bucket: bucket,
        Key: key,
      }));

      console.log('CONTENT TYPE:', ContentType);
      return ContentType;

    } catch (err) {
      console.log(err);
      const message = `Error getting object ${key} from bucket ${bucket}.
      Make sure they exist and your bucket is in the same region as this
      function.`;
      console.log(message);
      throw new Error(message);
    }
  };
};

```

## Consumare un evento S3 con Lambda utilizzando TypeScript

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> =>
{
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
};

```

```
try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
} catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make
sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
}
};
```

## PHP

### SDK per PHP

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite PHP.

```
<?php

use Bref\Context\Context;
use Bref\Event\S3\S3Event;
use Bref\Event\S3\S3Handler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends S3Handler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }
}
```

```

    }

    public function handleS3(S3Event $event, Context $context) : void
    {
        $this->logger->info("Processing S3 records");

        // Get the object from the event and show its content type
        $records = $event->getRecords();

        foreach ($records as $record)
        {
            $bucket = $record->getBucket()->getName();
            $key = urldecode($record->getObject()->getKey());

            try {
                $fileSize = urldecode($record->getObject()->getSize());
                echo "File Size: " . $fileSize . "\n";
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                echo $e->getMessage() . "\n";
                echo 'Error getting object ' . $key . ' from bucket ' .
                $bucket . '. Make sure they exist and your bucket is in the same region as
                this function.' . "\n";
                throw $e;
            }
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);

```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).



## Utilizzo di un evento S3 con Lambda tramite Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import json
import urllib.parse
import boto3

print('Loading function')

s3 = boto3.client('s3')

def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))

    # Get the object from the event and show its content type
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']
['key'], encoding='utf-8')
    try:
        response = s3.get_object(Bucket=bucket, Key=key)
        print("CONTENT TYPE: " + response['ContentType'])
        return response['ContentType']
    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. Make sure they
exist and your bucket is in the same region as this function.'.format(key,
bucket))
        raise e
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite Ruby.

```
require 'json'
require 'uri'
require 'aws-sdk'

puts 'Loading function'

def lambda_handler(event:, context:)
  s3 = Aws::S3::Client.new(region: 'region') # Your AWS region
  # puts "Received event: #{JSON.dump(event)}"

  # Get the object from the event and show its content type
  bucket = event['Records'][0]['s3']['bucket']['name']
  key = URI.decode_www_form_component(event['Records'][0]['s3']['object']
['key'], Encoding::UTF_8)
  begin
    response = s3.get_object(bucket: bucket, key: key)
    puts "CONTENT TYPE: #{response.content_type}"
    return response.content_type
  rescue StandardError => e
    puts e.message
    puts "Error getting object #{key} from bucket #{bucket}. Make sure they
exist and your bucket is in the same region as this function."
    raise e
  end
end
```

## Rust

### SDK per Rust

#### Note

C'è altro su GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Utilizzo di un evento S3 con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::s3::S3Event;
use aws_sdk_s3::{Client};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Main function
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    // Initialize the AWS SDK for Rust
    let config = aws_config::load_from_env().await;
    let s3_client = Client::new(&config);

    let res = run(service_fn(|request: LambdaEvent<S3Event>| {
        function_handler(&s3_client, request)
    })).await;

    res
}

async fn function_handler(
    s3_client: &Client,
```

```

    evt: LambdaEvent<S3Event>
) -> Result<(), Error> {
    tracing::info!(records = ?evt.payload.records.len(), "Received request
from SQS");

    if evt.payload.records.len() == 0 {
        tracing::info!("Empty S3 event received");
    }

    let bucket =
evt.payload.records[0].s3.bucket.name.as_ref().expect("Bucket name to
exist");
    let key = evt.payload.records[0].s3.object.key.as_ref().expect("Object
key to exist");

    tracing::info!("Request is for {} and object {}", bucket, key);

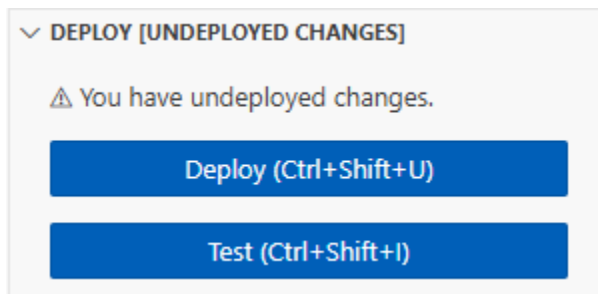
    let s3_get_object_result = s3_client
        .get_object()
        .bucket(bucket)
        .key(key)
        .send()
        .await;

    match s3_get_object_result {
        Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
contains a 'body' property of type ByteStream"),
        Err(_) => tracing::info!("Failure with S3 Get Object request")
    }

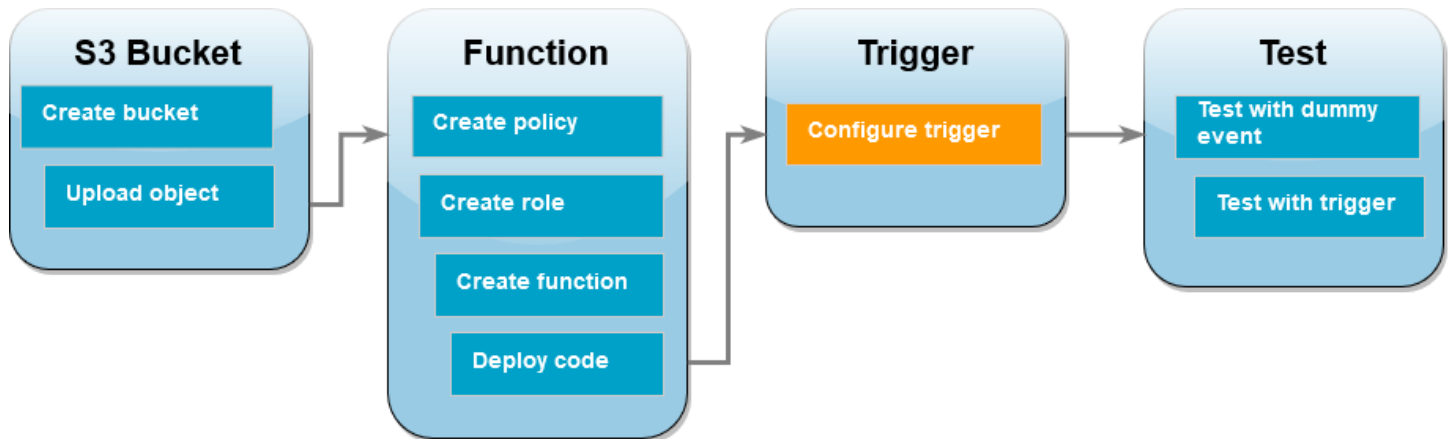
    Ok(())
}

```

2. Nel riquadro Codice sorgente della console Lambda, incolla il codice nell'editor di codice, sostituendo il codice creato da Lambda.
3. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:

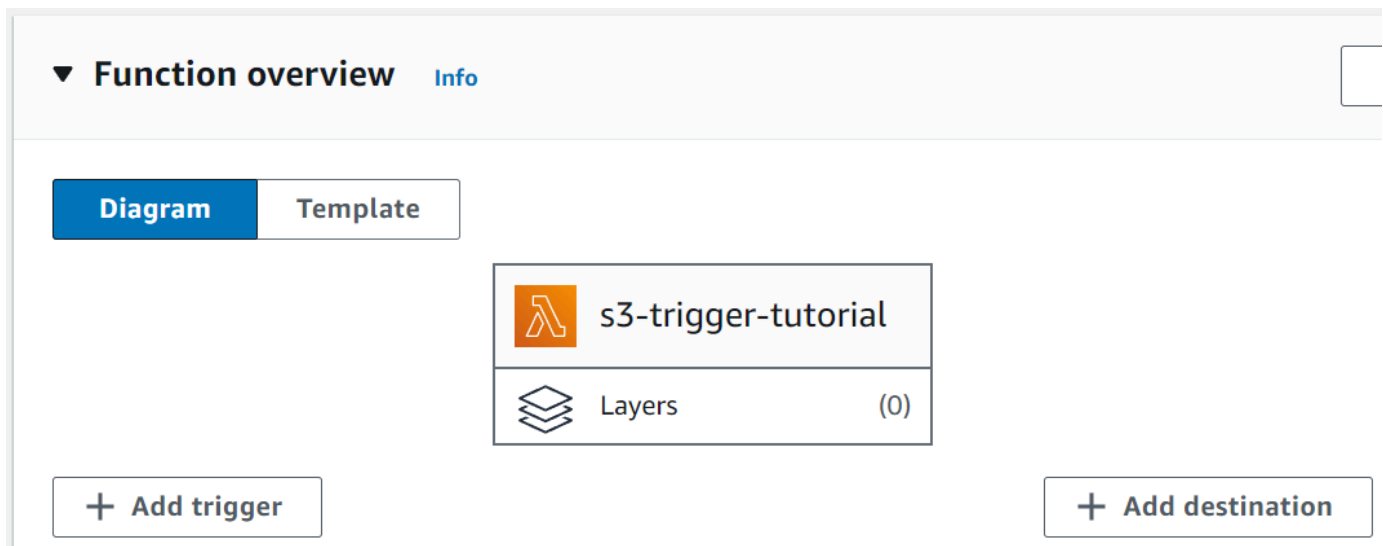


### Creazione del trigger Amazon S3



### Creazione del trigger Amazon S3

1. Nel riquadro Panoramica della funzione, scegli Aggiungi trigger.



2. Seleziona S3.
3. In Bucket, seleziona il bucket che hai creato in precedenza nel tutorial.

4. In Tipi di eventi, verifica che sia selezionata l'opzione Tutti gli eventi di creazione di oggetti.
5. In Invocazione ricorsiva, seleziona la casella di controllo per confermare che non è consigliabile utilizzare lo stesso bucket Amazon S3 per input e output.
6. Scegliere Aggiungi.

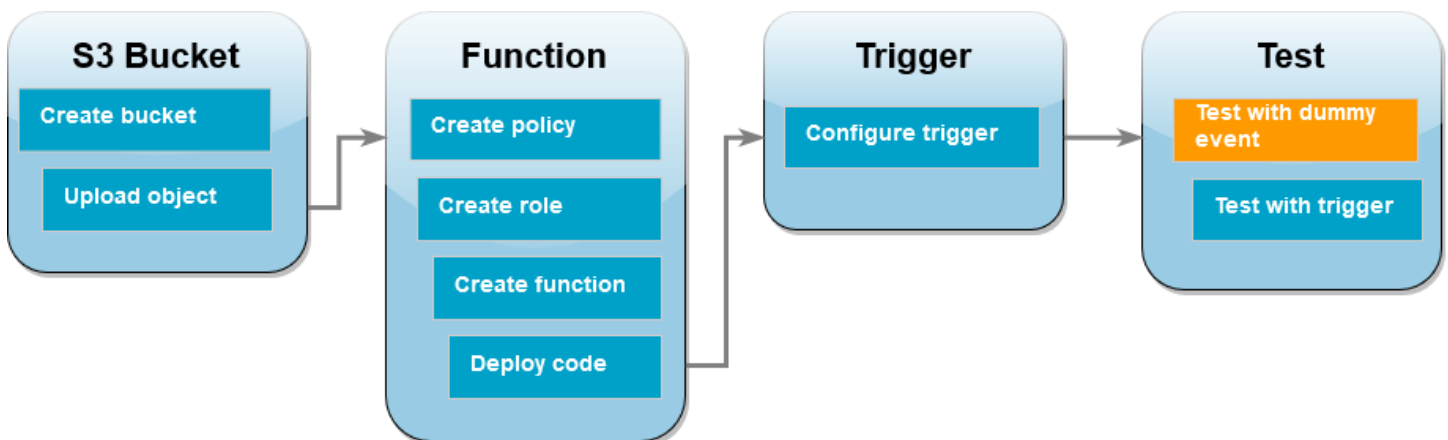
#### Note

Quando crei un trigger Amazon S3 per una funzione Lambda utilizzando la console Lambda, Amazon S3 configura una [notifica degli eventi](#) sul bucket specificato. Prima di configurare questa notifica di evento, Amazon S3 esegue una serie di controlli per confermare che la destinazione dell'evento esista e disponga delle policy IAM richieste. Amazon S3 esegue questi test anche su qualsiasi altra notifica di eventi configurata per quel bucket. A causa di questo controllo, se il bucket ha precedentemente configurato destinazioni di eventi per risorse che non esistono più o per risorse che non dispongono delle policy di autorizzazione richieste, Amazon S3 non sarà in grado di creare la nuova notifica di evento. Verrà visualizzato il seguente messaggio di errore che indica che non è stato possibile creare il trigger:

An error occurred when creating the trigger: Unable to validate the following destination configurations.

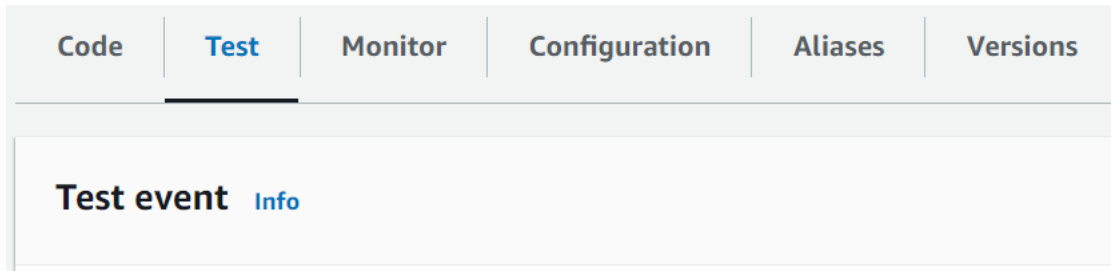
Puoi visualizzare questo errore se in precedenza hai configurato un trigger per un'altra funzione Lambda utilizzando lo stesso bucket e da allora hai eliminato la funzione o modificato le sue policy di autorizzazioni.

#### Test di una funzione Lambda con un evento fittizio



## Test della funzione Lambda con un evento fittizio

1. Nella pagina della console Lambda della funzione, seleziona la scheda Test.



2. Per Event name (Nome evento) immettere MyTestEvent.
3. Nella sezione JSON dell'evento, incolla il seguente evento di test. Assicurati di sostituire i seguenti valori:
  - Sostituisci `us-east-1` con la regione in cui è stato creato il bucket Amazon S3.
  - Sostituisci entrambe le istanze di `amzn-s3-demo-bucket` con il nome del bucket Amazon S3.
  - Sostituisci `test%2FKey` con il nome dell'oggetto di test che hai caricato in precedenza nel tuo bucket (ad esempio, `HappyFace.jpg`).

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
```

```

    "s3SchemaVersion": "1.0",
    "configurationId": "testConfigRule",
    "bucket": {
      "name": "amzn-s3-demo-bucket",
      "ownerIdentity": {
        "principalId": "EXAMPLE"
      },
      "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
    },
    "object": {
      "key": "test%2Fkey",
      "size": 1024,
      "eTag": "0123456789abcdef0123456789abcdef",
      "sequencer": "0A1B2C3D4E5F678901"
    }
  }
}
]
}

```

4. Selezionare Salva.
5. Scegli Test (Esegui test).
6. Se la funzione viene eseguita correttamente, vedrai un risultato simile al seguente nella scheda Risultati di esecuzione.

#### Response

"image/jpeg"

#### Function Logs

```

START RequestId: 12b3cae7-5f4e-415e-93e6-416b8f8b66e6 Version: $LATEST
2021-02-18T21:40:59.280Z    12b3cae7-5f4e-415e-93e6-416b8f8b66e6    INFO    INPUT
    BUCKET AND KEY:  { Bucket: 'amzn-s3-demo-bucket', Key: 'HappyFace.jpg' }
2021-02-18T21:41:00.215Z    12b3cae7-5f4e-415e-93e6-416b8f8b66e6    INFO    CONTENT
    TYPE: image/jpeg
END RequestId: 12b3cae7-5f4e-415e-93e6-416b8f8b66e6
REPORT RequestId: 12b3cae7-5f4e-415e-93e6-416b8f8b66e6    Duration: 976.25 ms
    Billed Duration: 977 ms    Memory Size: 128 MB    Max Memory Used: 90 MB    Init
    Duration: 430.47 ms

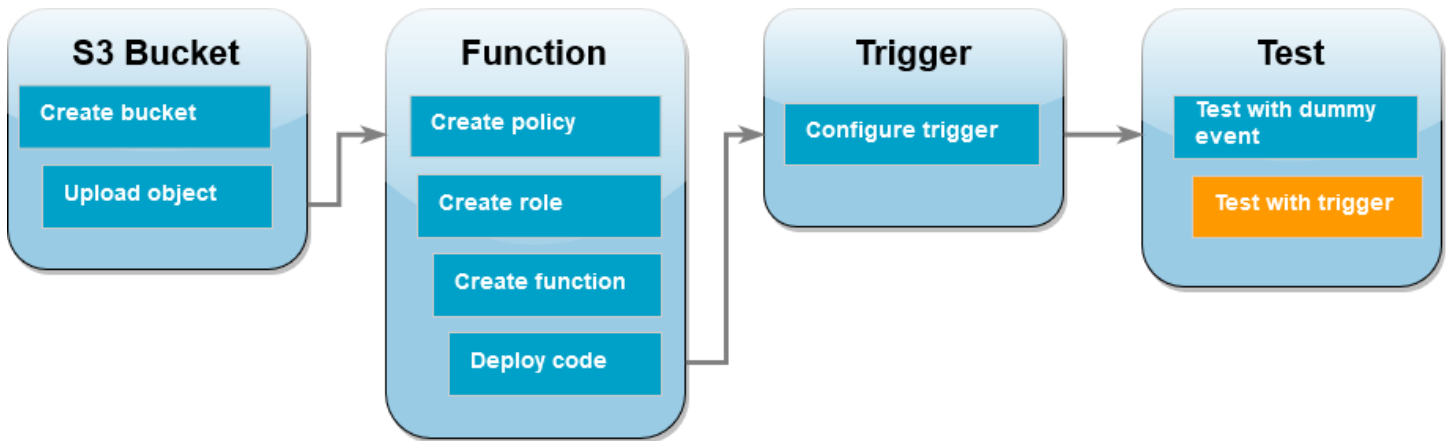
```

#### Request ID

12b3cae7-5f4e-415e-93e6-416b8f8b66e6



## Test della funzione Lambda mediante il trigger Amazon S3



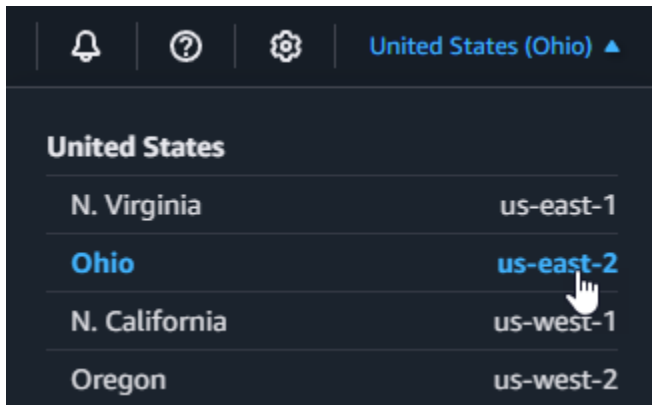
Per verificare la funzione con il trigger configurato, carica un oggetto sul bucket Amazon S3 utilizzando la console. Per verificare che la funzione Lambda abbia funzionato come previsto, utilizza CloudWatch Logs per visualizzare l'output della funzione.

### Caricamento di un oggetto nel bucket Amazon S3

1. Apri la pagina [Bucket](#) della console Amazon S3 e scegli il bucket che hai creato in precedenza.
2. Scegli Carica.
3. Scegli Aggiungi file e usa il selettore di file per scegliere un oggetto da caricare. Tale oggetto può essere qualsiasi file desideri.
4. Seleziona Apri, quindi Carica.

### Per verificare l'invocazione della funzione utilizzando Logs CloudWatch

1. Aprire la console [CloudWatch](#).
2. Assicurati di lavorare nello stesso modo in cui Regione AWS hai creato la tua funzione Lambda. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo.



3. Scegli Log e quindi Gruppi di log.
4. Scegli il nome del gruppo di log per la funzione (/aws/lambda/s3-trigger-tutorial).
5. In Flussi di log, scegli il flusso di log più recente.
6. Se la tua funzione è stata richiamata correttamente in risposta al trigger di Amazon S3, vedrai un risultato simile al seguente. Il CONTENT TYPE visualizzato dipende dal tipo di file che hai caricato nel bucket.

```
2022-05-09T23:17:28.702Z 0cae7f5a-b0af-4c73-8563-a3430333cc10 INFO CONTENT  
TYPE: image/jpeg
```

## Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti inutili addebiti al tuo Account AWS

### Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **confirm** nel campo di immissione del testo, quindi scegli Elimina.

### Come eliminare il ruolo di esecuzione

1. Aprire la pagina [Ruoli](#) della console IAM.
2. Selezionare il ruolo di esecuzione creato.

3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare il bucket S3

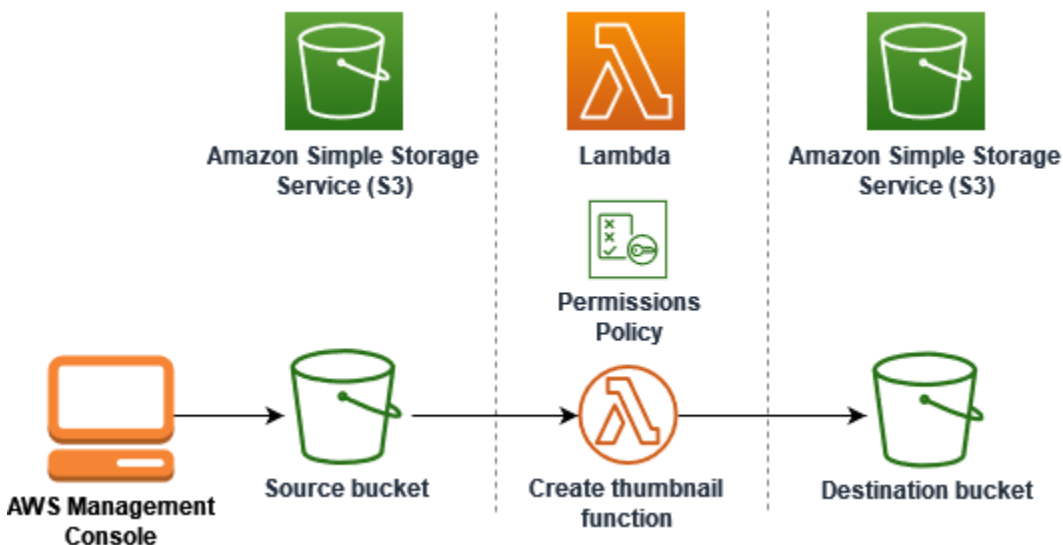
1. Aprire la [console Amazon S3](#).
2. Selezionare il bucket creato in precedenza.
3. Scegliere Elimina.
4. Inserisci il nome del bucket nel campo di immissione testo.
5. Scegli Delete Bucket (Elimina bucket).

Fasi successive

In [Tutorial: uso di un trigger Amazon S3 per creare immagini in miniatura](#), il trigger Amazon S3 richiama una funzione per creare un'immagine in miniatura per ogni file immagine caricato nel bucket. Questo tutorial richiede un livello moderato di conoscenza del AWS dominio Lambda. Dimostra come creare risorse utilizzando AWS Command Line Interface (AWS CLI) e come creare un pacchetto di distribuzione dell'archivio di file .zip per la funzione e le sue dipendenze.

## Tutorial: uso di un trigger Amazon S3 per creare immagini in miniatura

In questo tutorial, creerai e configurerai una funzione Lambda che ridimensiona le immagini aggiunte a un bucket Amazon Simple Storage Service (Amazon S3). Quando aggiungi un file di immagine al bucket, Amazon S3 richiama la tua funzione Lambda. La funzione crea quindi una versione in miniatura dell'immagine e la invia a un altro bucket Amazon S3.



Per completare questo tutorial, completa le seguenti attività:

1. Crea i bucket Amazon S3 di origine e destinazione e carica un'immagine di esempio.
2. Crea una funzione Lambda che ridimensiona un'immagine e restituisce una miniatura in un bucket Amazon S3.
3. Configura un trigger Lambda che richiama la tua funzione quando gli oggetti vengono caricati nel bucket di origine.
4. Testa la tua funzione, prima con un evento fittizio e poi caricando un'immagine nel tuo bucket di origine.

Completando questi passaggi, imparerai come utilizzare Lambda per eseguire un'attività di elaborazione di file su oggetti aggiunti a un bucket Amazon S3. Puoi completare questo tutorial utilizzando AWS Command Line Interface (AWS CLI) o. Console di gestione AWS

Se stai cercando un esempio più semplice per imparare a configurare un trigger Amazon S3 per Lambda, puoi provare [Tutorial: utilizzo di un trigger Amazon S3 per richiamare una funzione Lambda](#).

## Argomenti

- [Prerequisiti](#)
- [Creazione di due bucket Amazon S3](#)
- [Caricamento di un'immagine di test nel bucket di origine](#)
- [Creazione di una policy di autorizzazione](#)
- [Creazione di un ruolo di esecuzione](#)
- [Creazione del pacchetto di implementazione della funzione](#)
- [Creazione della funzione Lambda](#)
- [Configurazione di Amazon S3 per richiamare la funzione](#)
- [Test di una funzione Lambda con un evento fittizio](#)
- [Test della funzione tramite il trigger Amazon S3](#)
- [Pulizia delle risorse](#)

## Prerequisiti

Se desideri utilizzare il AWS CLI per completare il tutorial, installa la versione [più recente di AWS Command Line Interface](#).

Per il codice della funzione Lambda, puoi usare Python o Node.js. Installa gli strumenti di supporto linguistico e un gestore di pacchetti per il linguaggio che desideri utilizzare.

## Installa il AWS Command Line Interface

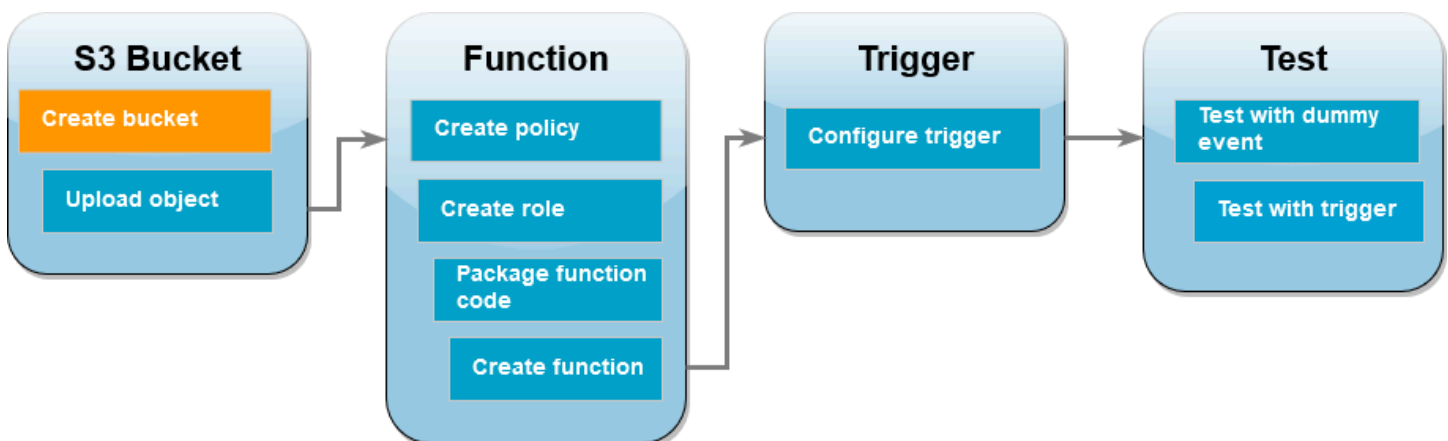
Se non hai ancora installato il AWS Command Line Interface, segui i passaggi in [Installazione o aggiornamento della versione più recente AWS CLI](#) per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

### Note

Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere una Windows-integrated versione di Ubuntu e Bash, [installa il sottosistema Windows per Linux](#).

## Creazione di due bucket Amazon S3

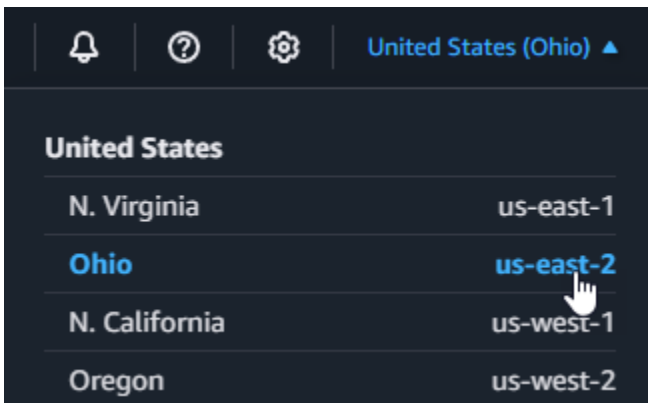


Per prima cosa, crea due bucket Amazon S3. Il primo bucket è il bucket sorgente in cui carichi le tue immagini. Il secondo bucket viene utilizzato da Lambda per salvare la miniatura ridimensionata quando richiami la tua funzione.

## Console di gestione AWS

### Creazione di bucket Amazon S3 (console)

1. Apri la console [Amazon S3](#) e seleziona la pagina General purpose buckets.
2. Seleziona quella più Regione AWS vicina alla tua posizione geografica. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo. Più avanti nel tutorial, è necessario creare la funzione Lambda nella stessa regione.



3. Scegliere Create bucket (Crea bucket).
4. In General configuration (Configurazione generale), eseguire le operazioni seguenti:
  - a. Per il tipo di secchio, assicurati che sia selezionato Uso generale.
  - b. Per Nome del bucket, inserisci un nome univoco globale che soddisfi le [regole di denominazione dei bucket](#) di Amazon S3. I nomi dei bucket possono contenere solo lettere minuscole, numeri, punti (.) e trattini (-).
5. Lascia tutte le altre opzioni impostate sui valori predefiniti e scegli Crea bucket.
6. Ripeti i passaggi da 1 a 5 per creare il bucket di destinazione. Per Nome del bucket, inserisci **amzn-s3-demo-source-bucket-resized**, dove **amzn-s3-demo-source-bucket** è il nome del bucket di origine che hai appena creato.

## AWS CLI

### Per creare i bucket Amazon S3 (AWS CLI)

1. Esegui il comando della CLI sotto riportato per creare il bucket di origine. Il nome che scegli per il bucket deve essere univoco a livello globale e seguire le [regole di denominazione dei bucket](#) di Amazon S3. I nomi possono contenere solo lettere minuscole, numeri, punti (.) e

trattini (-). Per region e LocationConstraint, scegli la [Regione AWS](#) più vicina alla tua posizione geografica.

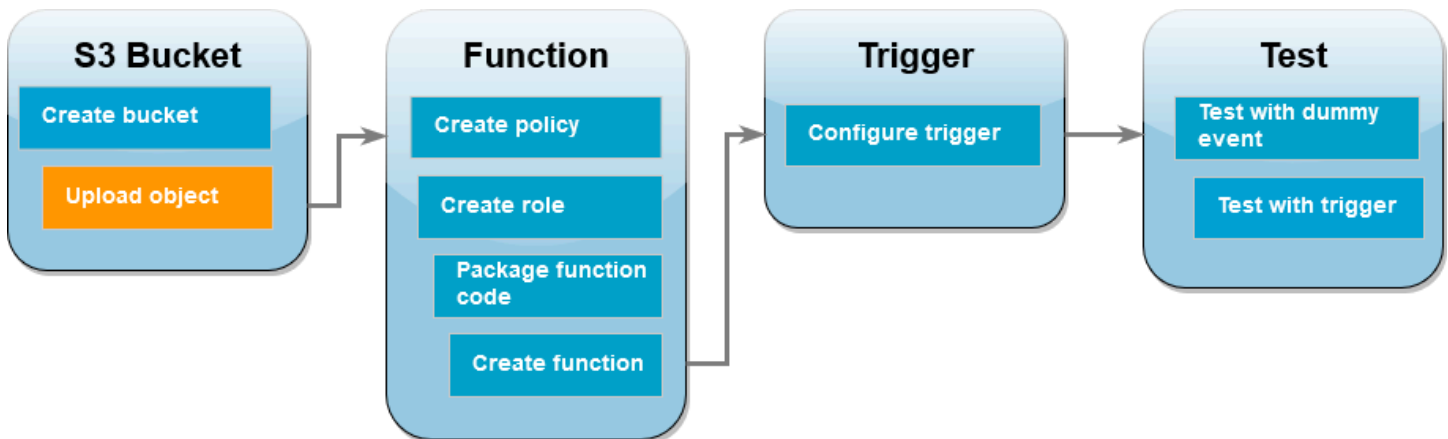
```
aws s3api create-bucket --bucket amzn-s3-demo-source-bucket --region us-east-1 \
--create-bucket-configuration LocationConstraint=us-east-1
```

Più avanti nel tutorial, devi creare la tua funzione Lambda nello stesso bucket sorgente, quindi prendi nota della regione che hai scelto.

2. Esegui il comando sotto riportato per creare il bucket di destinazione. Per il nome del bucket, devi usare **amzn-s3-demo-source-bucket-resized**, dove **amzn-s3-demo-source-bucket** è il nome del bucket di origine che hai creato nel passaggio 1. Per region e LocationConstraint, scegli lo stesso Regione AWS che hai usato per creare il tuo bucket sorgente.

```
aws s3api create-bucket --bucket amzn-s3-demo-source-bucket-resized --region us-east-1 \
--create-bucket-configuration LocationConstraint=us-east-1
```

Caricamento di un'immagine di test nel bucket di origine



Più avanti nel tutorial, testerai la tua funzione Lambda invocandola usando AWS CLI o la console Lambda. Per confermare che la funzione funzioni correttamente, il bucket di origine deve contenere un'immagine di test. Questa immagine può essere qualsiasi file JPG o PNG che scegli.

## Console di gestione AWS

Caricamento di un'immagine di test nel bucket di origine (console)

1. Nella console Amazon S3, apri la pagina [Bucket](#).
2. Seleziona il bucket di origine che hai creato nella fase precedente.
3. Scegli Carica.
4. Scegli Aggiungi file e usa il selettore di file per scegliere l'oggetto da caricare.
5. Seleziona Apri, quindi Carica.

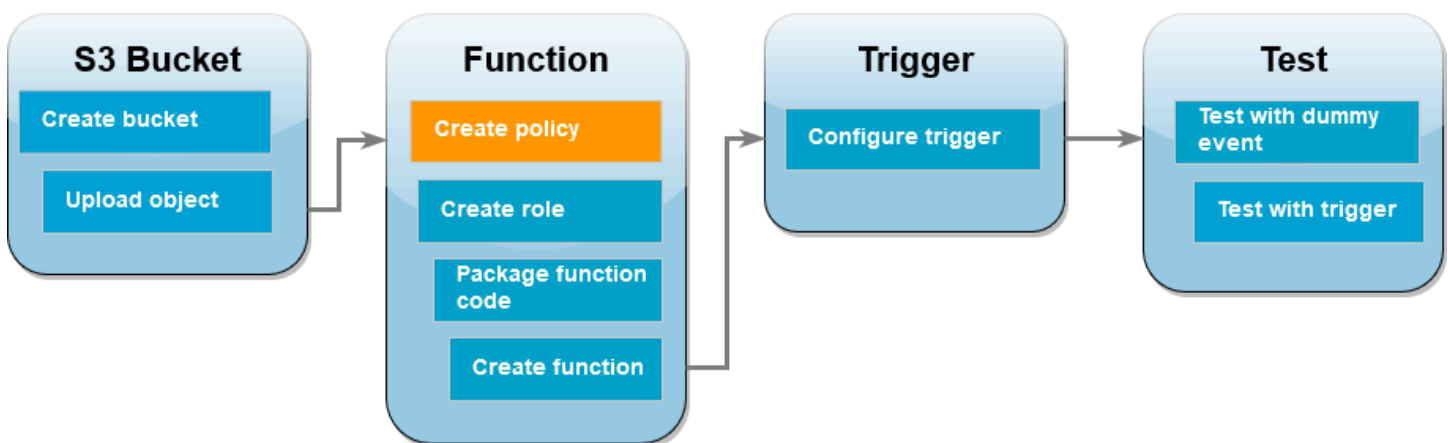
## AWS CLI

Per caricare un'immagine di prova nel bucket sorgente (AWS CLI)

- Dalla directory contenente l'immagine che desideri caricare, esegui il comando della CLI sotto riportato. Sostituisci il parametro `--bucket` con il nome del bucket di origine. Per i parametri `--key` e `--body`, usa il nome del file dell'immagine di test.

```
aws s3api put-object --bucket amzn-s3-demo-source-bucket --key HappyFace.jpg --  
body ./HappyFace.jpg
```

## Creazione di una policy di autorizzazione



Il primo passo per creare una funzione Lambda consiste nel creare una policy di autorizzazione. Questa politica fornisce alla tua funzione le autorizzazioni necessarie per accedere ad altre AWS risorse. Per questo tutorial, la policy fornisce a Lambda le autorizzazioni di lettura e scrittura per i bucket Amazon S3 e le consente di scrivere su Amazon Logs. CloudWatch



## Console di gestione AWS

### Creazione della policy (console)

1. Apri la [pagina](#) Policies della console AWS Identity and Access Management (IAM).
2. Scegli Crea policy.
3. Scegliere la scheda JSON e quindi incollare la seguente policy personalizzata nell'editor JSON.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:CreateLogStream"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::*/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::*/*"
    }
  ]
}
```

4. Scegli Avanti.
5. In Dettagli sulla policy, per Nome policy, inserisci **LambdaS3Policy**.
6. Scegli Crea policy.

## AWS CLI

Per creare la policy (AWS CLI)

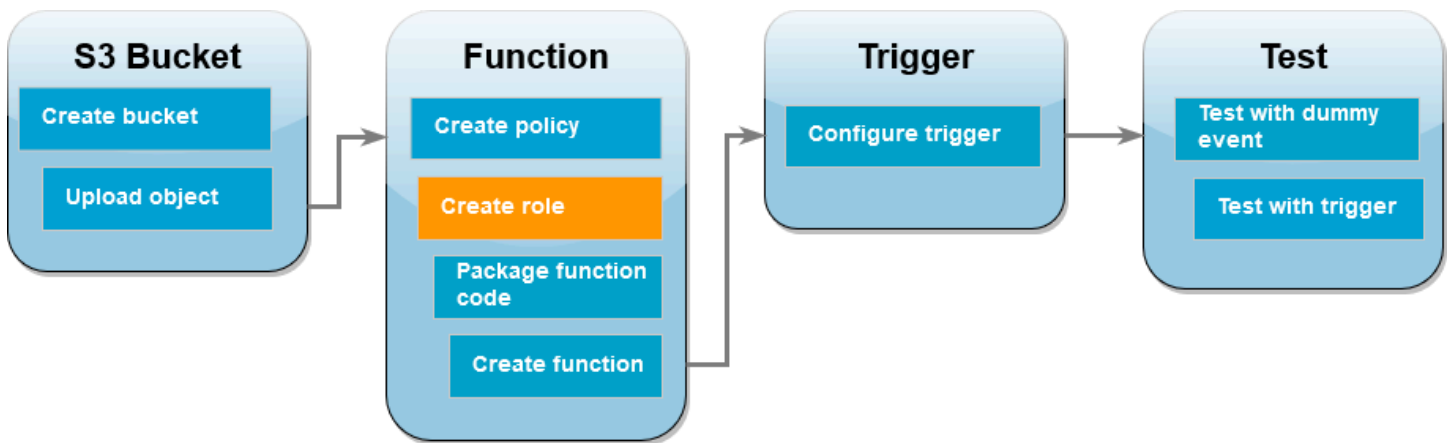
1. Salva il seguente JSON in un file denominato `policy.json`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:CreateLogStream"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::*/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::*/*"
    }
  ]
}
```

2. Nella directory in cui hai salvato il documento della policy JSON, esegui il comando della CLI sotto riportato.

```
aws iam create-policy --policy-name LambdaS3Policy --policy-document file://
policy.json
```

## Creazione di un ruolo di esecuzione



Un ruolo di esecuzione è un ruolo IAM che concede a una funzione Lambda l'autorizzazione all'accesso Servizi AWS e alle risorse. Per concedere alla funzione l'accesso in lettura e scrittura a un bucket Amazon S3, è necessario collegare la policy di autorizzazione che hai creato nel passaggio precedente.

## Console di gestione AWS

Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione (console)

1. Apri la pagina [Ruoli](#) della console (IAM).
2. Scegliere Crea ruolo.
3. Per Tipo di entità attendibile, seleziona Servizio AWS, mentre per Caso d'uso, seleziona Lambda.
4. Scegli Avanti.
5. Aggiungi la policy di autorizzazione che hai creato nel passaggio precedente effettuando le seguenti operazioni:
  - a. Nella casella di ricerca delle policy, immettere **LambdaS3Policy**.
  - b. Nei risultati di ricerca, seleziona la casella di controllo per LambdaS3Policy.
  - c. Scegli Avanti.
6. In Dettagli ruolo, per Nome ruolo, inserisci **LambdaS3Role**.
7. Scegliere Crea ruolo.

## AWS CLI

Per creare un ruolo di esecuzione e allegare la tua politica di autorizzazioni (AWS CLI)

1. Salva il seguente JSON in un file denominato `trust-policy.json`. Questa politica di fiducia consente a Lambda di utilizzare le autorizzazioni del ruolo concedendo al responsabile del servizio il `lambda.amazonaws.com` permesso di chiamare l'azione Servizio di token di sicurezza AWS (AWS STS. `AssumeRole`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

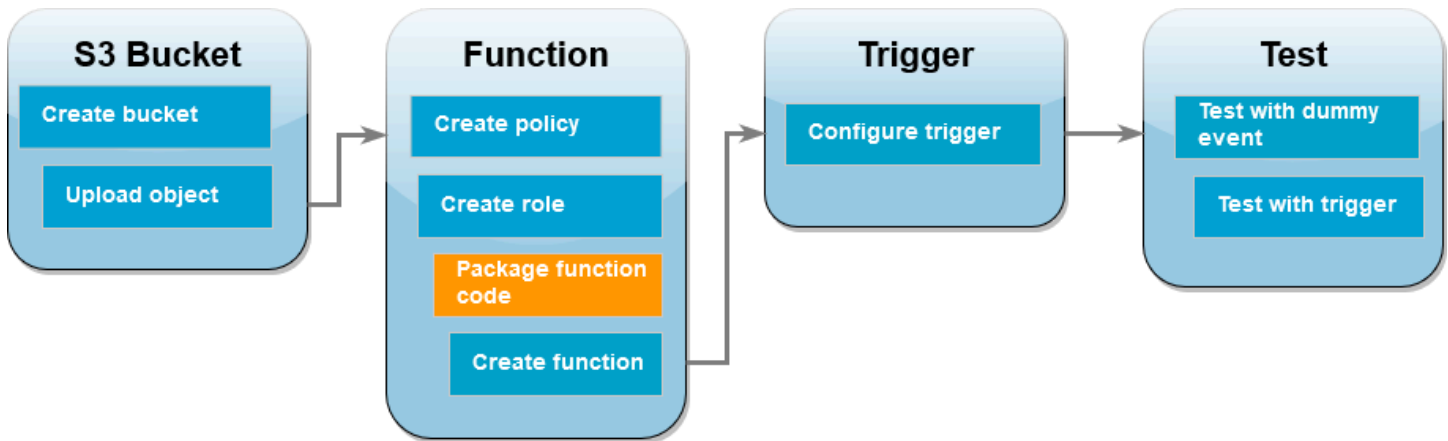
2. Nella directory in cui hai salvato il documento della policy di attendibilità JSON, esegui il comando della CLI sotto riportato per creare il ruolo di esecuzione.

```
aws iam create-role --role-name LambdaS3Role --assume-role-policy-document
file://trust-policy.json
```

3. Per collegare la policy di autorizzazione creata nel passaggio precedente, esegui il comando della CLI sotto riportato. Sostituisci il Account AWS numero nell'ARN della policy con il tuo numero di account.

```
aws iam attach-role-policy --role-name LambdaS3Role --policy-arn
arn:aws:iam::123456789012:policy/LambdaS3Policy
```

## Creazione del pacchetto di implementazione della funzione



Per creare la funzione, occorre creare un pacchetto di implementazione contenente la funzione e le rispettive dipendenze. Per questa funzione `CreateThumbnail`, il codice della funzione utilizza una libreria separata per il ridimensionamento dell'immagine. Segui le istruzioni per il linguaggio scelto per creare un pacchetto di implementazione contenente la libreria richiesta.

### Node.js

Per creare il pacchetto di distribuzione () Node.js

1. Crea una directory denominata `lambda-s3` per il codice della funzione e le dipendenze e naviga al suo interno.

```
mkdir lambda-s3
cd lambda-s3
```

2. Crea un nuovo Node.js progetto con npm. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi Enter.

```
npm init
```

3. Salvare il codice della funzione seguente in un file denominato `index.mjs`. Assicurati di sostituire `us-east-1` con la Regione AWS in cui hai creato i tuoi bucket di origine e destinazione.

```
// dependencies
import { S3Client, GetObjectCommand, PutObjectCommand } from '@aws-sdk/client-s3';
```

```
import { Readable } from 'stream';

import sharp from 'sharp';
import util from 'util';

// create S3 client
const s3 = new S3Client({region: 'us-east-1'});

// define the handler function
export const handler = async (event, context) => {

// Read options from the event parameter and get the source bucket
console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
  const srcBucket = event.Records[0].s3.bucket.name;

// Object key may have spaces or unicode non-ASCII characters
const srcKey    = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));
const dstBucket = srcBucket + "-resized";
const dstKey    = "resized-" + srcKey;

// Infer the image type from the file suffix
const typeMatch = srcKey.match(/\.[^\.]*$/);
if (!typeMatch) {
  console.log("Could not determine the image type.");
  return;
}

// Check that the image type is supported
const imageType = typeMatch[1].toLowerCase();
if (imageType !== "jpg" && imageType !== "png") {
  console.log(`Unsupported image type: ${imageType}`);
  return;
}

// Get the image from the source bucket. GetObjectCommand returns a stream.
try {
  const params = {
    Bucket: srcBucket,
    Key: srcKey
  };
  var response = await s3.send(new GetObjectCommand(params));
  var stream = response.Body;
```

```
// Convert stream to buffer to pass to sharp resize function.
if (stream instanceof Readable) {
    var content_buffer = Buffer.concat(await stream.toArray());

    } else {
        throw new Error('Unknown object stream type');
    }

} catch (error) {
    console.log(error);
    return;
}

// set thumbnail width. Resize will set the height automatically to maintain
// aspect ratio.
const width = 200;

// Use the sharp module to resize the image and save in a buffer.
try {
    var output_buffer = await sharp(content_buffer).resize(width).toBuffer();

} catch (error) {
    console.log(error);
    return;
}

// Upload the thumbnail image to the destination bucket
try {
    const destparams = {
        Bucket: dstBucket,
        Key: dstKey,
        Body: output_buffer,
        ContentType: "image"
    };

    const putResult = await s3.send(new PutObjectCommand(destparams));

} catch (error) {
    console.log(error);
    return;
}
```

```
console.log('Successfully resized ' + srcBucket + '/' + srcKey +  
  ' and uploaded to ' + dstBucket + '/' + dstKey);  
};
```

4. Nella directory `lambda-s3`, installa la libreria `sharp` utilizzando `npm`. Tieni presente che l'ultima versione di `sharp` (0.33) non è compatibile con Lambda. Per completare questo tutorial, installa la versione 0.32.6.

```
npm install sharp@0.32.6
```

Il comando `install` di `npm` crea una directory `node_modules` per i tuoi moduli. Dopo questo passaggio, la struttura di directory dovrebbe avere un aspetto simile al seguente.

```
lambda-s3  
|- index.mjs  
|- node_modules  
|  |- base64js  
|  |- bl  
|  |- buffer  
|  ...  
|- package-lock.json  
|- package.json
```

5. Crea un pacchetto di implementazione `.zip` contenente il codice della funzione e le rispettive dipendenze. Su MacOS o Linux, esegui il comando sotto riportato.

```
zip -r function.zip .
```

Su Windows, utilizza il tuo strumento di compressione preferito per creare il file `.zip`. Assicurati che i tuoi file `index.mjs`, `package.json` e `package-lock.json` e la tua directory `node_modules` si trovino tutti nella directory principale del tuo file `.zip`.

## Python

### Creazione del pacchetto di implementazione (Python)

1. Salva il codice di esempio come un file denominato `lambda_function.py`.

```
import boto3
```



```

import os
import sys
import uuid
from urllib.parse import unquote_plus
from PIL import Image
import PIL.Image

s3_client = boto3.client('s3')

def resize_image(image_path, resized_path):
    with Image.open(image_path) as image:
        image.thumbnail(tuple(x / 2 for x in image.size))
        image.save(resized_path)

def lambda_handler(event, context):
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key'])
        tmpkey = key.replace('/', '')
        download_path = '/tmp/{}'.format(uuid.uuid4(), tmpkey)
        upload_path = '/tmp/resized-{}'.format(tmpkey)
        s3_client.download_file(bucket, key, download_path)
        resize_image(download_path, upload_path)
        s3_client.upload_file(upload_path, '{}-resized'.format(bucket), 'resized-{}'.format(key))

```

2. Nella stessa directory in cui hai creato il file `lambda_function.py`, crea una nuova directory denominata `package` e installa la libreria [Pillow \(PIL\)](#) e AWS SDK per Python (Boto3). Sebbene il runtime Lambda di Python includa una versione dell'SDK Boto3, ti consigliamo di aggiungere tutte le dipendenze della funzione al pacchetto di implementazione, anche se sono incluse nel runtime. Per ulteriori informazioni, consulta [Dipendenze del runtime in Python](#).

```

mkdir package
pip install \
--platform manylinux2014_x86_64 \
--target=package \
--implementation cp \
--python-version 3.12 \
--only-binary=:all: --upgrade \
pillow boto3

```

La libreria Pillow contiene codice C/C ++. Utilizzando le `--only-binary=:all:` opzioni `--platform manylinux_2014_x86_64` and, pip scarica e installa una versione di Pillow che contiene file binari precompilati compatibili con il sistema operativo Amazon Linux 2. Ciò garantisce che il pacchetto di distribuzione funzioni nell'ambiente di esecuzione Lambda, indipendentemente dal sistema operativo e dall'architettura della macchina di compilazione locale.

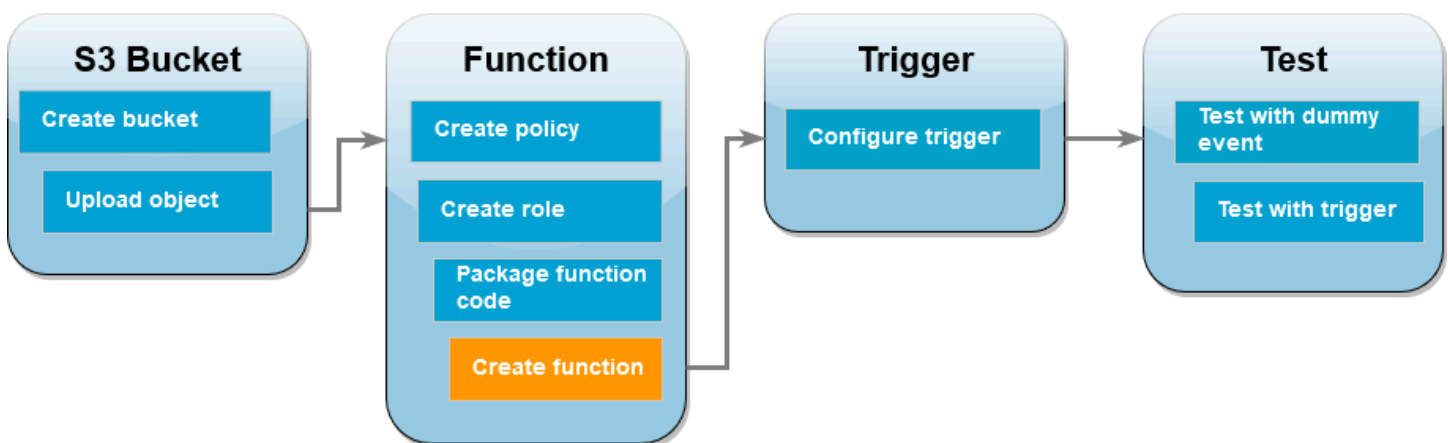
3. Crea un file `.zip` contenente il codice dell'applicazione e le librerie Pillow e Boto3. Su Linux o MacOS, esegui i comandi riportati di seguito dall'interfaccia della linea di comando.

```
cd package
zip -r ../lambda_function.zip .
cd ..
zip lambda_function.zip lambda_function.py
```

Su Windows, usa il tuo strumento di compressione preferito per creare il file `lambda_function.zip`. Assicurati che il tuo file `lambda_function.py` e le cartelle contenenti le tue dipendenze si trovino tutti nella directory principale del file `.zip`.

Puoi creare il tuo pacchetto di implementazione anche utilizzando un ambiente virtuale Python. Per informazioni, consultare [Utilizzo di archivi di file .zip per le funzioni Lambda in Python](#)

### Creazione della funzione Lambda



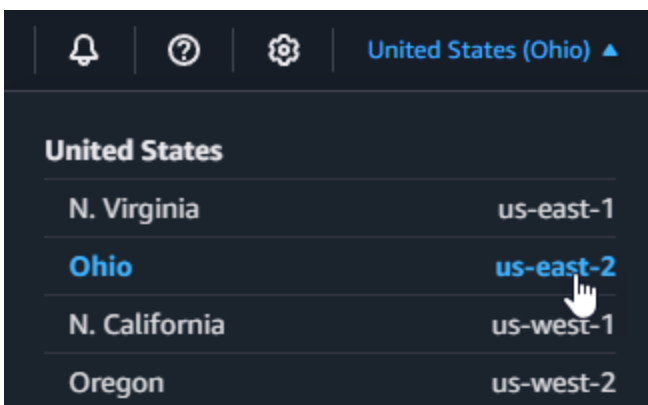
Puoi creare la tua funzione Lambda utilizzando la console AWS CLI o la console Lambda. Segui le istruzioni per il linguaggio scelto per creare la funzione.

## Console di gestione AWS

### Creazione della funzione (console)

Per creare la tua funzione Lambda utilizzando la console, devi prima creare una funzione di base contenente del codice «Hello world». Quindi, sostituisci questo codice con il codice della tua funzione caricando il file .zip o JAR creato nel passaggio precedente.

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Assicurati di lavorare nello stesso ambiente in cui Regione AWS hai creato il bucket Amazon S3. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo.



3. Selezionare Create function (Crea funzione).
4. Scegli Author from scratch (Crea da zero).
5. In Basic information (Informazioni di base) eseguire queste operazioni:
  - a. Nel campo Function name (Nome funzione), immettere **CreateThumbnail**.
  - b. Per Runtime, scegli Node.js 22.x o Python 3.12 in base alla lingua scelta per la tua funzione.
  - c. In Architecture (Architettura), scegli x86\_64.
6. Nella scheda Modifica ruolo di esecuzione predefinito, effettua le seguenti operazioni:
  - a. Espandi la scheda, quindi scegli Utilizza un ruolo esistente.
  - b. Seleziona il LambdaS3Role che hai creato in precedenza.
7. Scegli Crea funzione.

## Caricamento del codice della funzione (console)

1. Nel riquadro Origine del codice, scegli Carica da.
2. Scegli File .zip.
3. Scegli Carica.
4. Nel selettore di file, seleziona il tuo file .zip e scegli Apri.
5. Selezionare Salva.

## AWS CLI

Per creare la funzione (AWS CLI)

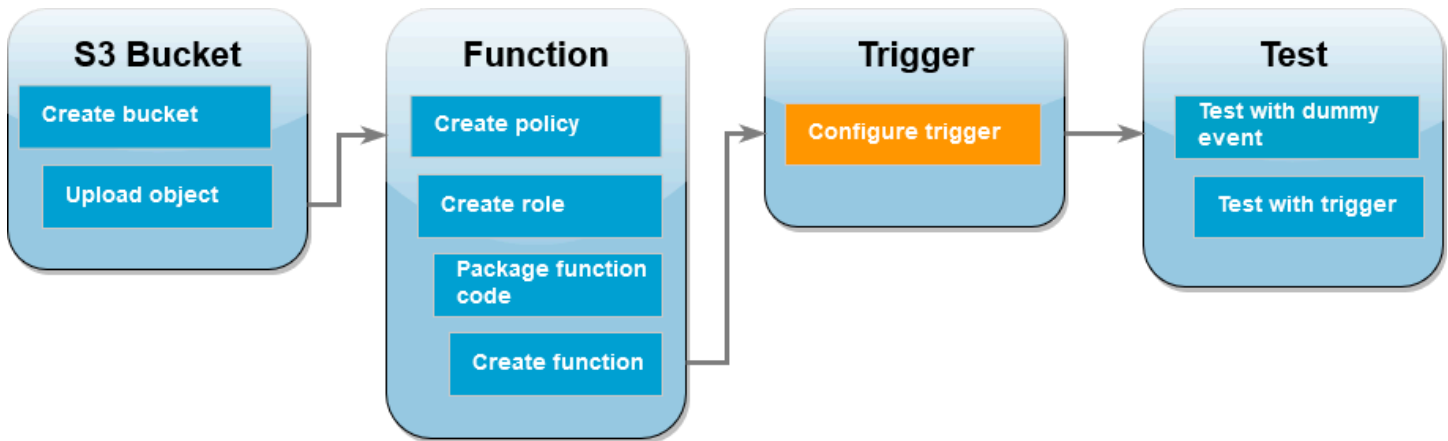
- Esegui il comando della CLI per il linguaggio che hai scelto. Per il `role` parametro, assicurati di sostituirlo `123456789012` con il tuo Account AWS ID. Per il parametro `region`, sostituisci `us-east-1` con la regione in cui hai creato i bucket Amazon S3.
- Per Node.js, esegui il seguente comando dalla directory contenente il tuo `function.zip` file.

```
aws lambda create-function --function-name CreateThumbnail \  
--zip-file fileb://function.zip --handler index.handler --runtime nodejs24.x \  
--timeout 10 --memory-size 1024 \  
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-east-1
```

- Per Python, esegui il comando sotto riportato dalla directory contenente il file `lambda_function.zip`.

```
aws lambda create-function --function-name CreateThumbnail \  
--zip-file fileb://lambda_function.zip --handler \  
lambda_function.lambda_handler \  
--runtime python3.14 --timeout 10 --memory-size 1024 \  
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-east-1
```

## Configurazione di Amazon S3 per richiamare la funzione



Affinché la funzione Lambda venga eseguita quando carichi un'immagine nel bucket di origine, devi configurare un trigger per la funzione. È possibile configurare il trigger Amazon S3 utilizzando la console Lambda o la AWS CLI.

### ⚠ Important

Questa procedura configura il bucket Amazon S3 per richiamare la funzione ogni volta che un oggetto viene creato nel bucket. Assicurati di configurare questa opzione solo sul bucket di origine. Se la tua funzione Lambda crea oggetti nello stesso bucket che la richiama, la tua funzione può essere [richiamata continuamente in un ciclo ricorsivo \(loop\)](#). Ciò può comportare la fatturazione di addebiti imprevisti al tuo Account AWS.

## Console di gestione AWS

### Configurazione del trigger Amazon S3 (console)

1. Apri la pagina [Funzioni](#) della console Lambda e scegli la tua funzione (CreateThumbnail).
2. Selezionare Add trigger (Aggiungi trigger).
3. Seleziona S3.
4. In Bucket, seleziona il tuo bucket di origine.
5. In Tipi di eventi, seleziona Tutti gli eventi di creazione di oggetti.
6. In Invocazione ricorsiva, seleziona la casella di controllo per confermare che non è consigliabile utilizzare lo stesso bucket Amazon S3 per input e output. Per maggiori informazioni sui modelli di invocazione ricorsivi in Lambda, consulta [Schemi ricorsivi che causano loop indeterminati delle funzioni Lambda](#) in Serverless Land.

## 7. Scegliere Aggiungi.

Quando crei un trigger utilizzando la console Lambda, Lambda crea automaticamente una [policy basata sulle risorse](#) per concedere al servizio selezionato l'autorizzazione a richiamare la funzione.

## AWS CLI

Per configurare il trigger di Amazon S3 (AWS CLI)

1. Affinché il bucket di origine Amazon S3 richiami la funzione quando aggiungi un file di immagine, devi prima configurare le autorizzazioni per la funzione utilizzando una [policy basata sulle risorse](#). Una dichiarazione politica basata sulle risorse fornisce altre Servizi AWS autorizzazioni per richiamare la tua funzione. Per autorizzare Amazon S3 a richiamare la tua funzione, esegui il comando della CLI comando sotto riportato. Assicurati di sostituire il `source-account` parametro con il tuo Account AWS ID e di utilizzare il tuo nome di bucket di origine.

```
aws lambda add-permission --function-name CreateThumbnail \  
--principal s3.amazonaws.com --statement-id s3invoke --action  
"lambda:InvokeFunction" \  
--source-arn arn:aws:s3:::amzn-s3-demo-source-bucket \  
--source-account 123456789012
```

La policy che definisci con questo comando consente ad Amazon S3 di richiamare la tua funzione solo quando viene eseguita un'operazione sul tuo bucket di origine.

### Note

Sebbene i nomi dei bucket Amazon S3 siano univoci a livello globale, quando utilizzi policy basate sulle risorse è consigliabile specificare che il bucket deve appartenere al tuo account. Questo perché se elimini un bucket, è possibile che un altro ne Account AWS crei uno con lo stesso Amazon Resource Name (ARN).

2. Salva il seguente JSON in un file denominato `notification.json`. Quando viene applicato al tuo bucket di origine, questo JSON configura il bucket in modo che invii una notifica alla funzione Lambda ogni volta che viene aggiunto un nuovo oggetto. Sostituisci

il Account AWS numero e Regione AWS nella funzione Lambda ARN con il tuo numero di account e la tua regione.

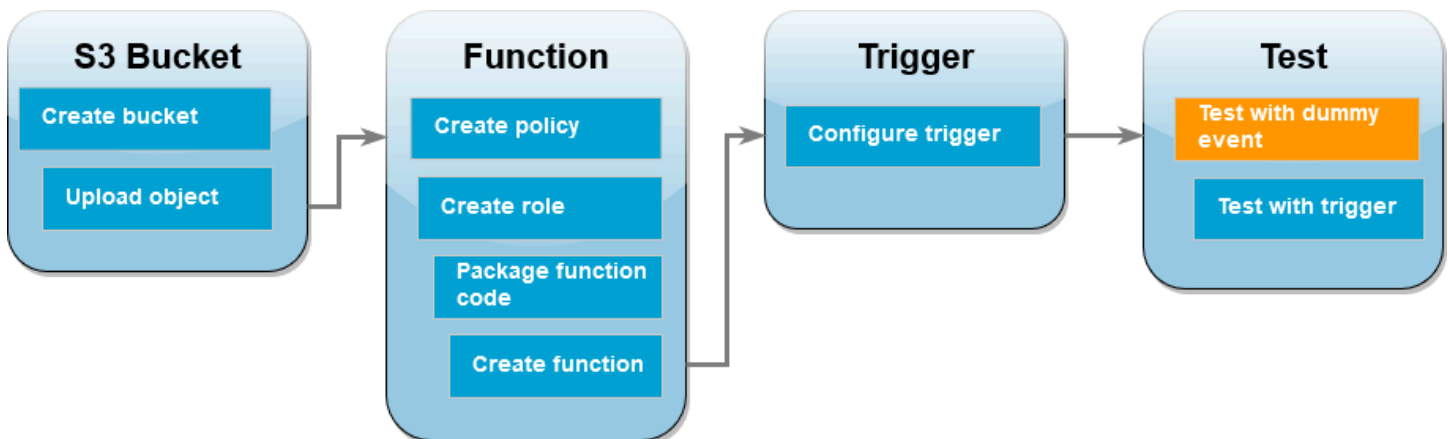
```
{
  "LambdaFunctionConfigurations": [
    {
      "Id": "CreateThumbnailEventConfiguration",
      "LambdaFunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:CreateThumbnail",
      "Events": [ "s3:ObjectCreated:Put" ]
    }
  ]
}
```

3. Esegui il comando della CLI comando sotto riportato per applicare le impostazioni di notifica nel file JSON che hai creato al tuo bucket di origine. Sostituisci `amzn-s3-demo-source-bucket` con il nome del tuo bucket di origine.

```
aws s3api put-bucket-notification-configuration --bucket amzn-s3-demo-source-
bucket \
--notification-configuration file://notification.json
```

Per ulteriori informazioni sul comando `put-bucket-notification-configuration` e sull'opzione `notification-configuration`, consulta [put-bucket-notification-configuration](#) in Riferimento ai comandi di AWS CLI.

Test di una funzione Lambda con un evento fittizio



Prima di testare l'intera configurazione aggiungendo un file di immagine al tuo bucket di origine Amazon S3, verifica che la tua funzione Lambda funzioni correttamente richiamandola con un evento fittizio. Un evento in Lambda è un JSON-formatted document che contiene dati che la funzione deve elaborare. Quando la funzione viene richiamata da Amazon S3, l'evento inviato alla funzione contiene informazioni come il nome del bucket, l'ARN del bucket e la chiave dell'oggetto.

## Console di gestione AWS

### Test di una funzione Lambda con un evento fittizio (console)

1. Apri la pagina [Funzioni](#) della console Lambda e scegli la tua funzione (CreateThumbnail).
2. Seleziona la scheda Test.
3. Per creare il tuo evento di test, nel riquadro Evento di test, procedi come segue:
  - a. In Operazione evento di test, seleziona Crea nuovo evento.
  - b. Per Event name (Nome evento) immettere **myTestEvent**.
  - c. Per Modello, seleziona S3 Put.
  - d. Sostituisci i valori dei seguenti parametri con i tuoi valori.
    - PerawsRegion, sostituiscilo us-east-1 con Regione AWS quello in cui hai creato i tuoi bucket Amazon S3.
    - Per name, sostituisci amzn-s3-demo-bucket con il nome del bucket di origine Amazon S3.
    - Per key, sostituisci test%2Fkey con il nome del file dell'oggetto di test che hai caricato nel bucket di origine durante il passaggio [Caricamento di un'immagine di test nel bucket di origine](#).

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      }
    }
  ],
}
```



```

    "requestParameters": {
      "sourceIPAddress": "127.0.0.1"
    },
    "responseElements": {
      "x-amz-request-id": "EXAMPLE123456789",
      "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
    },
    "s3": {
      "s3SchemaVersion": "1.0",
      "configurationId": "testConfigRule",
      "bucket": {
        "name": "amzn-s3-demo-bucket",
        "ownerIdentity": {
          "principalId": "EXAMPLE"
        },
        "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
      },
      "object": {
        "key": "test%2Fkey",
        "size": 1024,
        "eTag": "0123456789abcdef0123456789abcdef",
        "sequencer": "0A1B2C3D4E5F678901"
      }
    }
  }
]
}

```

- e. Selezionare Salva.
4. Nel riquadro Evento di test, scegli Test.
5. Per verificare che la funzione abbia creato una versione ridimensionata dell'immagine e l'abbia archiviata nel bucket Amazon S3 di destinazione, procedi come segue:
  - a. Nella console Amazon S3, apri la pagina [Bucket](#).
  - b. Scegli il bucket di destinazione e conferma che il file ridimensionato sia elencato nel riquadro Oggetti.

## AWS CLI

Per testare la tua funzione Lambda con un evento fittizio (AWS CLI)

1. Salva il seguente JSON in un file denominato `dummyS3Event.json`. Sostituisci i valori dei seguenti parametri con i tuoi valori:
  - Per `awsRegion`, sostituiscilo `us-east-1` con Regione AWS quello in cui hai creato i tuoi bucket Amazon S3.
  - Per `name`, sostituisci `amzn-s3-demo-bucket` con il nome del bucket di origine Amazon S3.
  - Per `key`, sostituisci `test%2Fkey` con il nome del file dell'oggetto di test che hai caricato nel bucket di origine durante il passaggio [Caricamento di un'immagine di test nel bucket di origine](#).

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          }
        }
      }
    }
  ]
}
```

```

    },
    "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
  },
  "object": {
    "key": "test%2Fkey",
    "size": 1024,
    "eTag": "0123456789abcdef0123456789abcdef",
    "sequencer": "0A1B2C3D4E5F678901"
  }
}
]
}

```

2. Nella directory in cui hai salvato il file `dummyS3Event.json`, richiama la funzione eseguendo il seguente comando della CLI. Questo comando richiama la funzione Lambda in modo sincrono specificando `RequestResponse` come valore del parametro `invocation-type`. Per ulteriori informazioni sulla chiamata sincrona e asincrona, consulta la pagina [Invocazione delle funzioni Lambda](#).

```

aws lambda invoke --function-name CreateThumbnail \
--invocation-type RequestResponse --cli-binary-format raw-in-base64-out \
--payload file://dummyS3Event.json outputfile.txt

```

L'opzione `cli-binary-format` è obbligatoria se si utilizza la versione 2 della AWS CLI. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta [Opzioni della riga di comando globali supportate da AWS CLI](#).

3. Verifica che la funzione abbia creato una versione in miniatura dell'immagine e l'abbia salvata nel bucket Amazon S3 di destinazione. Esegui il comando della CLI sotto riportato sostituendo `amzn-s3-demo-source-bucket-resized` con il nome del tuo bucket di destinazione.

```

aws s3api list-objects-v2 --bucket amzn-s3-demo-source-bucket-resized

```

Verrà visualizzato un output simile al seguente. Il parametro `Key` mostra il nome del file di immagine ridimensionato.

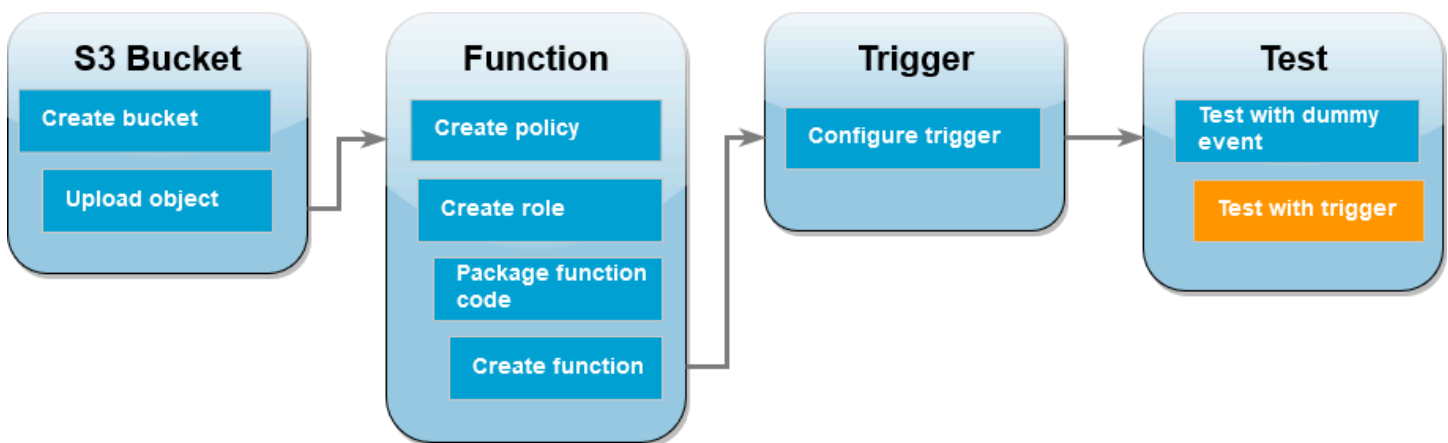
```

{

```

```
"Contents": [  
  {  
    "Key": "resized-HappyFace.jpg",  
    "LastModified": "2023-06-06T21:40:07+00:00",  
    "ETag": "\"d8ca652ffe83ba6b721ffc20d9d7174a\"",  
    "Size": 2633,  
    "StorageClass": "STANDARD"  
  }  
]  
}
```

### Test della funzione tramite il trigger Amazon S3



Ora che hai verificato che la funzione Lambda funziona correttamente, sei pronto per testare la configurazione completa aggiungendo un file di immagine al tuo bucket sorgente Amazon S3. Quando aggiungi l'immagine al bucket di origine, la tua funzione Lambda dovrebbe essere richiamata automaticamente. La tua funzione crea una versione ridimensionata del file e la archivia nel bucket di destinazione.

### Console di gestione AWS

#### Test della funzione Lambda tramite il trigger Amazon S3 (console)

1. Per caricare un'immagine nel bucket Amazon S3, procedi come segue:
  - a. Apri la pagina [Bucket](#) della console Amazon S3 e scegli il bucket di origine.
  - b. Scegli Carica.
  - c. Scegli Aggiungi file e utilizza il selettore di file per scegliere il file di immagine da caricare. L'oggetto dell'immagine può essere qualsiasi file .jpg o .png.

- d. Seleziona Apri, quindi Carica.
2. Verifica che Lambda abbia salvato una versione ridimensionata del tuo file di immagine nel bucket di destinazione effettuando le seguenti operazioni:
  - a. Torna alla pagina [Bucket](#) della console Amazon S3 e scegli il bucket di destinazione.
  - b. Nel riquadro Oggetti, ora dovresti vedere due file di immagine ridimensionati, uno per ogni test della tua funzione Lambda. Per scaricare l'immagine ridimensionata, seleziona il file, quindi scegli Scarica.

## AWS CLI

Per testare la tua funzione Lambda utilizzando il trigger Amazon S3 (AWS CLI)

1. Dalla directory contenente l'immagine che desideri caricare, esegui il comando della CLI sotto riportato. Sostituisci il parametro `--bucket` con il nome del bucket di origine. Per i parametri `--key` e `--body`, usa il nome del file dell'immagine di test. L'immagine di test può essere qualsiasi file .jpg o .png.

```
aws s3api put-object --bucket amzn-s3-demo-source-bucket --key SmileyFace.jpg --  
body ./SmileyFace.jpg
```

2. Verifica che la funzione abbia creato una versione in miniatura dell'immagine e l'abbia salvata nel bucket Amazon S3 di destinazione. Esegui il comando della CLI sotto riportato sostituendo `amzn-s3-demo-source-bucket-resized` con il nome del tuo bucket di destinazione.

```
aws s3api list-objects-v2 --bucket amzn-s3-demo-source-bucket-resized
```

Se la funzione viene eseguita correttamente, vedrai un risultato simile al seguente. Il bucket di destinazione dovrebbe ora contenere due file ridimensionati.

```
{  
  "Contents": [  
    {  
      "Key": "resized-HappyFace.jpg",  
      "LastModified": "2023-06-07T00:15:50+00:00",  
      "ETag": "\"7781a43e765a8301713f533d70968a1e\"",  
      "Size": 2763,  
      "StorageClass": "STANDARD"
```

```
    },  
    {  
      "Key": "resized-SmileyFace.jpg",  
      "LastModified": "2023-06-07T00:13:18+00:00",  
      "ETag": "\"ca536e5a1b9e32b22cd549e18792cdbc\"",  
      "Size": 1245,  
      "StorageClass": "STANDARD"  
    }  
  ]  
}
```

## Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili al tuo Account AWS.

Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **confirm** nel campo di immissione testo e scegli Delete (Elimina).

Per eliminare la policy creata

1. Apri la pagina [Policies \(Policy\)](#) nella console IAM.
2. Seleziona la policy creata (AWSLambdaS3Policy).
3. Scegliere Policy actions (Operazioni policy), Delete (Elimina).
4. Scegliere Delete (Elimina).

Per eliminare il ruolo di esecuzione

1. Aprire la pagina [Ruoli](#) della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

## Per eliminare il bucket S3

1. Aprire la [console Amazon S3](#).
2. Selezionare il bucket creato in precedenza.
3. Scegliere Elimina.
4. Inserisci il nome del bucket nel campo di immissione testo.
5. Scegli Delete Bucket (Elimina bucket).

## Uso dei segreti in Secrets Manager nelle funzioni Lambda

AWS Secrets Manager ti aiuta a gestire credenziali, chiavi API e altri segreti di cui hanno bisogno le tue funzioni Lambda. Hai due approcci principali per recuperare i segreti nelle tue funzioni Lambda, entrambi offrono prestazioni migliori e costi inferiori rispetto al recupero dei segreti direttamente tramite l'SDK: AWS

- AWS parametri e segreti Estensione Lambda: una soluzione indipendente dal runtime che fornisce una semplice interfaccia HTTP per il recupero dei segreti
- Utilità Powertools for AWS Lambda parameters: una soluzione integrata nel codice che supporta più provider (Secrets Manager, Parameter Store) con trasformazioni integrate AppConfig

Entrambi gli approcci mantengono cache locali di segreti, eliminando la necessità per la funzione di chiamare Secrets Manager per ogni invocazione. Quando la funzione richiede un segreto, la cache viene controllata per prima. Se il segreto è disponibile e non è scaduto, viene restituito immediatamente. Altrimenti, viene recuperato da Secrets Manager, memorizzato nella cache e restituito. Questo meccanismo di memorizzazione nella cache consente tempi di risposta più rapidi e costi ridotti riducendo al minimo le chiamate API.

### Scelta di un approccio

Considera questi fattori quando scegli tra l'estensione e: PowerTools

Usa i AWS parametri e i segreti dell'estensione Lambda quando:

- Vuoi una soluzione indipendente dal runtime che funzioni con qualsiasi runtime Lambda
- Preferisci non aggiungere dipendenze di codice alla tua funzione
- Devi solo recuperare i segreti da Secrets Manager o Parameter Store

Usa l'utilità Powertools for AWS Lambda parameters quando:

- Vuoi un'esperienza di sviluppo integrata con il codice dell'applicazione
- È necessario il supporto per più provider (Secrets Manager, Parameter Store, AppConfig)
- Desideri trasformazioni di dati integrate (analisi JSON, decodifica base64)
- Stai usando i runtime Python TypeScript, Java o .NET

### Quando usare Secrets Manager con Lambda

Gli scenari più comuni per l'utilizzo di Secrets Manager con Lambda includono:



- Memorizzazione delle credenziali del database utilizzate dalla funzione per connettersi ad Amazon RDS o ad altri database
- Gestione delle chiavi API per i servizi esterni chiamati dalle tue funzioni
- Memorizzazione delle chiavi di crittografia o di altri dati di configurazione sensibili
- Rotazione automatica delle credenziali senza dover aggiornare il codice della funzione

## Utilizzo di AWS parametri e segreti dell'estensione Lambda

I AWS parametri e i segreti dell'estensione Lambda utilizza una semplice interfaccia HTTP compatibile con qualsiasi runtime Lambda. Per impostazione predefinita, memorizza nella cache i segreti per 300 secondi (5 minuti) e può contenere fino a 1.000 segreti. Puoi [personalizzare queste impostazioni con variabili di ambiente](#).

### Usa Secrets Manager in una funzione Lambda

Questa sezione presuppone che tu abbia già un segreto Secrets Manager. Per creare un segreto, consulta [Creare un AWS Secrets Manager segreto](#).

### Creazione del pacchetto di implementazione

Scegli il tuo runtime preferito e segui i passaggi per creare una funzione che recuperi segreti da Secrets Manager. La funzione di esempio recupera un segreto da Secrets Manager e può essere utilizzata per accedere alle credenziali del database, alle chiavi API o ad altri dati di configurazione sensibili nelle applicazioni.

### Python

Per creare una funzione Python

1. Crea e vai in una nuova directory di progetto. Esempio:

```
mkdir my_function
cd my_function
```

2. Crea un file denominato `lambda_function.py` con il codice seguente. Per `secret_name`, utilizza il nome o il nome della risorsa Amazon (ARN) del segreto.

```
import json
import boto3
import requests
```

```

def lambda_handler(event, context):
    try:
        # Replace with the name or ARN of your secret
        secret_name = "arn:aws:secretsmanager:us-
east-1:111122223333:secret:SECRET_NAME"

        # Resolve the session token from the credential provider chain so that
        this
        # works across all initialization modes, including SnapStart. Resolve
        inside
        # the handler so the token stays fresh after a SnapStart restore.
        session_token =
        boto3.Session().get_credentials().get_frozen_credentials().token

        secrets_extension_endpoint = f"http://localhost:2773/secretsmanager/get?
secretId={secret_name}"
        headers = {"X-Aws-Parameters-Secrets-Token": session_token}

        response = requests.get(secrets_extension_endpoint, headers=headers)
        print(f"Response status code: {response.status_code}")

        secret = json.loads(response.text)["SecretString"]
        print(f"Retrieved secret: {secret}")

        return {
            'statusCode': response.status_code,
            'body': json.dumps({
                'message': 'Successfully retrieved secret',
                'secretRetrieved': True
            })
        }

    except Exception as e:
        print(f"Error: {str(e)}")
        return {
            'statusCode': 500,
            'body': json.dumps({
                'message': 'Error retrieving secret',
                'error': str(e)
            })
        }

```

3. Creare un file denominato `requirements.txt` con il seguente contenuto:

```
requests
```

4. Installare le dipendenze:

```
pip install -r requirements.txt -t .
```

5. Crea un file .zip contenente tutti i file:

```
zip -r function.zip .
```

## Node.js

Per creare una Node.js funzione

1. Crea e vai in una nuova directory di progetto. Esempio:

```
mkdir my_function  
cd my_function
```

2. Crea un file denominato `index.mjs` con il codice seguente. Per `secret_name`, utilizza il nome o il nome della risorsa Amazon (ARN) del segreto.

```
import http from 'http';  
import { fromNodeProviderChain } from '@aws-sdk/credential-providers';  
  
export const handler = async (event) => {  
  try {  
    // Replace with the name or ARN of your secret  
    const secretName = "arn:aws:secretsmanager:us-east-1:111122223333:secret:SECRET_NAME";  
  
    // Resolve the session token from the credential provider chain so that  
    this  
    // works across all initialization modes, including SnapStart. Resolve  
    inside  
    // the handler so the token stays fresh after a SnapStart restore.  
    const { sessionToken } = await fromNodeProviderChain();  
  
    const options = {  
      hostname: 'localhost',
```

```

        port: 2773,
        path: `/secretsmanager/get?secretId=${secretName}`,
        headers: {
            'X-Aws-Parameters-Secrets-Token': sessionToken
        }
    };

    const response = await new Promise((resolve, reject) => {
        http.get(options, (res) => {
            let data = '';
            res.on('data', (chunk) => { data += chunk; });
            res.on('end', () => {
                resolve({
                    statusCode: res.statusCode,
                    body: data
                });
            });
        }).on('error', reject);
    });

    const secret = JSON.parse(response.body).SecretString;
    console.log('Retrieved secret:', secret);

    return {
        statusCode: response.statusCode,
        body: JSON.stringify({
            message: 'Successfully retrieved secret',
            secretRetrieved: true
        })
    };
} catch (error) {
    console.error('Error:', error);
    return {
        statusCode: 500,
        body: JSON.stringify({
            message: 'Error retrieving secret',
            error: error.message
        })
    };
}
};

```

### 3. Crea un file .zip contenente il file `index.mjs`:

```
zip -r function.zip index.mjs
```

## Java

Per creare una funzione Java

1. Crea un progetto Maven:

```
mvn archetype:generate \  
  -DgroupId=example \  
  -DartifactId=lambda-secrets-demo \  
  -DarchetypeArtifactId=maven-archetype-quickstart \  
  -DarchetypeVersion=1.4 \  
  -DinteractiveMode=false
```

2. Vai alla directory del progetto:

```
cd lambda-secrets-demo
```

3. Apri `pom.xml` e sostituisci il contenuto con quanto riportato di seguito.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://  
maven.apache.org/xsd/maven-4.0.0.xsd">  
  <modelVersion>4.0.0</modelVersion>  
  
  <groupId>example</groupId>  
  <artifactId>lambda-secrets-demo</artifactId>  
  <version>1.0-SNAPSHOT</version>  
  
  <properties>  
    <maven.compiler.source>11</maven.compiler.source>  
    <maven.compiler.target>11</maven.compiler.target>  
  </properties>  
  
  <dependencies>  
    <dependency>  
      <groupId>com.amazonaws</groupId>  
      <artifactId>aws-lambda-java-core</artifactId>  
      <version>1.2.1</version>
```

```

        </dependency>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>auth</artifactId>
            <version>2.25.0</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.4</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <createDependencyReducedPom>>false</
createDependencyReducedPom>
                            <finalName>function</finalName>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>

```

4. Rinomina `/lambda-secrets-demo/src/main/java/example/App.java` in `Hello.java` in modo che corrisponda al nome del gestore Java predefinito di Lambda (`example.Hello::handleRequest`):

```
mv src/main/java/example/App.java src/main/java/example/Hello.java
```

5. Apri il file `Hello.java` e sostituisci il contenuto con quanto riportato di seguito. Per `secretName`, utilizza il nome o il nome della risorsa Amazon (ARN) del segreto.

```
package example;
```

```

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.auth.credentials.AwsSessionCredentials;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

public class Hello implements RequestHandler<Object, String> {
    private final HttpClient client = HttpClient.newHttpClient();
    private final DefaultCredentialsProvider credentialsProvider =
        DefaultCredentialsProvider.create();

    @Override
    public String handleRequest(Object input, Context context) {
        try {
            // Replace with the name or ARN of your secret
            String secretName = "arn:aws:secretsmanager:us-
east-1:111122223333:secret:SECRET_NAME";
            String endpoint = "http://localhost:2773/secretsmanager/get?
secretId=" + secretName;

            // Resolve the session token from the credential provider chain so
            that this
            // works across all initialization modes, including SnapStart.
            Resolve inside
            // the handler so the token stays fresh after a SnapStart restore.
            AwsSessionCredentials credentials = (AwsSessionCredentials)
            credentialsProvider.resolveCredentials();
            String sessionToken = credentials.sessionToken();

            HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(endpoint))
                .header("X-Aws-Parameters-Secrets-Token", sessionToken)
                .GET()
                .build();

            HttpResponse<String> response = client.send(request,
                HttpResponse.BodyHandlers.ofString());

            String secret = response.body();
            secret = secret.substring(secret.indexOf("SecretString") + 15);
            secret = secret.substring(0, secret.indexOf("\\"));

```

```
        System.out.println("Retrieved secret: " + secret);
        return String.format(
            "{\"statusCode\": %d, \"body\": \"%s\"}",
            response.statusCode(), "Successfully retrieved secret"
        );
    } catch (Exception e) {
        e.printStackTrace();
        return String.format(
            "{\"body\": \"Error retrieving secret: %s\"}",
            e.getMessage()
        );
    }
}
```

6. Rimuovi la directory di test. Maven la crea per impostazione predefinita, ma non ne abbiamo bisogno per questo esempio.

```
rm -rf src/test
```

7. Compila il progetto:

```
mvn package
```

8. Scarica il file JAR (target/function.jar) per un uso successivo.

## Creazione della funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli Crea funzione.
3. Scegli Crea da zero.
4. Nel campo Function name (Nome funzione), immettere **secret-retrieval-demo**.
5. Scegli il runtime preferito.
6. Scegli Crea funzione.



## Per caricare il pacchetto di implementazione

1. Nella scheda Codice della funzione, scegli Carica da e seleziona il file .zip (per Python e Node.js) o .jar (per Java).
2. Carica il pacchetto di implementazione creato in precedenza.
3. Selezionare Salva.

## Aggiunta dell'estensione

### Per aggiungere AWS Parametri e segreti: estensione Lambda come livello

1. Nella scheda Codice della funzione, scorri verso il basso fino a Livelli.
2. Scegliere Add a layer (Aggiungi un livello).
3. Seleziona AWS livelli.
4. Scegli AWS- Parameters-and-Secrets-Lambda-Extension.
5. Scegli la versione più recente.
6. Scegliere Aggiungi.

## Aggiunta di autorizzazioni

### Per aggiungere i permessi di Secrets Manager al ruolo di esecuzione

1. Quindi, seleziona la scheda Configuration (Configurazione) e poi Permissions (Autorizzazioni).
2. In Nome del ruolo, scegli il link al tuo ruolo di esecuzione. Questo ruolo si apre nella console IAM.

## Execution role

### Role name

secret-retrieval-demo-role- 

3. Scegli Aggiungi autorizzazioni, quindi seleziona Crea policy in linea.

### Permissions policies (1) [Info](#)

You can attach up to 10 managed policies.

 Search

#### Filter by Type

All types 



Simulate 

Remove

Add permissions 

Attach policies

Create inline policy 

< 1 >

- Scegli la scheda JSON e aggiungi la policy seguente. Per Resource, inserisci l'ARN del tuo segreto.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "arn:aws:secretsmanager:us-east-1:111122223333:secret:SECRET_NAME"
    }
  ]
}
```

- Scegli Avanti.
- Inserisci un nome per la policy.
- Scegli Crea policy.

## Test della funzione

Per testare la funzione

- Torna alla console Lambda.
- Seleziona la scheda Test.
- Scegli Test (Esegui test). Dovrebbe essere visualizzata la seguente risposta:

✓ Executing function: succeeded ([logs](#))

### ▼ Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": 200,
  "body": "{\"message\": \"Successfully retrieved secret\", \"secretRetrieved\": true}"
}
```

## Variabili di ambiente

L'estensione AWS Parameters and Secrets Lambda utilizza le seguenti impostazioni predefinite. Puoi sovrascrivere queste impostazioni creando le variabili [ambiente corrispondenti](#). Per visualizzare le impostazioni correnti di una funzione, imposta PARAMETERS\_SECRETS\_EXTENSION\_LOG\_LEVEL su DEBUG. L'estensione registrerà le informazioni di configurazione in CloudWatch Logs all'inizio di ogni chiamata di funzione.

| Impostazione            | Valore predefinito | Valori validi   | Variabile di ambiente                      | Informazioni                                                                                                                                                                                                   |
|-------------------------|--------------------|-----------------|--------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Porta HTTP              | 2773               | 1 - 65535       | PARAMETERS_SECRETS_EXTENSION_HTTP_PORT     | Porta per il server HTTP locale                                                                                                                                                                                |
| Cache abilitata         | TRUE               | TRUE   FALSE    | PARAMETERS_SECRETS_EXTENSION_CACHE_ENABLED | Abilita o disabilita la cache                                                                                                                                                                                  |
| Dimensioni della cache  | 1000               | 0 - 1.000       | PARAMETERS_SECRETS_EXTENSION_CACHE_SIZE    | Imposta su 0 per disabilitare la memorizzazione nella cache                                                                                                                                                    |
| TTL di Secrets Manager. | 300 secondi        | 0 - 300 secondi | SECRETS_MANAGER_TTL                        | Time-to-live per i segreti memorizzati nella cache. Imposta su 0 per disabilitare la memorizzazione nella cache. Questa variabile viene ignorata se il valore per PARAMETERS_SECRETS_EXTENSION_CACHE_SIZE è 0. |
| TTL Parameter Store     | 300 secondi        | 0 - 300 secondi | SSM_PARAMETER_STORE_TTL                    | Time-to-live per i parametri memorizzati nella cache. Imposta su 0 per disabilitare la                                                                                                                         |

| Impostazione               | Valore predefinito | Valori validi                      | Variabile di ambiente                                      | Informazioni                                                                                                                            |
|----------------------------|--------------------|------------------------------------|------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
|                            |                    |                                    |                                                            | memorizzazione nella cache. Questa variabile viene ignorata se il valore per <code>PARAMETER_S_SECRETS_EXTENSION_CACHE_SIZE</code> è 0. |
| Livello di log             | INFO               | DEBUG   INFO   WARN   ERROR   NONE | <code>PARAMETER_S_SECRETS_EXTENSION_LOG_LEVEL</code>       | Il livello di dettaglio riportato nei log per l'estensione                                                                              |
| Connessioni max            | 3                  | Uguale o maggiore di 1             | <code>PARAMETER_S_SECRETS_EXTENSION_MAX_CONNECTIONS</code> | Numero massimo di connessioni HTTP per richieste a Parameter Store o Secrets Manager                                                    |
| Timeout di Secrets Manager | 0 (nessun timeout) | Tutti i numeri interi              | <code>SECRETS_MANAGER_TIMEOUT_MILLIS</code>                | Timeout per le richieste a Secrets Manager (in millisecondi)                                                                            |
| Timeout Parameter Store    | 0 (nessun timeout) | Tutti i numeri interi              | <code>SSM_PARAMETER_STORE_TIMEOUT_MILLIS</code>            | Timeout per le richieste a Parameter Store (in millisecondi).                                                                           |

## Operazioni con rotazione di un segreto

Se ruoti spesso i segreti, la durata predefinita di 300 secondi della cache potrebbe far sì che la funzione utilizzi segreti obsoleti. Sono disponibili due opzioni per garantire che la funzione utilizzi il valore segreto più recente:

- Ridurre il TTL della cache impostando la variabile di ambiente `SECRETS_MANAGER_TTL` su un valore inferiore (in secondi). Ad esempio, impostandolo su 60 garantire che la funzione non utilizzi mai un segreto vecchio più di un minuto.

- Usare le etichette temporanee `AWSCURRENT` o `AWSPREVIOUS` nella tua richiesta di segreto per assicurarti di ottenere la versione specifica che desideri:

```
secretsmanager/get?secretId=YOUR_SECRET_NAME&versionStage=AWSCURRENT
```

Scegli l'approccio che meglio bilancia le tue esigenze di prestazioni e aggiornamento. Un TTL inferiore significa chiamate più frequenti a Secrets Manager, ma garantisce l'utilizzo dei valori segreti più recenti.

## Utilizzo dell'utilità dei parametri di Powertools per AWS Lambda

L'utilità dei parametri di Powertools for AWS Lambda fornisce un'interfaccia unificata per il recupero dei segreti da più provider, tra cui Secrets Manager, parameter store e AppConfig. Gestisce la memorizzazione nella cache, le trasformazioni e offre un'esperienza di sviluppo più integrata rispetto all'approccio di estensione.

### Vantaggi dell'utilità parametri

- Provider multipli: recupera i parametri da Secrets Manager, Parameter Store e utilizza la stessa interfaccia AppConfig
- Built-in trasformazioni: analisi automatica di JSON, decodifica base64 e altre trasformazioni di dati
- Memorizzazione nella cache integrata: memorizzazione nella cache configurabile con supporto TTL per ridurre le chiamate API
- Sicurezza dei tipi: supporto avanzato per la digitazione in e altri runtime supportati TypeScript
- Gestione degli errori: logica dei Built-in tentativi e gestione degli errori

### Esempi di codice

Negli esempi seguenti viene illustrato come recuperare i segreti utilizzando l'utilità Parametri in runtime diversi:

#### Python

##### Note

Per esempi completi e istruzioni di configurazione, consulta la [documentazione dell'utilità Parametri](#).

## Recupero dei segreti da Secrets Manager con l'utilità Powertools for Parameters. AWS Lambda

```
from aws_lambda_powertools import Logger
from aws_lambda_powertools.utilities import parameters

logger = Logger()

def lambda_handler(event, context):
    try:
        # Get secret with caching (default TTL: 5 seconds)
        secret_value = parameters.get_secret("my-secret-name")

        # Get secret with custom TTL
        secret_with_ttl = parameters.get_secret("my-secret-name", max_age=300)

        # Get secret and transform JSON
        secret_json = parameters.get_secret("my-json-secret", transform="json")

        logger.info("Successfully retrieved secrets")

        return {
            'statusCode': 200,
            'body': 'Successfully retrieved secrets'
        }

    except Exception as e:
        logger.error(f"Error retrieving secret: {str(e)}")
        return {
            'statusCode': 500,
            'body': f'Error: {str(e)}'
        }
```

### TypeScript

#### Note

Per esempi completi e istruzioni di configurazione, consulta la [documentazione dell'utilità Parametri](#).

## Recupero dei segreti da Secrets Manager con l'utilità Powertools for Parameters. AWS Lambda

```
import { Logger } from '@aws-lambda-powertools/logger';
import { getSecret } from '@aws-lambda-powertools/parameters/secrets';
import type { Context } from 'aws-lambda';

const logger = new Logger();

export const handler = async (event: any, context: Context) => {
  try {
    // Get secret with caching (default TTL: 5 seconds)
    const secretValue = await getSecret('my-secret-name');

    // Get secret with custom TTL
    const secretWithTtl = await getSecret('my-secret-name', { maxAge: 300 });

    // Get secret and transform JSON
    const secretJson = await getSecret('my-json-secret', { transform: 'json' });

    logger.info('Successfully retrieved secrets');

    return {
      statusCode: 200,
      body: 'Successfully retrieved secrets'
    };
  } catch (error) {
    logger.error('Error retrieving secret', { error });
    return {
      statusCode: 500,
      body: `Error: ${error}`
    };
  }
};
```

## Java

### Note

Per esempi completi e istruzioni di configurazione, consulta la [documentazione dell'utilità Parametri](#).

Recupero dei segreti da Secrets Manager con l'utilità Powertools for Parameters. AWS Lambda

```
import software.amazon.lambda.powertools.logging.Logging;
import software.amazon.lambda.powertools.parameters.SecretsProvider;
import software.amazon.lambda.powertools.parameters.ParamManager;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class SecretHandler implements RequestHandler<Object, String> {

    private final SecretsProvider secretsProvider =
        ParamManager.getSecretsProvider();

    @Logging
    @Override
    public String handleRequest(Object input, Context context) {
        try {
            // Get secret with caching (default TTL: 5 seconds)
            String secretValue = secretsProvider.get("my-secret-name");

            // Get secret with custom TTL (300 seconds)
            String secretWithTtl = secretsProvider.withMaxAge(300).get("my-secret-
name");

            // Get secret and transform JSON
            MySecret secretJson = secretsProvider.get("my-json-secret",
MySecret.class);

            return "Successfully retrieved secrets";

        } catch (Exception e) {
            return "Error retrieving secret: " + e.getMessage();
        }
    }

    public static class MySecret {
        // Define your secret structure here
    }
}
```



## .NET

### Note

Per esempi completi e istruzioni di configurazione, consulta la [documentazione dell'utilità Parametri](#).

## Recupero dei segreti da Secrets Manager con l'utilità Powertools for Parameters. AWS Lambda

```
using AWS.Lambda.Powertools.Logging;
using AWS.Lambda.Powertools.Parameters;
using Amazon.Lambda.Core;

[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

public class Function
{
    private readonly ISecretsProvider _secretsProvider;

    public Function()
    {
        _secretsProvider = ParametersManager.SecretsProvider;
    }

    [Logging]
    public async Task<string> FunctionHandler(object input, ILambdaContext context)
    {
        try
        {
            // Get secret with caching (default TTL: 5 seconds)
            var secretValue = await _secretsProvider.GetAsync("my-secret-name");

            // Get secret with custom TTL
            var secretWithTtl = await
                _secretsProvider.WithMaxAge(TimeSpan.FromMinutes(5))
                    .GetAsync("my-secret-name");

            // Get secret and transform JSON
            var secretJson = await _secretsProvider.GetAsync<MySecret>("my-json-secret");
        }
    }
}
```

```
        return "Successfully retrieved secrets";
    }
    catch (Exception e)
    {
        return $"Error retrieving secret: {e.Message}";
    }
}

public class MySecret
{
    // Define your secret structure here
}
}
```

## Configurazione e autorizzazioni

Per utilizzare l'utilità Parametri, devi:

1. Installa Powertools for per AWS Lambda il tuo runtime. Per informazioni dettagliate, vedi [Utensili elettrici per AWS Lambda](#).
2. Aggiungere le autorizzazioni IAM necessarie al ruolo di esecuzione della tua funzione. Fare riferimento a [Gestione delle autorizzazioni in AWS Lambda](#) per ulteriori dettagli.
3. Configura eventuali impostazioni opzionali tramite [variabili di ambiente](#).

Le autorizzazioni IAM richieste sono le stesse dell'approccio di estensione. L'utilità gestisce automaticamente la memorizzazione nella cache e le chiamate API a Secrets Manager in base alla configurazione.

## Utilizzo di Lambda con Amazon SQS

### Note

Se desideri inviare dati a una destinazione diversa da una funzione Lambda o arricchire i dati prima di inviarli, consulta [Amazon EventBridge Pipes](#).

È possibile utilizzare una funzione Lambda per elaborare i messaggi in una coda Amazon Simple Queue Service (Amazon SQS). Lambda supporta sia [code standard](#) che [code first-in, first-out \(FIFO\)](#) per gli [strumenti di mappatura dell'origine degli eventi](#). Puoi anche utilizzare la modalità provisioned per allocare risorse di polling dedicate per le mappature delle sorgenti degli eventi Amazon SQS. La funzione Lambda e la coda Amazon SQS devono essere uguali, anche se possono essere diverse. Regione AWS [Account AWS](#)

Per un esempio di funzione che elabora i messaggi Amazon SQS, consulta. [the section called “Tutorial”](#) Per l'elaborazione su più account, consulta. [the section called “Tutorial su SQS tra più account”](#) Per un elenco completo delle opzioni di configurazione, consulta [the section called “Parameters”](#).

Durante l'elaborazione dei messaggi Amazon SQS, è necessario implementare una logica di risposta batch parziale per evitare che i messaggi elaborati correttamente vengano ritentati quando alcuni messaggi in un batch falliscono. L'utilità [Batch Processor](#) di Powertools for AWS Lambda semplifica questa implementazione gestendo automaticamente la logica di risposta batch parziale, riducendo i tempi di sviluppo e migliorando l'affidabilità.

### Argomenti

- [Informazioni sul comportamento di polling e batch per gli strumenti di mappatura dell'origine degli eventi di Amazon SQS](#)
- [Utilizzo della modalità provisioned con le mappature delle sorgenti di eventi di Amazon SQS](#)
- [Configurazione della modalità di provisioning per la mappatura delle sorgenti di eventi di Amazon SQS](#)
- [Esempio di evento con messaggio di coda standard](#)
- [Esempio di evento di messaggio di coda FIFO](#)
- [Creazione e configurazione di uno strumento di mappatura dell'origine degli eventi Amazon SQS](#)
- [Configurazione del comportamento di dimensionamento per gli strumenti di mappatura dell'origine degli eventi](#)

- [Gestione degli errori per un'origine eventi SQS in Lambda](#)
- [Parametri Lambda per gli strumenti di mappatura dell'origine degli eventi di Amazon SQS](#)
- [Utilizzo del filtro eventi con un'origine eventi Amazon SQS](#)
- [Tutorial: Utilizzo di Lambda con Amazon SQS](#)
- [Tutorial: utilizzo di una coda Amazon SQS tra più account come origine eventi](#)

## Informazioni sul comportamento di polling e batch per gli strumenti di mappatura dell'origine degli eventi di Amazon SQS

Con gli strumenti di mappatura dell'origine degli eventi di Amazon SQS, Lambda esegue il polling della coda e richiama la funzione [in modo sincrono](#) con un evento. Ogni evento può contenere un batch di più messaggi dalla coda. Lambda riceve questi eventi un batch alla volta e richiama la funzione una volta per ogni batch. Quando la funzione elabora correttamente un batch, Lambda elimina i relativi messaggi dalla coda.

Quando Lambda legge un batch, i messaggi rimangono nella coda ma vengono nascosti per la durata del [timeout visibilità](#) della coda. Se la funzione elabora correttamente tutti i messaggi nel batch, Lambda elimina i messaggi dalla coda. Per impostazione predefinita, se la funzione rileva un errore durante l'elaborazione di un batch, una volta scaduto il timeout di visibilità tutti i messaggi in quel batch diventeranno nuovamente visibili nella coda. Per questo motivo, il codice della funzione deve riuscire a elaborare lo stesso messaggio più volte, senza intoppi indesiderati.

### Warning

Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per saperne di più, vedi [Come posso rendere idempotente](#) la mia funzione Lambda nel Knowledge Center. AWS

Per evitare che Lambda elabori un messaggio più volte, puoi configurare la mappatura dell'origine degli eventi in modo da includere gli errori degli elementi del [batch](#) nella risposta della funzione oppure puoi utilizzare l'[DeleteMessage](#) API per rimuovere i messaggi dalla coda man mano che la funzione Lambda li elabora correttamente.

Per ulteriori informazioni sui parametri di configurazione supportati da Lambda per gli strumenti di mappatura dell'origine degli eventi di SQS, consulta [the section called “Creazione di uno strumento di mappatura dell'origine degli eventi SQS”](#).

## Utilizzo della modalità provisioned con le mappature delle sorgenti di eventi di Amazon SQS

Per i carichi di lavoro in cui è necessario ottimizzare il throughput dello strumento di mappatura dell'origine degli eventi, è possibile utilizzare la modalità provisioning. In modalità provisioning, vengono definiti i limiti minimi e massimi per la quantità di poller di eventi assegnati. Questi poller di eventi con provisioning sono dedicati allo strumento di mappatura dell'origine degli eventi e possono gestire picchi di messaggi imprevisti tramite un dimensionamento automatico reattivo. La mappatura delle sorgenti di eventi di Amazon SQS configurata con la modalità Provisioned è 3 volte più veloce (fino a 1.000 richiami simultanei al minuto) e supporta una concorrenza 80 volte superiore (fino a 100.000 richiami simultanei) rispetto alla funzionalità predefinita di mappatura delle sorgenti di eventi di Amazon SQS.

Ti consigliamo di utilizzare la modalità provisioned per carichi di lavoro basati su eventi di Amazon SQS con requisiti prestazionali rigorosi, come società di servizi finanziari che elaborano feed di dati di mercato, piattaforme di e-commerce che forniscono consigli personalizzati in tempo reale e società di giochi che gestiscono le interazioni con i giocatori dal vivo. L'utilizzo della modalità provisioning comporta costi aggiuntivi. Per i prezzi dettagliati, consulta la sezione prezzi. [AWS Lambda](#)

Ogni poller di eventi in modalità provisioned può gestire fino all'1% MB/s del throughput, fino a 10 richiami simultanei o fino a 10 chiamate API di polling di Amazon SQS al secondo. L'intervallo di valori accettati per il numero minimo di event pollers (MinimumPollers) è compreso tra 2 e 200, con un valore predefinito di 2. L'intervallo di valori accettati per il numero massimo di event pollers (MaximumPollers) è compreso tra 2 e 10.000, con il valore predefinito di 200. MaximumPollers deve essere maggiore o uguale a MinimumPollers.

### Determinazione dei poller di eventi richiesti

Per stimare il numero di poller di eventi necessari per garantire prestazioni ottimali di elaborazione dei messaggi quando si utilizza la modalità provisioning per SQS ESM, raccogliete le seguenti metriche per la vostra applicazione: picchi di eventi SQS al secondo che richiedono un'elaborazione a bassa latenza, dimensione media del payload degli eventi SQS, durata media della funzione Lambda e dimensione del batch configurato.

Innanzitutto puoi stimare il numero di eventi SQS al secondo (EPS) supportati da un event poller per il tuo carico di lavoro utilizzando la seguente formula:

```
EPS per event poller =  
    minimum(  
        ceiling(1024 / average event size in KB),  
        ceiling(10 / average function duration in seconds) * batch size,  
        min(100, 10 * batch size)  
    )
```

Quindi, puoi calcolare il numero minimo di poller richiesti utilizzando la formula seguente. Questo calcolo garantisce la disponibilità di una capacità sufficiente per gestire i requisiti di traffico di punta.

```
Required event pollers = (Peak number of events per second in Queue) / EPS per event  
poller
```

Considerate un carico di lavoro con una dimensione batch predefinita di 10, una dimensione media degli eventi di 3 KB, una durata media della funzione di 100 ms e un requisito per gestire 1.000 eventi al secondo. In questo scenario, ogni poller di eventi supporta circa 100 eventi al secondo (EPS). Pertanto, ti consigliamo di impostare un poller minimo su 10 per gestire adeguatamente i requisiti relativi ai picchi di traffico. Se il carico di lavoro presenta le stesse caratteristiche ma con una durata media della funzione di 1 secondo, ogni poller supporta solo 10 EPS, richiedendo la configurazione di almeno 100 poller per supportare 1.000 eventi al secondo a bassa latenza.

Consigliamo di utilizzare una dimensione del batch predefinita pari o superiore a 10 per massimizzare l'efficienza dei poller di eventi in modalità provisioning. Le dimensioni dei batch più elevate consentono a ciascun poller di elaborare più eventi per chiamata, migliorando la produttività e l'efficienza dei costi. Quando pianificate la capacità del vostro event poller, tenete conto dei potenziali picchi di traffico e valutate la possibilità di impostare il valore `MinimumPollers` leggermente superiore al valore minimo calcolato per fornire un buffer. Inoltre, monitorate le caratteristiche del carico di lavoro nel tempo, poiché le modifiche nella dimensione dei messaggi, nella durata delle funzioni o nei modelli di traffico potrebbero richiedere modifiche alla configurazione dell'event poller per mantenere prestazioni ottimali ed efficienza dei costi. Per una pianificazione precisa della capacità, consigliamo di testare il carico di lavoro specifico per determinare l'effettivo EPS generato da ciascun event poller.

## Configurazione della modalità di provisioning per la mappatura delle sorgenti di eventi di Amazon SQS

Puoi configurare la modalità di provisioning per la mappatura delle sorgenti di eventi di Amazon SQS utilizzando la console o l'API Lambda.

Per configurare la modalità di provisioning per una mappatura delle sorgenti di eventi Amazon SQS esistente (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione con la mappatura delle sorgenti di eventi di Amazon SQS per la quale desideri configurare la modalità di provisioning.
3. Scegli la scheda Configurazione, quindi scegli Trigger.
4. Scegli la mappatura delle sorgenti di eventi di Amazon SQS per la quale desideri configurare la modalità di provisioning, quindi scegli Modifica.
5. In Configurazione dello strumento di mappatura dell'origine degli eventi, scegli Configura la modalità provisioning.
  - Per i poller di eventi minimi, inserisci un valore compreso tra 2 e 200. Se non specifichi un valore, Lambda sceglie un valore predefinito pari a 2.
  - Per i sondaggi relativi al numero massimo di eventi, inserisci un valore compreso tra 2 e 10.000. Questo valore deve essere maggiore o uguale al valore specificato in Numero minimo di poller di eventi. Se non si specifica un valore, Lambda assegna il valore predefinito 200.
6. Scegli Save (Salva).

È possibile configurare la modalità provisioning a livello di codice utilizzando l'oggetto nel ProvisionedPollerConfig file. EventSourceMappingConfiguration Ad esempio, il seguente comando UpdateEventSourceMapping CLI configura un MinimumPollers valore di 5 e un valore di 100. MaximumPollers

```
aws lambda update-event-source-mapping \  
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \  
  --provisioned-poller-config '{"MinimumPollers": 5, "MaximumPollers": 100}'
```

Dopo aver configurato la modalità provisioning, puoi osservare l'utilizzo dei poller di eventi per il tuo carico di lavoro monitorando il parametro ProvisionedPollers. Per ulteriori informazioni, consulta Metriche di mappatura dell'origine degli eventi.

Per disabilitare la modalità predisposta e tornare alla modalità predefinita (su richiesta), puoi utilizzare il seguente comando CLI: `UpdateEventSourceMapping`

```
aws lambda update-event-source-mapping \  
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \  
  --provisioned-poller-config '{}'
```

#### Note

La modalità provisioned non può essere utilizzata insieme all'impostazione di concorrenza massima. Quando si utilizza la modalità provisioning, si controlla la massima concorrenza tramite il numero massimo di poller di eventi.

Per ulteriori informazioni sulla configurazione della modalità con provisioning, vedere. [the section called “Creare lo strumento di mappatura”](#)

## Esempio di evento con messaggio di coda standard

Example Evento messaggio Amazon SQS (coda standard)

```
{  
  "Records": [  
    {  
      "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",  
      "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgXlaS3SLy0a...",  
      "body": "Test message.",  
      "attributes": {  
        "ApproximateReceiveCount": "1",  
        "SentTimestamp": "1545082649183",  
        "SenderId": "AIDAIENQZJOL023YVJ4V0",  
        "ApproximateFirstReceiveTimestamp": "1545082649185"  
      },  
      "messageAttributes": {  
        "myAttribute": {  
          "stringValue": "myValue",  
          "stringListValues": [],  
          "binaryListValues": [],  
          "dataType": "String"  
        }  
      },  
    },  
  ],  
}
```



```

        "md50fBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
        "eventSource": "aws:sqs",
        "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
        "awsRegion": "us-east-2"
    },
    {
        "messageId": "2e1424d4-f796-459a-8184-9c92662be6da",
        "receiptHandle": "AQEBzWwafTRI0KuVm4tP+/7q1rGgNqicHq...",
        "body": "Test message.",
        "attributes": {
            "ApproximateReceiveCount": "1",
            "SentTimestamp": "1545082650636",
            "SenderId": "AIDAIENQZJOL023YVJ4V0",
            "ApproximateFirstReceiveTimestamp": "1545082650649"
        },
        "messageAttributes": {},
        "md50fBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
        "eventSource": "aws:sqs",
        "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
        "awsRegion": "us-east-2"
    }
]
}

```

Per impostazione predefinita, Lambda eseguirà il polling di un massimo di 10 messaggi contemporaneamente nella coda e invierà il batch alla funzione. Per evitare di richiamare la funzione con un piccolo numero di record, è possibile configurare l'origine eventi per memorizzare nel buffer i record per un massimo di 5 minuti definendo un periodo di batch. Prima di richiamare la funzione, Lambda continua a eseguire il polling dei messaggi dalla coda standard fino alla scadenza del periodo di batch, al raggiungimento della [quota della dimensione del payload di richiesta](#) o al raggiungimento della dimensione massima per la configurazione di un batch.

Se stai utilizzando un periodo di batch e la tua coda SQS contiene un traffico molto basso, Lambda potrebbe attendere fino a 20 secondi prima di richiamare la tua funzione. Lo stesso vale anche se imposti un periodo di batch inferiore a 20 secondi.

#### Note

In Java, potresti riscontrare errori di puntatore null durante la deserializzazione di JSON. Ciò potrebbe essere dovuto al modo in cui "Records" e "eventSourceARN" vengono convertiti dal mappatore di oggetti JSON.

## Esempio di evento di messaggio di coda FIFO

Per le code FIFO, i record contengono attributi aggiuntivi correlati alla deduplicazione e al sequenziamento.

### Example Evento messaggio Amazon SQS (coda FIFO)

```
{
  "Records": [
    {
      "messageId": "11d6ee51-4cc7-4302-9e22-7cd8afdaadf5",
      "receiptHandle": "AQEBBx8nesZEXmkhsmZeyIE8iQAMig7qw...",
      "body": "Test message.",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1573251510774",
        "SequenceNumber": "18849496460467696128",
        "MessageGroupId": "1",
        "SenderId": "AIDAI023YVJENQZJOL4V0",
        "MessageDeduplicationId": "1",
        "ApproximateFirstReceiveTimestamp": "1573251510774"
      },
      "messageAttributes": {},
      "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
      "eventSource": "aws:sqs",
      "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:fifo.fifo",
      "awsRegion": "us-east-2"
    }
  ]
}
```

## Creazione e configurazione di uno strumento di mappatura dell'origine degli eventi Amazon SQS

Per elaborare i messaggi Amazon SQS con Lambda, configura la coda con le impostazioni appropriate, quindi crea uno strumento di mappatura dell'origine degli eventi Lambda.

### Argomenti

- [Configurazione di una coda da utilizzare con Lambda](#)
- [Impostazione delle autorizzazioni del ruolo di esecuzione Lambda](#)
- [Creazione di uno strumento di mappatura dell'origine degli eventi SQS](#)

## Configurazione di una coda da utilizzare con Lambda

Se non disponi già di una coda Amazon SQS esistente, [creane una](#) da utilizzare come origine eventi per la funzione Lambda. La funzione Lambda e la coda Amazon SQS devono essere uguali Regione AWS, anche se possono essere diverse. [Account AWS](#)

Per concedere alla funzione il tempo necessario per elaborare ogni batch di record, imposta il [timeout di visibilità](#) della coda di origine su un valore pari ad almeno sei volte il [timeout di configurazione](#) per la funzione. Il tempo aggiuntivo consente a Lambda di riprovare se l'esecuzione della funzione viene limitata durante l'elaborazione di un batch precedente.

### Note

Il timeout della funzione deve essere inferiore o uguale al timeout di visibilità della coda. Lambda convalida questo requisito quando si crea o si aggiorna una mappatura dell'origine di un evento e restituisce un errore se il timeout della funzione supera il timeout di visibilità della coda.

Per impostazione predefinita, se Lambda rileva un errore in qualsiasi momento durante l'elaborazione di un batch, tutti i messaggi in quel batch ritorneranno nella coda. Dopo il [timeout di visibilità](#), i messaggi diventano nuovamente visibili a Lambda. È possibile configurare lo strumento di mappatura dell'origine degli eventi in modo da utilizzare [risposte batch parziali](#) per restituire alla coda solo i messaggi non riusciti.

Se la funzione dovesse non riuscire più volte a elaborare un messaggio, Amazon SQS può inviarlo a una [coda DLQ](#). Ti consigliamo di impostare `maxReceiveCount` sulla [policy redrive](#) della coda di origine su un valore pari almeno a 5. Ciò offre a Lambda alcune possibilità di riprovare prima di inviare i messaggi non riusciti direttamente alla coda DLQ.

## Impostazione delle autorizzazioni del ruolo di esecuzione Lambda

La policy [AWSLambdaSQSQueueExecutionRole](#) AWS gestita include le autorizzazioni necessarie a Lambda per leggere dalla coda Amazon SQS. Puoi aggiungere questa policy gestita al [ruolo di esecuzione](#) della tua funzione.

Facoltativamente, se utilizzi una coda crittografata, devi anche aggiungere la seguente autorizzazione al tuo ruolo di esecuzione:

- [kms:Decrypt](#)

## Creazione di uno strumento di mappatura dell'origine degli eventi SQS

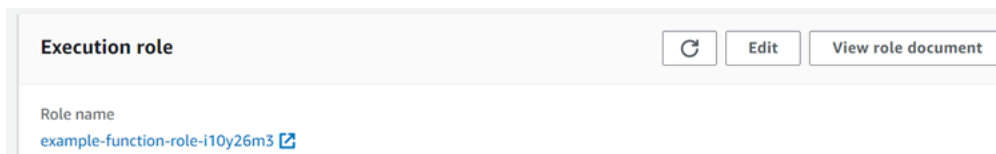
Creare una mappatura dell'origine eventi per indicare a Lambda di inviare le voci dalla coda a una funzione Lambda. È possibile creare più mappature delle origini eventi per elaborare elementi da più code con una singola funzione. Quando Lambda richiama la funzione di destinazione, l'evento può contenere più voci, fino a una dimensione batch massima configurabile.

Per configurare la tua funzione per la lettura da Amazon SQS, collega la policy

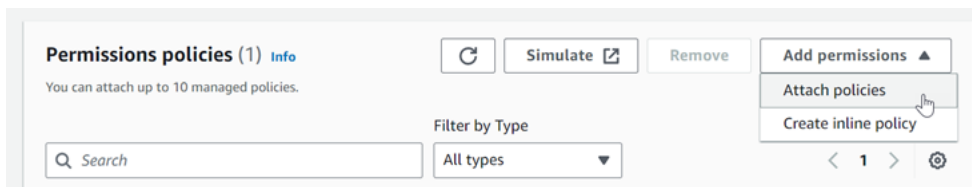
[AWSLambdaSQSQueueExecutionRole](#) AWS gestita al tuo ruolo di esecuzione. Quindi, crea uno strumento di mappatura dell'origine degli eventi SQS dalla console utilizzando i seguenti passaggi.

Per aggiungere le autorizzazioni e creare un trigger

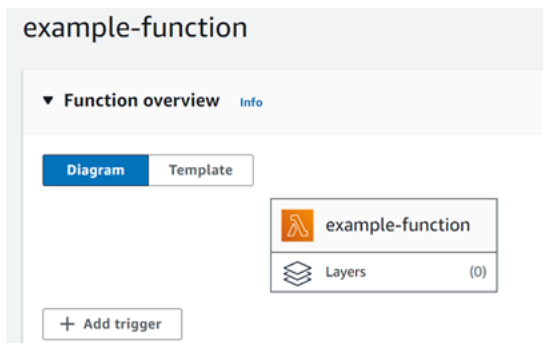
1. Aprire la pagina [Functions](#) (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Quindi, seleziona la scheda Configuration (Configurazione) e poi Permissions (Autorizzazioni).
4. In Nome del ruolo, scegli il link al tuo ruolo di esecuzione. Questo ruolo si apre nella console IAM.



5. Seleziona Aggiungi autorizzazioni, quindi seleziona Collega policy.



6. Inserisci `AWSLambdaSQSQueueExecutionRole` nel campo di ricerca. Aggiungi questa policy al tuo ruolo di esecuzione. Si tratta di una policy AWS gestita che contiene le autorizzazioni necessarie alla funzione per leggere da una coda Amazon SQS. Per ulteriori informazioni su questa policy, consulta il Managed Policy [AWSLambdaSQSQueueExecutionRole](#) Reference. AWS
7. Torna alla funzione nella console Lambda. In Panoramica delle funzioni, scegliere Aggiungi trigger.



8. Scegliere un tipo di trigger.
9. Configurare le opzioni richieste, quindi scegliere Add (Aggiungi).

Lambda supporta le seguenti opzioni di configurazione per le origini eventi di Amazon SQS:

### Coda SQS

La coda Amazon SQS da cui leggere i record. La funzione Lambda e la coda Amazon SQS devono essere uguali Regione AWS, anche se possono essere diverse. [Account AWS](#)

### Abilita trigger

Lo stato dello strumento di mappatura dell'origine degli eventi. L'opzione Enable trigger (Abilita trigger) è selezionata per impostazione predefinita.

### Dimensione batch

Il numero massimo di record da inviare alla funzione in ogni batch. Per una coda standard il numero di registri può arrivare fino a 10.000. Per una coda FIFO il massimo è 10. Per un batch di dimensioni superiori a 10, è inoltre necessario la finestra batch (MaximumBatchingWindowInSeconds) su almeno 1 secondo.

Configura il [timeout della funzione](#) per consentire il tempo necessario all'elaborazione di un intero batch di elementi. Se gli elementi richiedono tempi di elaborazione più lunghi, scegli un batch di dimensioni più piccole. Un batch di grandi dimensioni può migliorare l'efficienza per i carichi di lavoro molto veloci o con costi di gestione molto elevati. Se si configura la [concorrenza riservata](#) sulla funzione, impostare un minimo di cinque esecuzioni simultanee per ridurre le probabilità di errori di limitazione (della larghezza di banda della rete) quando Lambda richiama la funzione.

Lambda passa tutti i registri del batch alla funzione in una singola chiamata, purché la dimensione totale degli eventi non superi la [quota della dimensione del payload di invocazione](#) per una invocazione sincrona (6 MB). Per ogni registro, vengono generati metadati sia da Lambda

che da Amazon SQS. Questi metadati aggiuntivi vengono conteggiati per la dimensione totale del payload e possono far sì che il numero totale di registri inviati in un batch sia inferiore alla dimensione del batch configurato. I campi di metadati inviati da Amazon SQS possono essere di lunghezza variabile. Per ulteriori informazioni sui campi di metadati di Amazon SQS, consulta la documentazione sul funzionamento dell'API nell'Amazon Simple [ReceiveMessage](#) Queue Service API Reference.

## Finestra batch

Il tempo massimo in secondi per la raccolta dei record prima di richiamare la funzione. Questo parametro si applica solo alle code standard.

Se si utilizza un periodo batch superiore a 0 secondi, è necessario tenere conto dell'aumento del tempo di elaborazione nel [timeout di visibilità](#) della coda. Si consiglia di impostare il timeout di visibilità della coda ad un tempo sei volte maggiore rispetto al [timeout della funzione](#), più il valore di `MaximumBatchingWindowInSeconds`. Ciò consente alla funzione Lambda di elaborare ogni batch di eventi e riprovare in caso di errore di throttling.

Quando i messaggi diventano disponibili, Lambda avvia l'elaborazione dei messaggi in batch. Lambda inizia a elaborare cinque batch alla volta con cinque chiamate simultanee della funzione. Se i messaggi sono ancora disponibili, Lambda aggiunge fino a 300 richiami simultanei della tua funzione al minuto, fino a un massimo di 1.250 richiami simultanei.

Quando si utilizza la modalità provisioned, ogni poller di eventi può gestire fino all'1% MB/s del throughput, fino a 10 richiami simultanei o fino a 10 chiamate API di polling di Amazon SQS al secondo. Lambda scala il numero di poller di eventi tra il minimo e il massimo configurati, sommando rapidamente fino a 1.000 invocazioni simultanee al minuto per fornire un'elaborazione a bassa latenza degli eventi Amazon SQS. Puoi controllare la scalabilità e la concorrenza tramite queste impostazioni minime e massime del poller di eventi. Per saperne di più sul ridimensionamento e sulla concorrenza delle funzioni, consulta [Informazioni sulla scalabilità della funzione Lambda](#)

Per elaborare più messaggi, puoi ottimizzare la funzione Lambda per un maggiore throughput. Per ulteriori informazioni, consulta [Understanding how AWS Lambda scales with Amazon SQS standard queues](#).

## Criteri di filtro

Aggiungi i criteri di filtro per controllare gli eventi che Lambda invia alla funzione per l'elaborazione. Per ulteriori informazioni, consulta [Controllare gli eventi che Lambda invia alla funzione](#).

## Simultaneità massima

Il numero massimo di funzioni simultanee che l'origine eventi può richiamare. Non può essere utilizzato con la modalità provvisoria abilitata. Per ulteriori informazioni, consulta [Configurazione della simultaneità massima per le origini eventi di Amazon SQS](#).

## Modalità assegnata

Se abilitata, alloca risorse di polling dedicate per la mappatura della sorgente dell'evento. È possibile configurare il numero minimo (2-200) e massimo (2-2000) di poller di eventi. Ogni poller di eventi può gestire fino all'1% MB/sec del throughput, fino a 10 richiami simultanei o fino a 10 chiamate API di polling Amazon SQS al secondo.

### Note

Nota: non è possibile utilizzare contemporaneamente Provisioned Mode e Maximum concurrency. Quando la modalità provvisoria è abilitata, utilizzate l'impostazione del numero massimo di pollers per controllare la concorrenza.

## Configurazione del comportamento di dimensionamento per gli strumenti di mappatura dell'origine degli eventi

Puoi controllare il comportamento di scalabilità delle tue mappature delle sorgenti di eventi di Amazon SQS tramite le impostazioni di concorrenza massima o abilitando la modalità di provisioning. Si tratta di opzioni che si escludono a vicenda.

Per impostazione predefinita, Lambda ridimensiona automaticamente i poller di eventi in base al volume dei messaggi. Quando si attiva la modalità con provisioning, si assegna un numero minimo e massimo di risorse di polling dedicate che rimangono pronte a gestire i modelli di traffico previsti. Puoi ottimizzare le prestazioni della mappatura della sorgente degli eventi in due modi:

- Modalità standard (impostazione predefinita): Lambda gestisce automaticamente il ridimensionamento, partendo da un numero limitato di poller e aumentando o diminuendo la scala in base al carico di lavoro.
- Modalità provisioned: si configurano risorse di polling dedicate con limiti minimi e massimi, consentendo una scalabilità 3 volte più veloce e una capacità di elaborazione fino a 16 volte superiore.

Per le code standard, Lambda utilizza il [long polling](#) per eseguire il polling di una coda fino a quando questa non diventa attiva. Quando sono disponibili dei messaggi, Lambda inizia a elaborare cinque batch alla volta con cinque chiamate simultanee della funzione. Se i messaggi sono ancora disponibili, Lambda aumenta il numero di processi che leggono i batch fino a 300 richiami simultanei in più al minuto. Il numero massimo di richiami che una mappatura della sorgente di eventi può elaborare contemporaneamente è 1.250. Quando il traffico è basso, Lambda riduce l'elaborazione a cinque richiami simultanei e può ottimizzarli fino a un minimo di 2 richiami simultanei per ridurre le chiamate Amazon SQS e i costi corrispondenti. Tuttavia, questa ottimizzazione non è disponibile quando si abilita l'impostazione di concorrenza massima.

Per le code FIFO, Lambda invia messaggi alla funzione nell'ordine in cui li riceve. Quando invii un messaggio a una coda FIFO, è necessario specificare un [ID gruppo di messaggi](#). Amazon SQS garantisce che i messaggi dello stesso gruppo vengano consegnati a Lambda in ordine. Quando Lambda legge i messaggi in batch, ogni batch può contenere messaggi provenienti da più di un gruppo di messaggi, ma l'ordine dei messaggi viene mantenuto. Se la funzione restituisce un errore, questa esegue nuovamente tutti i tentativi necessari sui messaggi interessati, prima che Lambda riceva ulteriori messaggi dallo stesso gruppo.

Quando si utilizza la modalità provisioned, ogni poller di eventi può gestire fino all'1% MB/sec del throughput, fino a 10 richiami simultanei o fino a 10 chiamate API di polling Amazon SQS al secondo. Lambda scala il numero di poller di eventi tra il minimo e il massimo configurati, sommando rapidamente fino a 1.000 simultanee al minuto per fornire un'elaborazione coerente e a bassa latenza degli eventi Amazon SQS. L'utilizzo della modalità provisioning comporta costi aggiuntivi. Per i prezzi dettagliati, consulta la sezione prezzi. [AWS Lambda](#)

Ogni event poller utilizza un [lungo processo di polling](#) sulla coda SQS, con un massimo di 10 sondaggi al secondo, il che comporta un costo per le richieste di API SQS. <https://aws.amazon.com/sqs/pricing/> Consulta i prezzi di Amazon SQS per i dettagli. Puoi controllare la scalabilità e la concorrenza tramite queste impostazioni minime e massime del poller degli eventi, anziché utilizzare l'impostazione di concorrenza massima, poiché queste opzioni non possono essere utilizzate insieme.

#### Note

Non è possibile utilizzare contemporaneamente l'impostazione di concorrenza massima e la modalità di provisioning. Quando la modalità di provisioning è abilitata, controlla la scalabilità e la concorrenza della mappatura delle sorgenti di eventi di Amazon SQS tramite il numero minimo e massimo di poller di eventi.



## Configurazione della simultaneità massima per le origini eventi di Amazon SQS

È possibile utilizzare l'impostazione di simultaneità massima per controllare il comportamento di dimensionamento delle origini eventi SQS. Tieni presente che la concorrenza massima non può essere utilizzata con la modalità di provisioning abilitata. L'impostazione della simultaneità massima limita il numero di istanze simultanee della funzione che l'origine dell'evento Amazon SQS può richiamare. La simultaneità massima è un'impostazione a livello di origine dell'evento. Se disponi di più origini degli eventi Amazon SQS mappate a una funzione, ogni origine di evento può avere un'impostazione di simultaneità massima separata.

È possibile utilizzare la simultaneità massima per evitare che una coda utilizzi tutta la [simultaneità riservata](#) della funzione o il resto della [quota di simultaneità dell'account](#). Non è previsto alcun addebito per la configurazione della simultaneità massima su un'origine di eventi Amazon SQS.

È importante sottolineare che la simultaneità massima e la simultaneità riservata sono due impostazioni indipendenti. Non è possibile impostare la simultaneità massima su un valore maggiore della simultaneità riservata della funzione. Dopo aver configurato la simultaneità massima, assicurati di non ridurre la simultaneità riservata della funzione a un valore inferiore alla simultaneità massima totale per tutte le origini di eventi Amazon SQS sulla funzione. Altrimenti, Lambda potrebbe limitare i messaggi.

Quando la quota di simultaneità del tuo account è impostata sul valore predefinito di 1.000, uno strumento di mappatura dell'origine degli eventi Amazon SQS può scalare per richiamare istanze di funzioni fino a questo valore, a meno che non si specifichi una simultaneità massima.

Se ricevi un aumento della quota di concorrenza predefinita del tuo account, Lambda potrebbe non essere in grado di richiamare istanze di funzioni simultanee fino alla nuova quota. Per impostazione predefinita, Lambda può scalare per richiamare fino a 1.250 istanze di funzioni simultanee per uno strumento di mappatura dell'origine degli eventi Amazon SQS. Se ciò non è sufficiente per il tuo caso d'uso, contatta l'AWS assistenza per discutere di un aumento della concorrenza di mappatura delle sorgenti di eventi Amazon SQS del tuo account.

### Note

Per le code FIFO, le chiamate simultanee sono limitate dal numero di [ID dei gruppi di messaggi](#) (messageGroupId) o dall'impostazione massima di simultaneità, a seconda di quale sia il valore più basso. Ad esempio, se hai sei ID di gruppi di messaggi e la simultaneità massima è impostata su 10, la tua funzione può avere un massimo di sei chiamate simultanee.

Puoi configurare la simultaneità massima sugli strumenti di mappatura dell'origine degli eventi Amazon SQS nuovi ed esistenti.

### Configurazione della simultaneità massima tramite la console Lambda

1. Aprire la pagina [Functions](#) (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. In Function overview (Panoramica delle funzioni), scegli SQS. Viene aperta la scheda Configuration (Configurazione).
4. Seleziona il trigger Amazon SQS e scegli Edit (Modifica).
5. In Maximum concurrency (Simultaneità massima), inserisci un numero compreso tra 2 e 1.000. Per disattivare la simultaneità massima, lascia la casella vuota.
6. Scegli Save (Salva).

### Configura la massima concorrenza utilizzando AWS Command Line Interface (AWS CLI)

Usa il comando [update-event-source-mapping](#) con l'opzione `--scaling-config`. Esempio:

```
aws lambda update-event-source-mapping \  
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \  
  --scaling-config '{"MaximumConcurrency":5}'
```

Per disattivare la simultaneità massima, inserisci un valore vuoto per `--scaling-config`:

```
aws lambda update-event-source-mapping \  
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \  
  --scaling-config "{}"
```

### Configurazione della simultaneità massima tramite l'API Lambda

Usa l'[UpdateEventSourceMapping](#)azione [CreateEventSourceMapping](#) o con un [ScalingConfig](#) oggetto.

### Gestione degli errori per un'origine eventi SQS in Lambda

Per gestire gli errori relativi a un'origine eventi SQS, Lambda utilizza automaticamente una strategia di riprova con una strategia di backoff. Puoi anche personalizzare il comportamento di gestione degli errori configurando lo strumento di mappatura dell'origine degli eventi SQS per restituire [risposte parziali in batch](#).

## Strategia di backoff per le chiamate non riuscite

Quando una chiamata non riesce, Lambda riprovare la chiamata mentre implementa una strategia di backoff. La strategia di backoff varia leggermente a seconda che Lambda abbia riscontrato l'errore a causa di un errore nel codice della funzione o a causa di una limitazione.

- Se il codice della funzione ha causato l'errore, Lambda interrompe l'elaborazione e ritenta l'invocazione. Allo stesso tempo, Lambda si interrompe gradualmente, riducendo la quantità di simultaneità allocata allo strumento di mappatura dell'origine degli eventi di Amazon SQS. Allo scadere del timeout di visibilità della coda, il messaggio riappare nella coda.
- Se la chiamata non riesce a causa della limitazione, Lambda interrompe gradualmente i nuovi tentativi riducendo la quantità di simultaneità allocata allo strumento di mappatura dell'origine degli eventi di Amazon SQS. Lambda continua a riprovare il messaggio fino a quando il timestamp del messaggio non supera il timeout di visibilità della coda e a quel punto Lambda elimina il messaggio.

## Implementazione di risposte batch parziali

Per impostazione predefinita, se la funzione rileva un errore durante l'elaborazione di un batch, tutti i messaggi in quel batch diventano nuovamente visibili nella coda, inclusi i messaggi che Lambda ha elaborato correttamente. Di conseguenza, la tua funzione potrebbe elaborare lo stesso messaggio più volte.

Per evitare di rielaborare tutti i messaggi correttamente elaborati in un batch con errori, puoi configurare lo strumento di mappatura dell'origine degli eventi in modo da rendere nuovamente visibili solo i messaggi con errori. Questa operazione è nota come risposta batch parziale. Per attivare le risposte batch parziali, specifica l'[FunctionResponseType](#) da eseguire durante la configurazione della `ReportBatchItemFailures` mappatura della sorgente dell'evento. Con questa impostazione, la funzione può restituire un esito parziale, il che può aiutare a ridurre il numero di tentativi non necessari sui record.

### Note

L'utilità [Batch Processor](#) di Powertools for AWS Lambda gestisce automaticamente tutta la logica di risposta parziale ai batch. Questa utilità semplifica l'implementazione dei modelli di elaborazione in batch e riduce il codice personalizzato necessario per gestire correttamente gli errori degli articoli in batch. È disponibile per Python, Java, Typescript e .NET.

Se `ReportBatchItemFailures` è attivata, Lambda non [riduce il polling dei messaggi](#) quando le invocazioni delle funzioni hanno esito negativo. Se prevedi che alcuni messaggi falliscano e non desideri che tali errori influiscano sulla velocità di elaborazione dei messaggi, utilizza `ReportBatchItemFailures`.

#### Note

Quando utilizzi le risposte batch, tieni presente quanto segue:

- Se la funzione restituisce un'eccezione, l'intero batch viene considerato completamente non riuscito.
- Se si utilizza questa caratteristica con una coda FIFO, la funzione dovrebbe interrompere l'elaborazione dei messaggi dopo il primo errore e restituire tutti i messaggi di errore e non elaborati in `batchItemFailures`. Ciò aiuta a preservare l'ordine dei messaggi nella coda.

### Attivazione di report batch parziali

1. Rivedi le [best practice per l'implementazione di risposte batch parziali](#).
2. Per attivare `ReportBatchItemFailures` per la tua funzione, esegui il comando riportato di seguito. Per recuperare l'UUID della mappatura dell'origine degli eventi, esegui il comando `list-event-source-mappings`. <https://docs.aws.amazon.com/cli/latest/reference/lambda/list-event-source-mappings.html> AWS CLI

```
aws lambda update-event-source-mapping \  
--uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \  
--function-response-types "ReportBatchItemFailures"
```

3. Aggiorna il codice della funzione per rilevare tutte le eccezioni e restituire i messaggi con errori in una risposta `batchItemFailures` in JSON. La risposta `batchItemFailures` deve includere un elenco di ID dei messaggi, come i valori `itemIdentifier` in JSON.

Ad esempio, supponiamo di avere un batch di cinque messaggi, con ID messaggio `id1`, `id2`, `id3`, `id4`, e `id5`. La tua funzione elabora correttamente `id1`, `id3`, e `id5`. Per rendere i messaggi `id2` e `id4` di nuovo visibili nella coda, la funzione dovrebbe restituire la seguente risposta:

```
{  
  "batchItemFailures": [  

```

```

    {
        "itemIdentifier": "id2"
    },
    {
        "itemIdentifier": "id4"
    }
]
}

```

Di seguito sono riportati alcuni esempi di codice di funzione che restituiscono l'elenco degli ID dei messaggi con errori nel batch:

## .NET

### SDK per .NET

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite .NET.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be
// converted into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJson
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
        ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
        List<SQSBatchResponse.BatchItemFailure>();
    }
}

```

```
foreach(var message in evnt.Records)
{
    try
    {
        //process your message
        await ProcessMessageAsync(message, context);
    }
    catch (System.Exception)
    {
        //Add failed message identifier to the batchItemFailures list
        batchItemFailures.Add(new
SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
    }
}
return new SQSBatchResponse(batchItemFailures);
}

private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
ILambdaContext context)
{
    if (String.IsNullOrEmpty(message.Body))
    {
        throw new Exception("No Body in SQS Message.");
    }
    context.Logger.LogInformation($"Processed message {message.Body}");
    // TODO: Do interesting work based on the new message
    await Task.CompletedTask;
}
}
```

Go

SDK per Go V2

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent)
    (map[string]interface{}, error) {
    batchItemFailures := []map[string]interface{}{}

    for _, message := range sqsEvent.Records {
        if len(message.Body) > 0 {
            // Your message processing condition here
            fmt.Printf("Successfully processed message: %s\n", message.Body)
        } else {
            // Message processing failed
            fmt.Printf("Failed to process message %s\n", message.MessageId)
            batchItemFailures = append(batchItemFailures, map[string]interface{}{
                "itemIdentifier": message.MessageId})
        }
    }

    sqsBatchResponse := map[string]interface{}{
        "batchItemFailures": batchItemFailures,
    }
    return sqsBatchResponse, nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Segnalazione di errori di elementi batch di SQS con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSBatchResponse;

import java.util.ArrayList;
import java.util.List;

public class ProcessSQSMessageBatch implements RequestHandler<SQSEvent,
    SQSBatchResponse> {
    @Override
    public SQSBatchResponse handleRequest(SQSEvent sqsEvent, Context context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
        ArrayList<SQSBatchResponse.BatchItemFailure>();

        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
            try {
                //process your message
            } catch (Exception e) {
                //Add failed message identifier to the batchItemFailures
                list
                batchItemFailures.add(new
                SQSBatchResponse.BatchItemFailure(message.getMessageId()));
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }
}
```



```
}
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Segnalazione dei guasti degli articoli del batch SQS con Lambda using. JavaScript

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
  const batchItemFailures = [];
  for (const record of event.Records) {
    try {
      await processMessageAsync(record, context);
    } catch (error) {
      batchItemFailures.push({ itemIdentifier: record.messageId });
    }
  }
  return { batchItemFailures };
};

async function processMessageAsync(record, context) {
  if (record.body && record.body.includes("error")) {
    throw new Error("There is an error in the SQS Message.");
  }
  console.log(`Processed message: ${record.body}`);
}
```

### Segnalazione degli errori relativi agli articoli del batch SQS con Lambda using. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure,
        SQSRecord } from 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
    Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
```

## PHP

### SDK per PHP

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php
```

```
use Bref\Context\Context;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();

        foreach ($records as $record) {
            try {
                // Assuming the SQS message is in JSON format
                $message = json_decode($record->getBody(), true);
                $this->logger->info(json_encode($message));
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $this->markAsFailed($record);
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords SQS
records");
    }
}

$logger = new StderrLogger();
```

```
return new Handler($logger);
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event, context):
    if event:
        batch_item_failures = []
        sqs_batch_response = {}

        for record in event["Records"]:
            try:
                print(f"Processed message: {record['body']}")
            except Exception as e:
                batch_item_failures.append({"itemIdentifier":
record['messageId']})

        sqs_batch_response["batchItemFailures"] = batch_item_failures
        return sqs_batch_response
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Segnalazione di errori di elementi batch di SQS con Lambda tramite Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'json'

def lambda_handler(event:, context:)
  if event
    batch_item_failures = []
    sqs_batch_response = {}

    event["Records"].each do |record|
      begin
        # process message
        rescue StandardError => e
          batch_item_failures << {"itemIdentifier" => record['messageId']}
        end
      end

      sqs_batch_response["batchItemFailures"] = batch_item_failures
      return sqs_batch_response
    end
  end
end
```

## Rust

### SDK per Rust

#### Note

C'è altro su GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) ->
    Result<SqsBatchResponse, Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
    })
}
```

```
#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(function_handler)).await
}
```

Se gli eventi non riusciti non ritornano in coda, vedi [Come si risolvono i problemi della funzione Lambda SQS? ReportBatchItemFailures](#) AWS nel Knowledge Center.

### Condizioni di successo e di errore

Lambda considera un batch completamente riuscito se la funzione restituisce uno dei seguenti elementi:

- Una `batchItemFailures` lista vuota
- Un `batchItemFailures` elenco nullo
- Un vuoto `EventResponse`
- Un valore nullo `EventResponse`

Lambda considera un batch completamente non riuscito se la funzione restituisce uno dei seguenti elementi:

- Risposta JSON non valida
- Una stringa vuota `itemIdentifier`
- Un valore nullo `itemIdentifier`
- Un `itemIdentifier` con un nome chiave errato
- Un `itemIdentifier` valore di con un ID messaggio inesistente

### CloudWatch metriche

Per determinare se la tua funzione sta segnalando correttamente gli errori degli articoli in batch, puoi monitorare le metriche `NumberOfMessagesDeleted` e quelle di `ApproximateAgeOfOldestMessage` Amazon SQS in Amazon. CloudWatch

- `NumberOfMessagesDeleted` tiene traccia del numero di messaggi rimossi dalla coda. Se questo scende a 0, significa che la risposta alla funzione non sta restituendo correttamente i messaggi non riusciti.

- `ApproximateAgeOfOldestMessage` tiene traccia del tempo di permanenza del messaggio meno recente all'interno della coda. Un incremento considerevole di questo parametro può indicare che la funzione non sta restituendo correttamente i messaggi non riusciti.

## Utilizzo di Powertools per AWS Lambda processore batch

L'utilità di elaborazione batch di Powertools per la gestione AWS Lambda automatica della logica di risposta parziale ai batch, riducendo la complessità dell'implementazione della segnalazione degli errori dei batch. Ecco alcuni esempi di utilizzo del processore batch:

### Python

#### Note

Per esempi completi e istruzioni di configurazione, consultate la documentazione del processore [batch](#).

Elaborazione di messaggi Amazon SQS con processore AWS Lambda batch.

```
import json
from aws_lambda_powertools import Logger
from aws_lambda_powertools.utilities.batch import BatchProcessor, EventType,
    process_partial_response
from aws_lambda_powertools.utilities.data_classes import SQSEvent
from aws_lambda_powertools.utilities.typing import LambdaContext

processor = BatchProcessor(event_type=EventType.SQS)
logger = Logger()

def record_handler(record):
    logger.info(record)
    # Your business logic here
    # Raise an exception to mark this record as failed

def lambda_handler(event, context: LambdaContext):
    return process_partial_response(
        event=event,
        record_handler=record_handler,
        processor=processor,
        context=context
```



)

## TypeScript

### Note

Per esempi completi e istruzioni di configurazione, consulta la documentazione [del processore batch](#).

Elaborazione di messaggi Amazon SQS con processore AWS Lambda batch.

```
import { BatchProcessor, EventType, processPartialResponse } from '@aws-lambda-powertools/batch';
import { Logger } from '@aws-lambda-powertools/logger';
import type { SQSEvent, Context } from 'aws-lambda';

const processor = new BatchProcessor(EventType.SQS);
const logger = new Logger();

const recordHandler = async (record: any): Promise<void> => {
  logger.info('Processing record', { record });
  // Your business logic here
  // Throw an error to mark this record as failed
};

export const handler = async (event: SQSEvent, context: Context) => {
  return processPartialResponse(event, recordHandler, processor, {
    context,
  });
};
```

## Parametri Lambda per gli strumenti di mappatura dell'origine degli eventi di Amazon SQS

Tutti i tipi di sorgenti di eventi Lambda condividono le stesse operazioni [CreateEventSourceMapping](#) e quelle dell'[UpdateEventSourceMapping](#) API. Tuttavia, solo alcuni dei parametri si applicano ad Amazon SQS.

| Parametro                      | Obbligatorio | Predefinito | Note                                                                                                                                                                                                                                         |
|--------------------------------|--------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BatchSize                      | N            | 10          | Per le code standard il massimo è 10.000. Per le code FIFO il massimo è 10.                                                                                                                                                                  |
| Abilitato                      | N            | true        | nessuno                                                                                                                                                                                                                                      |
| EventSourceArn                 | Y            | N/D         | L'ARN del flusso dei dati o di un consumatore di flusso                                                                                                                                                                                      |
| FunctionName                   | Y            | N/D         | nessuno                                                                                                                                                                                                                                      |
| FilterCriteria                 | N            | N/D         | <a href="#">Controllare gli eventi che Lambda invia alla funzione</a>                                                                                                                                                                        |
| FunctionResponseType           | N            | N/D         | Per consentire alla funzione di segnalare errori specifici in un batch, includi il valore ReportBatchItemFailures in FunctionResponseType. Per ulteriori informazioni, consulta <a href="#">Implementazione di risposte batch parziali</a> . |
| MaximumBatchingWindowInSeconds | N            | 0           | La creazione di finestre di batch non è supportata per le code FIFO                                                                                                                                                                          |

| Parametro               | Obbligatorio | Predefinito | Note                                                                                                                                                                                                                      |
|-------------------------|--------------|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ProvisionedPollerConfig | N            | N/D         | Configura il numero minimo (2-200) e massimo (2-2000) di poller di eventi dedicati per la mappatura delle sorgenti degli eventi SQS. Ogni poller può gestire fino all'1% del throughput e 10 richiami simultanei MB/sec . |
| ScalingConfig           | N            | N/D         | <a href="#">Configurazione della simultaneità massima per le origini eventi di Amazon SQS</a>                                                                                                                             |

## Utilizzo del filtro eventi con un'origine eventi Amazon SQS

Puoi utilizzare il filtraggio degli eventi per controllare quali record di un flusso o di una coda Lambda invia alla funzione. Per informazioni generali sul funzionamento del filtraggio eventi, consulta [the section called “Filtro eventi”](#).

Questa sezione si concentra sul filtraggio degli eventi per le sorgenti di eventi di Amazon SQS.

### Note

Le mappature delle sorgenti di eventi di Amazon SQS supportano solo il filtraggio sulla chiave. body

## Argomenti

- [Nozioni di base sul filtraggio degli eventi di Amazon SQS](#)

## Nozioni di base sul filtraggio degli eventi di Amazon SQS

Supponiamo che la coda Amazon SQS contenga messaggi nel formato JSON seguente.

```
{
  "RecordNumber": 1234,
  "TimeStamp": "yyyy-mm-ddThh:mm:ss",
  "RequestCode": "AAAA"
}
```

Un record di esempio per questa coda sarebbe il seguente.

```
{
  "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
  "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
  "body": "{\n  \"RecordNumber\": 1234,\n  \"TimeStamp\": \"yyyy-mm-ddThh:mm:ss\",\n  \"RequestCode\": \"AAAA\"\n}",
  "attributes": {
    "ApproximateReceiveCount": "1",
    "SentTimestamp": "1545082649183",
    "SenderId": "AIDAIENQZJOL023YVJ4V0",
    "ApproximateFirstReceiveTimestamp": "1545082649185"
  },
  "messageAttributes": {},
  "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
  "eventSource": "aws:sqs",
  "eventSourceARN": "arn:aws:sqs:us-west-2:123456789012:my-queue",
  "awsRegion": "us-west-2"
}
```

Per filtrare in base al contenuto dei messaggi Amazon SQS, utilizza la chiave `body` nel record dei messaggi Amazon SQS. Supponiamo di voler elaborare solo i record `RequestCode` in cui il messaggio Amazon SQS contiene «BBBB». L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"body\" : { \"RequestCode\" : [ \"BBBB\" ] } }"
    }
  ]
}
```

Per una maggiore chiarezza, ecco il valore del Pattern del filtro espanso in JSON semplice.

```
{
  "body": {
    "RequestCode": [ "BBBB" ]
  }
}
```

Puoi aggiungere il filtro utilizzando la console AWS CLI o un modello. AWS SAM

### Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in [Collegamento dei criteri di filtro a una mappatura dell'origine evento \(console\)](#) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "body" : { "RequestCode" : [ "BBBB" ] } }
```

### AWS CLI

Per creare una nuova mappatura della sorgente di eventi con questi criteri di filtro utilizzando AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RequestCode\" : [ \"BBBB\" ] } }"]}]'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RequestCode\" : [ \"BBBB\" ] } }"]}]'
```

### AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungete il seguente frammento al modello YAML per la fonte dell'evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "body" : { "RequestCode" : [ "BBBB" ] } }'
```

Supponiamo che tu voglia che la tua funzione elabori solo i record in cui RecordNumber è maggiore di 9999. L'oggetto FilterCriteria dovrebbe avere la struttura seguente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"body\" : { \"RecordNumber\" : [ { \"numeric\": [ \">\",
9999 ] } ] } }"
    }
  ]
}
```

Per una maggiore chiarezza, ecco il valore del Pattern del filtro espanso in JSON semplice.

```
{
  "body": {
    "RecordNumber": [
      {
        "numeric": [ ">", 9999 ]
      }
    ]
  }
}
```

Puoi aggiungere il filtro usando la console o un modello. AWS CLI AWS SAM

## Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in [Collegamento dei criteri di filtro a una mappatura dell'origine evento \(console\)](#) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }
```

## AWS CLI

Per creare una nuova mappatura della sorgente di eventi con questi criteri di filtro utilizzando AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
aws lambda create-event-source-mapping \  
  --function-name my-function \  
  --event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue \  
  --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RecordNumber\" :  
  [ { \"numeric\" : [ \">\", 9999 ] } ] } ] }"]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \  
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \  
  --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RecordNumber\" :  
  [ { \"numeric\" : [ \">\", 9999 ] } ] } ] }"]}'
```

## AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungete il seguente frammento al modello YAML per la fonte dell'evento.

```
FilterCriteria:  
  Filters:  
    - Pattern: '{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }'
```

Per Amazon SQS, il corpo del messaggio può essere qualsiasi stringa. Tuttavia, questo può essere problematico se il `FilterCriteria` si aspetta che `body` sia in un formato JSON valido. Anche lo scenario inverso è vero: se il corpo del messaggio in arrivo è in formato JSON ma i criteri di filtraggio si aspettano che `body` sia una stringa semplice, allora si potrebbe riscontrare un comportamento indesiderato.

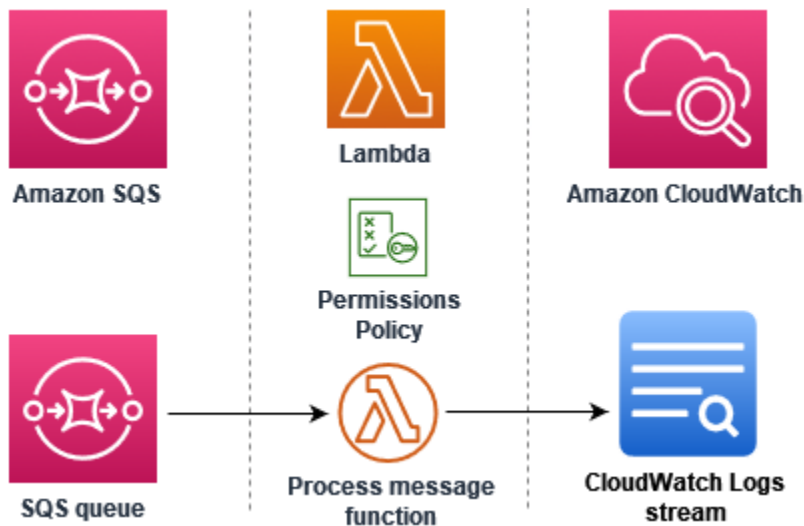
Per evitare questo problema, assicurati che il formato del corpo nei `FilterCriteria` corrisponda al formato previsto per il `body` nei messaggi ricevuti dalla coda. Prima di filtrare i messaggi, Lambda valuta automaticamente il formato del corpo del messaggio in arrivo e il modello di filtraggio per `body`. In caso di mancata corrispondenza, Lambda rilascia il messaggio. La tabella seguente riepiloga questa valutazione:

| Formato <b>body</b> messaggio in arrivo | Formato <b>body</b> modello di filtro              | Operazione risultante                                                                 |
|-----------------------------------------|----------------------------------------------------|---------------------------------------------------------------------------------------|
| Stringa normale                         | Stringa normale                                    | Filtri Lambda in base ai criteri di filtro.                                           |
| Stringa normale                         | Nessun modello di filtro per le proprietà dei dati | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro. |
| Stringa normale                         | JSON valido                                        | Lambda rilascia il messaggio.                                                         |
| JSON valido                             | Stringa normale                                    | Lambda rilascia il messaggio.                                                         |
| JSON valido                             | Nessun modello di filtro per le proprietà dei dati | Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro. |
| JSON valido                             | JSON valido                                        | Filtri Lambda in base ai criteri di filtro.                                           |

## Tutorial: Utilizzo di Lambda con Amazon SQS

In questo tutorial creerai una funzione Lambda che utilizza messaggi da una coda di [Amazon Simple Queue Service \(Amazon SQS\)](#). La funzione Lambda viene eseguita ogni volta che viene aggiunto un nuovo messaggio alla coda. La funzione scrive i messaggi in un flusso Amazon CloudWatch Logs. Il seguente diagramma illustra le risorse AWS utilizzate per completare il tutorial.





Per completare questo tutorial, completa le seguenti attività:

1. Crea una funzione Lambda che scrive messaggi nei log. CloudWatch
2. Creare una coda Amazon SQS.
3. Crea una mappatura dell'origine degli eventi Lambda. La mappatura dell'origine degli eventi legge la coda di Amazon SQS e richiama la funzione Lambda quando viene aggiunto un nuovo messaggio.
4. Verifica la configurazione aggiungendo messaggi alla coda e monitorando i risultati nei log. CloudWatch

## Prerequisiti

### Installa il AWS Command Line Interface

Se non hai ancora installato il AWS Command Line Interface, segui i passaggi in [Installazione o aggiornamento della versione più recente AWS CLI](#) per installarlo.

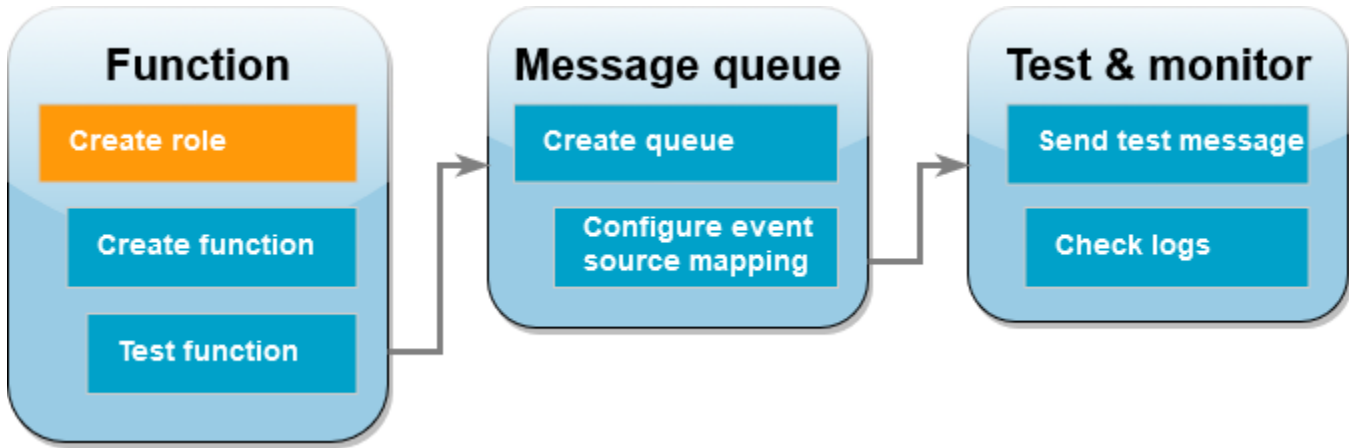
Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

#### Note

Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere

una Windows-integrated versione di Ubuntu e Bash, [installa il sottosistema Windows per Linux](#).

### Creazione del ruolo di esecuzione



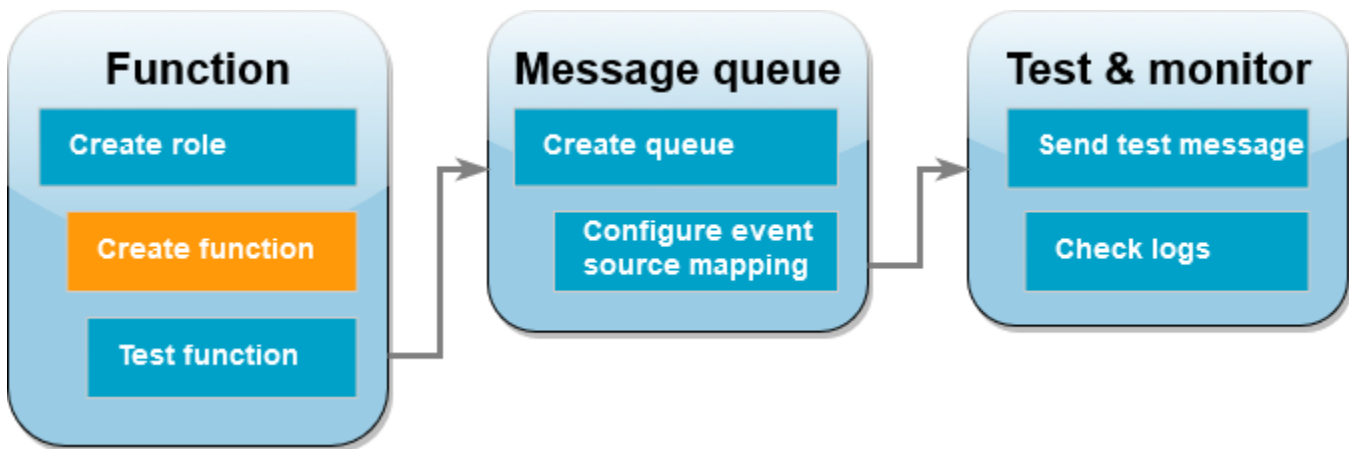
Un ruolo di [esecuzione](#) è un ruolo AWS Identity and Access Management (IAM) che concede a una funzione Lambda l'autorizzazione all'accesso Servizi AWS e alle risorse. Per consentire alla tua funzione di leggere gli elementi da Amazon SQS, allega la politica delle `AWSLambdaSQSQueueExecutionRole` autorizzazioni.

Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione Amazon SQS personalizzata

1. Aprire la [pagina Roles \(Ruoli\)](#) della console IAM.
2. Scegli Crea ruolo.
3. Per Tipo di entità attendibile, scegli Servizio AWS .
4. In Caso d'uso, scegli Lambda.
5. Scegli Next (Successivo).
6. Nella casella di ricerca Policy di autorizzazione, inserisci **AWSLambdaSQSQueueExecutionRole**.
7. Seleziona la `AWSLambdaSQSQueueExecutionRole` policy, quindi scegli Avanti.
8. In Dettagli del ruolo, per Nome del ruolo inserisci **lambda-sqs-role**, quindi scegli Crea ruolo.

Dopo la creazione del ruolo, prendi nota del valore del nome della risorsa Amazon (ARN) del ruolo di esecuzione. Ne avrai bisogno nelle fasi successive.

## Creazione della funzione



Crea una funzione Lambda che elabora i messaggi Amazon SQS. Il codice della funzione registra il corpo del messaggio Amazon SQS in Logs. CloudWatch

Questo tutorial utilizza il runtime Node.js 24, ma abbiamo anche fornito codice di esempio in altri linguaggi di runtime. Per visualizzare il codice per il runtime che ti interessa, seleziona la scheda corrispondente nella casella seguente. Il JavaScript codice che utilizzerai in questo passaggio si trova nel primo esempio mostrato nella `JavaScript` scheda.

.NET

SDK per .NET

### Note

C'è altro su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
into a .NET class.
```

```
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
            Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

## Go

### SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
    for _, record := range event.Records {
        err := processMessage(record)
        if err != nil {
            return err
        }
    }
    fmt.Println("done")
    return nil
}

func processMessage(record events.SQSMessage) error {
    fmt.Printf("Processed message %s\n", record.Body)
    // TODO: Do interesting work based on the new message
    return nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class Function implements RequestHandler<SQSEvent, Void> {
    @Override
    public Void handleRequest(SQSEvent sqsEvent, Context context) {
        for (SQSMessage msg : sqsEvent.getRecords()) {
            processMessage(msg, context);
        }
        context.getLogger().log("done");
        return null;
    }

    private void processMessage(SQSMessage msg, Context context) {
        try {
            context.getLogger().log("Processed message " + msg.getBody());

            // TODO: Do interesting work based on the new message

        } catch (Exception e) {
            context.getLogger().log("An error occurred");
            throw e;
        }
    }
}
```

```
}  
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Utilizzo di un evento SQS utilizzando Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
exports.handler = async (event, context) => {  
  for (const message of event.Records) {  
    await processMessageAsync(message);  
  }  
  console.info("done");  
};  
  
async function processMessageAsync(message) {  
  try {  
    console.log(`Processed message ${message.body}`);  
    // TODO: Do interesting work based on the new message  
    await Promise.resolve(1); //Placeholder for actual async work  
  } catch (err) {  
    console.error("An error occurred");  
    throw err;  
  }  
}
```

### Utilizzo di un evento SQS con Lambda utilizzando. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";
```

```
export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

## PHP

### SDK per PHP

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
```



```
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for message in event['Records']:
        process_message(message)
    print("done")

def process_message(message):
    try:
        print(f"Processed message {message['body']}")
        # TODO: Do interesting work based on the new message
    except Exception as err:
        print("An error occurred")
        raise err
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
    event['Records'].each do |message|
        process_message(message)
    end
    puts "done"
end

def process_message(message)
    begin
        puts "Processed message #{message['body']}"
        # TODO: Do interesting work based on the new message
    end
```

```

    rescue StandardError => err
      puts "An error occurred"
      raise err
    end
  end
end

```

## Rust

### SDK per Rust

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite Rust.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
        tracing::info!("Message body: {}",
            record.body.as_deref().unwrap_or_default())
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
        time.
        .without_time()
}

```

```
        .init();

    run(service_fn(function_handler)).await
}
```

Per creare una funzione Node.js Lambda

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir sqs-tutorial
cd sqs-tutorial
```

2. Copia il JavaScript codice di esempio in un nuovo file denominato `index.js`.
3. Crea un pacchetto di implementazione utilizzando il seguente comando `zip`.

```
zip function.zip index.js
```

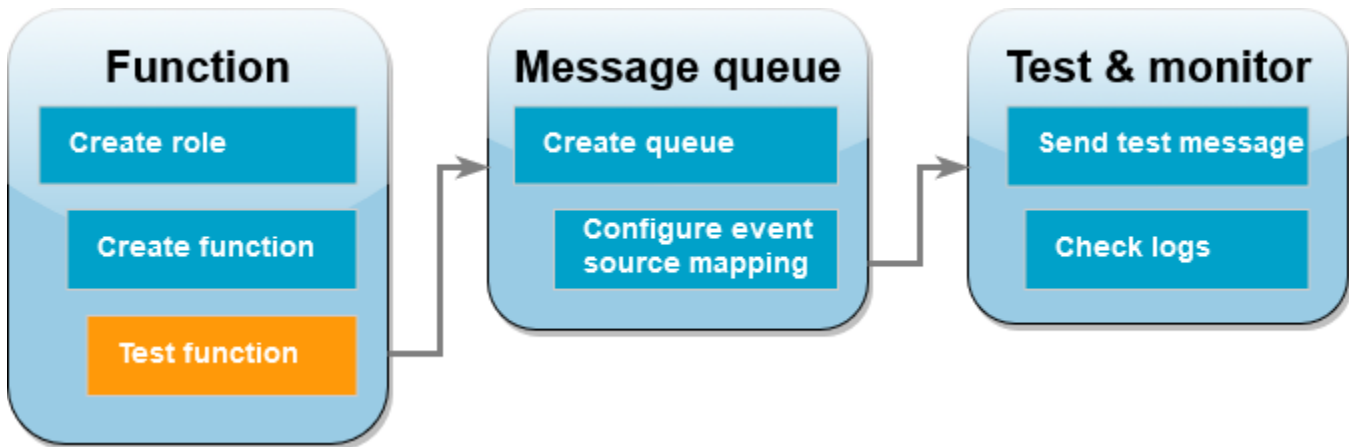
4. Crea una funzione Lambda utilizzando il comando [create-function](#) della AWS CLI . Per il `role` parametro, immettere l'ARN del ruolo di esecuzione creato in precedenza.

 Note

La funzione Lambda e la coda Amazon SQS devono trovarsi nella stessa Regione AWS.

```
aws lambda create-function --function-name ProcessSQSRecord \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs24.x \
--role arn:aws:iam::111122223333:role/lambda-sqs-role
```

## Test della funzione



Richiama manualmente la tua funzione Lambda utilizzando il `invoke` AWS CLI comando e un evento Amazon SQS di esempio.

Invocazione della funzione Lambda con un evento di esempio

1. Salva il seguente JSON come un file denominato `input.json`. Questo JSON simula un evento che Amazon SQS potrebbe inviare alla tua funzione Lambda, dove `"body"` contiene il messaggio effettivo dalla coda. In questo esempio, il messaggio è `"test"`.

### Example Evento Amazon SQS

Questo è un evento di test: non è necessario modificare il messaggio o il numero di account.

```
{
  "Records": [
    {
      "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
      "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgXlaS3SLy0a...",
      "body": "test",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
        "SenderId": "AIDAIENQZJOL023YVJ4V0",
        "ApproximateFirstReceiveTimestamp": "1545082649185"
      },
      "messageAttributes": {},
      "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
      "eventSource": "aws:sqs",
      "eventSourceARN": "arn:aws:sqs:us-east-1:111122223333:my-queue",
    }
  ]
}
```

```
    "awsRegion": "us-east-1"  
  }  
]  
}
```

2. Esegui il seguente [comando invoke](#). AWS CLI Questo comando restituisce CloudWatch i log della risposta. Per ulteriori informazioni sul recupero di oggetti, consulta [Registri di accesso con AWS CLI](#).

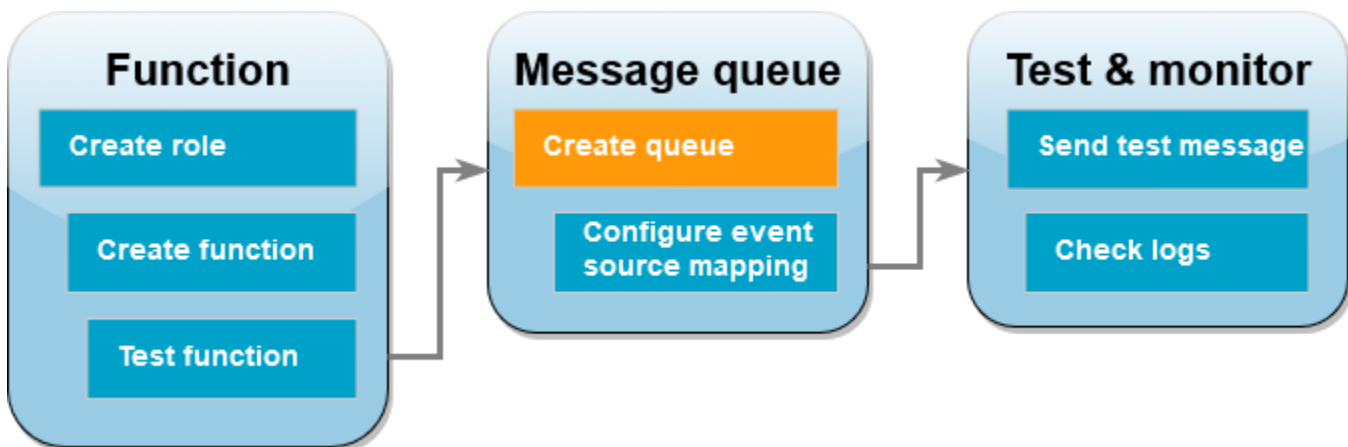
```
aws lambda invoke --function-name ProcessSQSRecord --payload file://input.json out  
--log-type Tail \  
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --  
decode
```

L'cli-binary-format opzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

3. Individua il log INFO nella risposta. È qui che la funzione Lambda registra il corpo del messaggio. I log visualizzati dovrebbero essere di questo tipo:

```
2023-09-11T22:45:04.271Z 348529ce-2211-4222-9099-59d07d837b60 INFO Processed  
message test  
2023-09-11T22:45:04.288Z 348529ce-2211-4222-9099-59d07d837b60 INFO done
```

### Creazione di una coda Amazon SQS



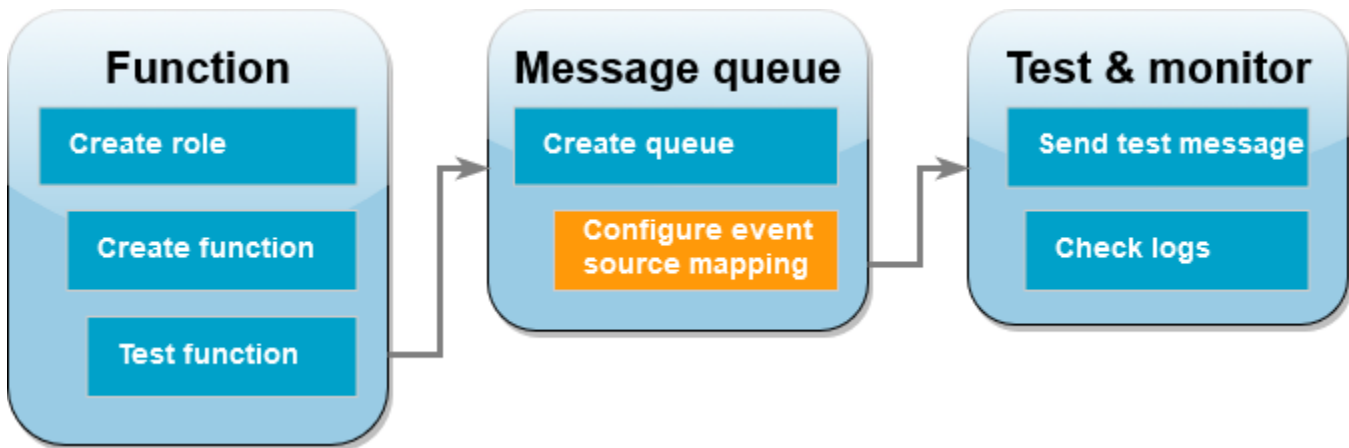
Creare una coda Amazon SQS che la funzione Lambda può utilizzare come origine eventi. La funzione Lambda e la coda Amazon SQS devono trovarsi nella stessa Regione AWS.

Come creare una coda

1. Apri la [console Amazon SQS](#).
2. Scegliere Crea coda.
3. Inserisci un nome per la coda. Lascia tutte le altre proprietà sui valori predefiniti.
4. Scegliere Crea coda.

Dopo aver creato la coda, prendi nota del suo ARN. Questa operazione è necessaria nella fase successiva quando si associa la coda alla funzione Lambda.

Configurazione dell'origine eventi



Collega la coda Amazon SQS alla tua funzione Lambda creando una [mappatura dell'origine degli eventi](#). La mappatura dell'origine degli eventi legge la coda di Amazon SQS e richiama la funzione Lambda quando viene aggiunto un nuovo messaggio.

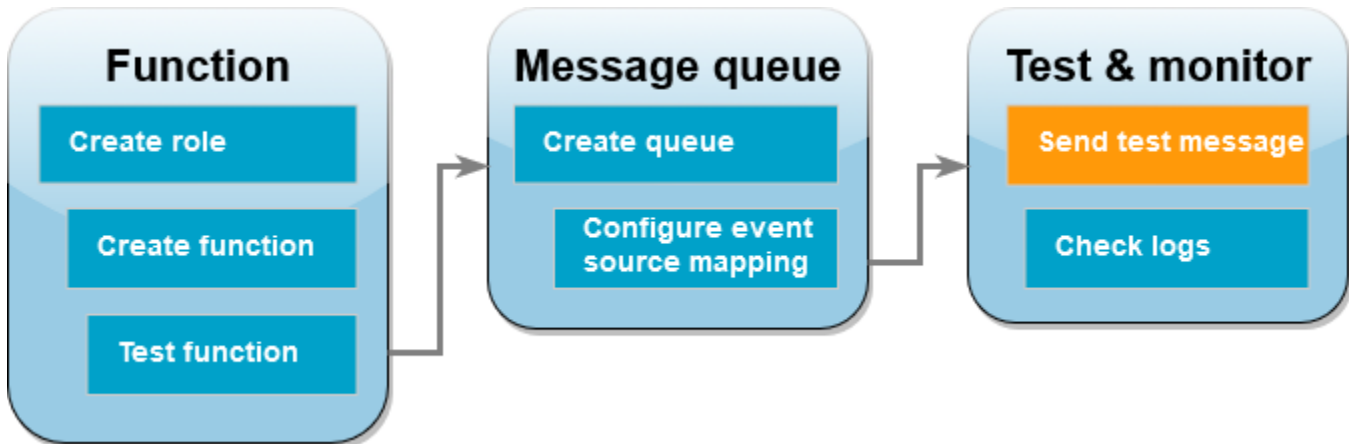
Per creare una mappatura tra la coda Amazon SQS e la funzione Lambda, usa il comando `create-event-source-mapping`. <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/create-event-source-mapping.html> AWS CLI Esempio:

```
aws lambda create-event-source-mapping --function-name ProcessSQSRecord --batch-size 10 \
--event-source-arn arn:aws:sqs:us-east-1:111122223333:my-queue
```

Per ottenere un elenco delle mappature delle origini degli eventi, usa il comando [list-event-source-mappings](#). Esempio:

```
aws lambda list-event-source-mappings --function-name ProcessSQSRecord
```

Invio di un messaggio di test



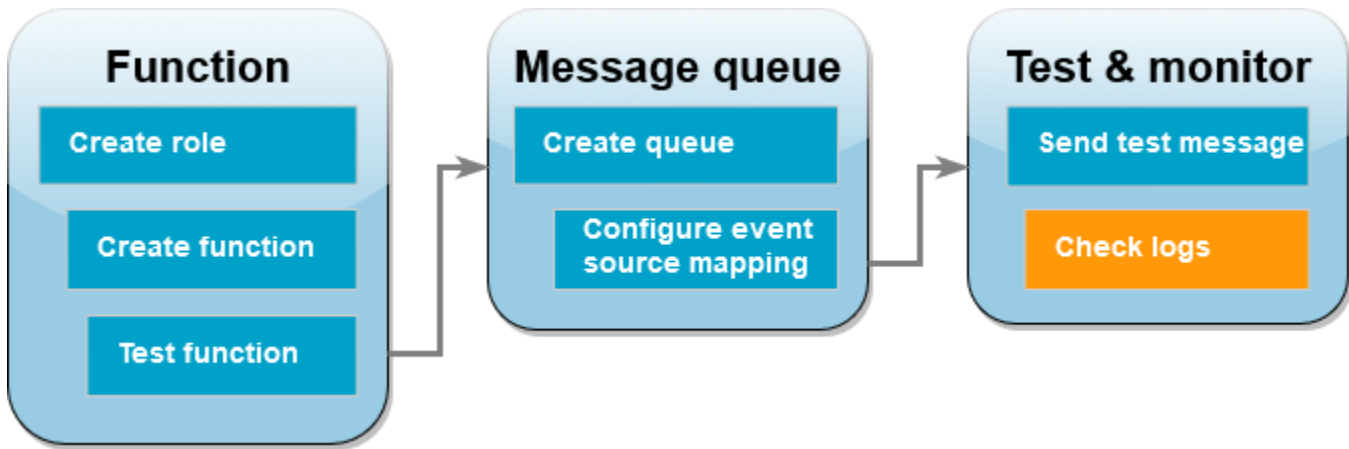
Invio di un messaggio Amazon SQS alla funzione Lambda

1. Apri la [console Amazon SQS](#).
2. Scegli la coda creata in precedenza.
3. Scegli **Invia e ricevi messaggi**.
4. Nella sezione **Corpo del messaggio**, inserisci un messaggio di test, ad esempio "questo è un messaggio di prova".
5. Scegliere **Invia messaggio**.

Lambda esegue il polling della coda per gli aggiornamenti. Quando c'è un nuovo messaggio, Lambda richiama la tua funzione con questi nuovi dati di evento dalla coda. Se il gestore della funzione conclude senza eccezioni, Lambda considera il messaggio elaborato correttamente e inizia a leggere nuovi messaggi nella coda. Dopo aver elaborato correttamente un messaggio, Lambda lo elimina automaticamente dalla coda. Se il gestore genera un'eccezione, Lambda considera il batch dei messaggi come non correttamente elaborato e Lambda richiama la funzione con lo stesso batch di messaggi.



## Controlla i log CloudWatch



Verifica della corretta elaborazione del messaggio da parte della funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione ProcessSQSRecord.
3. Scegli Monitor (Monitoraggio).
4. Scegli Visualizza i log CloudWatch .
5. Nella CloudWatch console, scegli il Log stream per la funzione.
6. Individua il log INFO. È qui che la funzione Lambda registra il corpo del messaggio. Dovresti vedere il messaggio che hai inviato dalla coda Amazon SQS. Esempio:

```
2023-09-11T22:49:12.730Z b0c41e9c-0556-5a8b-af83-43e59efeec71 INFO Processed  
message this is a test message.
```

## Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti inutili addebiti al tuo Account AWS.

### Come eliminare il ruolo di esecuzione

1. Aprire la pagina [Ruoli](#) della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

## Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **confirm** nel campo di immissione testo e scegli Delete (Elimina).

## Per eliminare la coda Amazon SQS

1. Accedi Console di gestione AWS e apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Selezionare la coda creata.
3. Scegli Elimina.
4. Inserisci **confirm** nel campo di immissione del testo.
5. Scegli Delete (Elimina).

## Tutorial: utilizzo di una coda Amazon SQS tra più account come origine eventi

In questo tutorial, crei una funzione Lambda che consuma i messaggi da una coda Amazon Simple Queue Service (Amazon SQS) in un altro account. AWS Questo tutorial coinvolge due AWS account: l'account A si riferisce all'account che contiene la funzione Lambda e l'account B si riferisce all'account che contiene la coda Amazon SQS.

### Prerequisiti

#### Installa il AWS Command Line Interface

Se non hai ancora installato il AWS Command Line Interface, segui i passaggi in [Installazione o aggiornamento della versione più recente AWS CLI](#) per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

#### Note

Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, zip) non sono supportati dai terminali integrati del sistema operativo. Per ottenere

una Windows-integrated versione di Ubuntu e Bash, [installa il sottosistema Windows per Linux](#).

## Creazione del ruolo di esecuzione (account A)

Nell'Account A, crea un ruolo di [esecuzione](#) che autorizzi la tua funzione ad accedere alle risorse richieste. AWS

### Creazione di un ruolo di esecuzione

1. Apri la pagina [Ruoli](#) nella console AWS Identity and Access Management (IAM).
2. Scegliere Create role (Crea ruolo).
3. Creare un ruolo con le seguenti proprietà.
  - Entità attendibile – AWS Lambda
  - Autorizzazioni: AWSLambdaSQSQueueExecutionRole
  - Nome ruolo – **cross-account-lambda-sqs-role**

La AWSLambdaSQSQueueExecutionRole policy dispone delle autorizzazioni necessarie alla funzione per leggere gli elementi da Amazon SQS e scrivere i log su Amazon CloudWatch Logs.

## Creazione della funzione (account A)

In Account A, crea una funzione Lambda che elabora i messaggi Amazon SQS. La funzione Lambda e la coda Amazon SQS devono trovarsi nella stessa Regione AWS.

Il seguente esempio di Node.js codice scrive ogni messaggio in un log in CloudWatch Logs.

### Example index.mjs

```
export const handler = async function(event, context) {
  event.Records.forEach(record => {
    const { body } = record;
    console.log(body);
  });
  return {};
}
```

## Creazione della funzione

### Note

Seguendo questi passaggi viene creata una Node.js funzione. Per le altre lingue i passaggi sono simili, ma alcuni dettagli sono diversi.

1. Salva l'esempio di codice come un file denominato `index.mjs`.
2. Crea un pacchetto di implementazione.

```
zip function.zip index.mjs
```

3. Crea la funzione utilizzando il comando `create-function` AWS Command Line Interface (AWS CLI). Sostituisci `arn:aws:iam::111122223333:role/cross-account-lambda-sqs-role` con l'ARN del ruolo di esecuzione che hai creato in precedenza.

```
aws lambda create-function --function-name CrossAccountSQSExample \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs24.x \
--role arn:aws:iam::111122223333:role/cross-account-lambda-sqs-role
```

## Testa la funzione (Account A)

Nell'Account A, testa manualmente la tua funzione Lambda utilizzando il `invoke` AWS CLI comando e un evento Amazon SQS di esempio.

Se il gestore termina normalmente senza eccezioni, Lambda considera il messaggio come elaborato correttamente e inizia a leggere nuovi messaggi nella coda. Dopo aver elaborato correttamente un messaggio, Lambda lo elimina automaticamente dalla coda. Se il gestore genera un'eccezione, Lambda considera il batch dei messaggi come non correttamente elaborato e Lambda richiama la funzione con lo stesso batch di messaggi.

1. Salva il seguente JSON come un file denominato `input.txt`.

```
{
  "Records": [
    {
      "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
      "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgXlaS3SLy0a..."
    }
  ]
}
```

```
{
  "body": "test",
  "attributes": {
    "ApproximateReceiveCount": "1",
    "SentTimestamp": "1545082649183",
    "SenderId": "AIDAIENQZJOL023YVJ4V0",
    "ApproximateFirstReceiveTimestamp": "1545082649185"
  },
  "messageAttributes": {},
  "md5ofBody": "098f6bcd4621d373cade4e832627b4f6",
  "eventSource": "aws:sqs",
  "eventSourceARN": "arn:aws:sqs:us-east-1:111122223333:example-queue",
  "awsRegion": "us-east-1"
}
```

Il precedente JSON simula un evento che Amazon SQS potrebbe inviare alla tua funzione Lambda, dove "body" contiene il messaggio effettivo dalla coda.

2. Eseguire il seguente comando `invoke` AWS CLI .

```
aws lambda invoke --function-name CrossAccountSQSExample \
  --cli-binary-format raw-in-base64-out \
  --payload file://input.txt outputfile.txt
```

L'`cli-binary-format` opzione è obbligatoria se utilizzi la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

3. Verificare l'output nel file `outputfile.txt`.

## Creazione di una coda Amazon SQS (account B)

In Account B, crea una coda Amazon SQS che la funzione Lambda in Account A può utilizzare come origine eventi. La funzione Lambda e la coda Amazon SQS devono trovarsi nella stessa Regione AWS.

### Come creare una coda

1. Apri la [console Amazon SQS](#).

2. Scegliere Crea coda.

3. Crea una coda con le seguenti proprietà.

- Type (Tipo): Standard
- Nome: LambdaCrossAccountQueue
- Configuration (Configurazione): mantieni le impostazioni predefinite.
- Access policy (Policy di accesso): scegli Advanced (Avanzata). Incolla la seguente politica JSON. Sostituisci i valori seguenti:
  - 111122223333: Account AWS ID per l'account A
  - 444455556666: Account AWS ID per l'account B

JSON

```
{
  "Version": "2012-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement": [
    {
      "Sid": "Queue1_AllActions",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:role/cross-account-lambda-
sqs-role"
        ]
      },
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:us-
east-1:444455556666:LambdaCrossAccountQueue"
    }
  ]
}
```

Questa policy concede al ruolo di esecuzione Lambda nell'Account A le autorizzazioni per utilizzare i messaggi provenienti da questa coda Amazon SQS.

4. Dopo aver creato la coda, registra il suo Amazon Resource Name (ARN). Questa operazione è necessaria nella fase successiva quando si associa la coda alla funzione Lambda.

## Configurazione dell'origine eventi (account A)

Nell'Account A, crea una mappatura dell'origine degli eventi tra la coda Amazon SQS nell'Account B e la tua funzione Lambda eseguendo il seguente comando. `create-event-source-mapping` AWS CLI Sostituisci `arn:aws:sqs:us-east-1:444455556666:LambdaCrossAccountQueue` con l'ARN della coda Amazon SQS che hai creato nel passaggio precedente.

```
aws lambda create-event-source-mapping --function-name CrossAccountSQSExample --batch-size 10 \
--event-source-arn arn:aws:sqs:us-east-1:444455556666:LambdaCrossAccountQueue
```

È possibile ottenere un elenco delle mappature delle fonti eventi eseguendo il comando riportato di seguito.

```
aws lambda list-event-source-mappings --function-name CrossAccountSQSExample \
--event-source-arn arn:aws:sqs:us-east-1:444455556666:LambdaCrossAccountQueue
```

## Eseguire il test della configurazione

A questo punto è possibile eseguire il test della configurazione come indicato di seguito:

1. Nell'Account B, apri [Console Amazon SQS](#).
2. Scegli quello che `LambdaCrossAccountQueue` hai creato in precedenza.
3. Scegli **Invia e ricevi messaggi**.
4. In **Corpo del messaggio**, inserisci un messaggio di prova.
5. Scegliere **Send Message** (**Invia messaggio**).

La funzione Lambda nell'Account A dovrebbe ricevere il messaggio. Lambda continua a interrogare la coda per gli aggiornamenti. Quando c'è un nuovo messaggio, Lambda richiama la tua funzione con questi nuovi dati di evento dalla coda. La tua funzione viene eseguita e crea log in Amazon. CloudWatch Puoi visualizzare i log nella [console CloudWatch](#).

## Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili al tuo Account AWS.

Nell'Account A elimina il ruolo di esecuzione e la funzione Lambda.

## Come eliminare il ruolo di esecuzione

1. Aprire la pagina [Ruoli](#) della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

## Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **confirm** nel campo di immissione testo e scegli Delete (Elimina).

## Nell'Account B elimina la coda Amazon SQS.

### Per eliminare la coda Amazon SQS

1. Accedi Console di gestione AWS e apri la console Amazon SQS all'indirizzo. <https://console.aws.amazon.com/sqs/>
2. Selezionare la coda creata.
3. Scegli Elimina.
4. Inserisci **confirm** nel campo di immissione del testo.
5. Scegli Delete (Elimina).



## Orchestrazione delle funzioni Lambda con Step Functions

AWS Step Functions fornisce un'orchestrazione visiva del flusso di lavoro per coordinare le funzioni Lambda con altri servizi. AWS Con integrazioni native con oltre 220 AWS servizi e un'infrastruttura completamente gestita e senza manutenzione, Step Functions è ideale quando hai bisogno di una progettazione visiva del flusso di lavoro e integrazioni di servizi completamente gestite.

Per l'orchestrazione utilizzando linguaggi di programmazione standard all'interno di Lambda in cui la logica del flusso di lavoro convive con la logica aziendale, prendi in considerazione le funzioni durevoli di Lambda. ??? Per assistenza nella scelta tra queste opzioni, consulta [Durable functions o Step Functions](#).

Ad esempio, l'elaborazione di un ordine potrebbe richiedere la convalida dei dettagli dell'ordine, il controllo dei livelli di inventario, l'elaborazione del pagamento e la generazione di una fattura. Scrivi funzioni Lambda separate per ogni attività e usa Step Functions per gestire il flusso di lavoro. Step Functions coordina il flusso di dati tra le tue funzioni e gestisce gli errori in ogni fase. Questa separazione semplifica la visualizzazione, la modifica e la manutenzione dei flussi di lavoro man mano che diventano più complessi.

### Quando usare Step Functions con Lambda

I seguenti scenari sono buoni esempi di quando Step Functions è particolarmente adatto per orchestrare Lambda-based le applicazioni.

- [Elaborazione sequenziale](#)
- [Gestione complessa degli errori](#)
- [Flussi di lavoro condizionali e approvazioni umane](#)
- [Elaborazione parallela](#)

#### Elaborazione sequenziale

L'elaborazione sequenziale è quando un'attività deve essere completata prima di poter iniziare l'attività successiva. Ad esempio, in un sistema di elaborazione degli ordini, l'elaborazione dei pagamenti non può iniziare fino al completamento della convalida dell'ordine e la generazione della fattura deve attendere la conferma del pagamento. Scrivi funzioni Lambda separate per ogni attività e usa Step Functions per gestire la sequenza e gestire il flusso di dati tra le funzioni.

## Anti-pattern esempio

Una singola funzione Lambda gestisce l'intero flusso di lavoro di elaborazione degli ordini tramite:

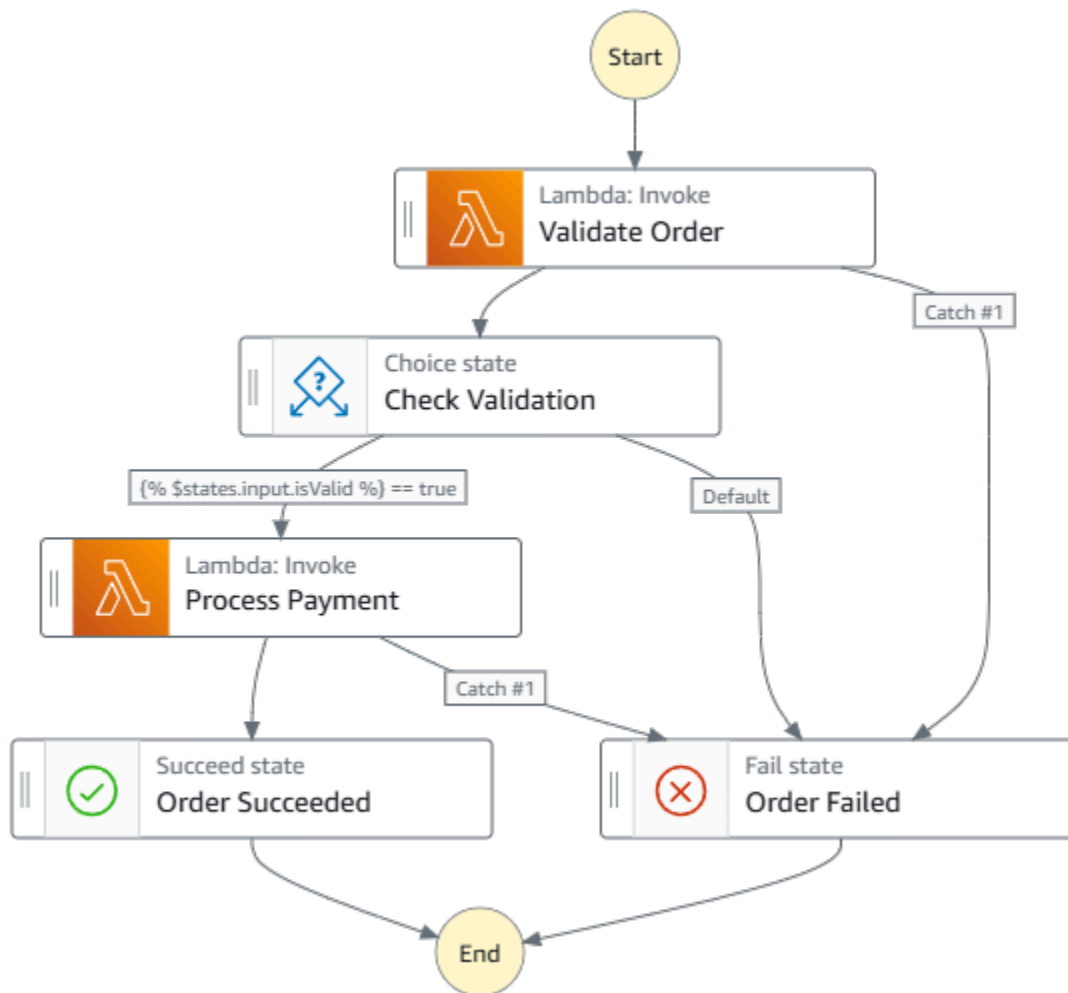
- Richiamare altre funzioni Lambda in sequenza
- Analisi e convalida delle risposte di ciascuna funzione
- Implementazione della logica di gestione e ripristino degli errori
- Gestione del flusso di dati tra le funzioni

## Approccio consigliato

Utilizza due funzioni Lambda: una per convalidare l'ordine e una per elaborare il pagamento. Step Functions coordina queste funzioni mediante:

- Esecuzione delle attività nella sequenza corretta
- Passaggio di dati tra funzioni
- Implementazione della gestione degli errori in ogni fase
- Utilizzo [degli stati](#) Choice per garantire che solo gli ordini validi procedano al pagamento

## Example grafico del flusso di lavoro



### Note

Code-first alternativa: per l'elaborazione sequenziale con checkpoint e riprova basati su codice, vedi i passaggi di Lambda durable functions. ???

## Gestione complessa degli errori

Mentre Lambda fornisce funzionalità di [ripetizione per chiamate asincrone e mappature delle sorgenti degli eventi](#), Step Functions offre una gestione degli errori più sofisticata per flussi di lavoro complessi. È possibile [configurare i tentativi automatici](#) con backoff esponenziale e impostare politiche di ripetizione diverse per diversi tipi di errori. Quando i tentativi sono esauriti, utilizzalo per indirizzare gli errori `Catch` a uno stato di fallback. <https://docs.aws.amazon.com/step-functions/latest/>

[dg/concepts-error-handling.html#error-handling-fallback-states](https://aws.amazon.com/lambda/dg/concepts-error-handling.html#error-handling-fallback-states) Ciò è particolarmente utile quando è necessaria una gestione degli errori a livello di flusso di lavoro che coordini più funzioni e servizi.

Per saperne di più sulla gestione degli errori delle funzioni Lambda in una macchina a stati, vedi [Gestione degli errori](#) in The Workshop. AWS Step Functions

### Anti-pattern esempio

Una singola funzione Lambda gestisce tutto quanto segue:

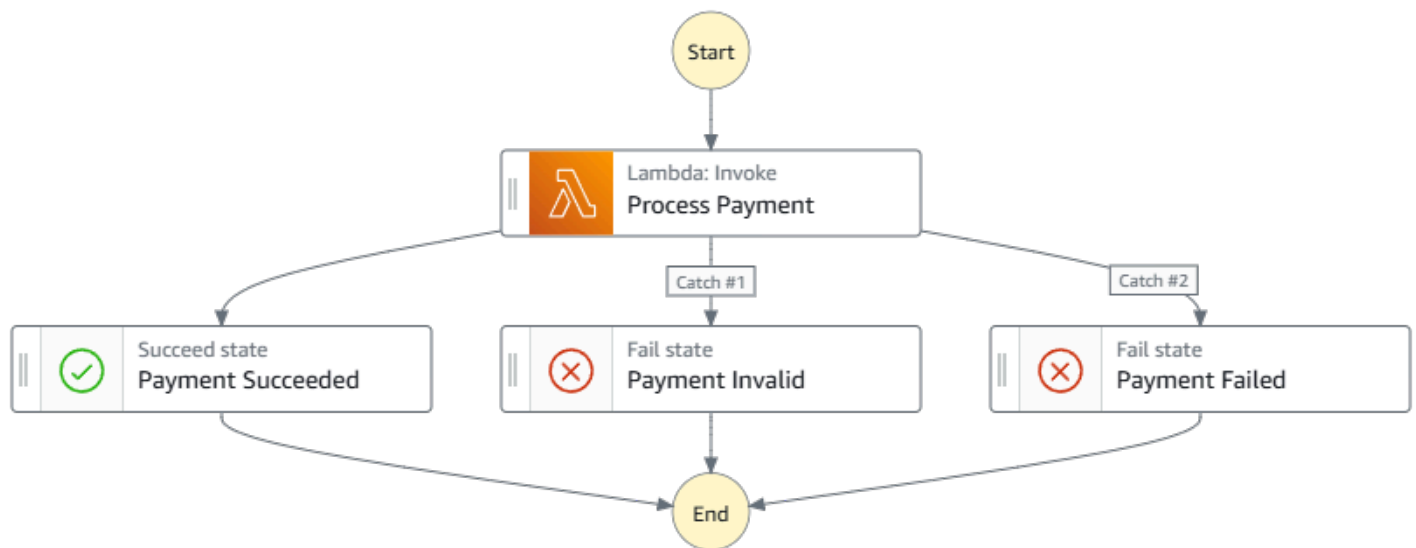
- Tenta di chiamare un servizio di elaborazione dei pagamenti
- Se il servizio di pagamento non è disponibile, la funzione attende e riprova più tardi.
- Implementa un backoff esponenziale personalizzato per il tempo di attesa
- Dopo che tutti i tentativi falliscono, individua l'errore e scegli un altro flusso

### Approccio consigliato

Utilizza un'unica funzione Lambda focalizzata esclusivamente sull'elaborazione dei pagamenti. Step Functions gestisce la gestione degli errori tramite:

- [Riprova automatica delle attività non riuscite con periodi di backoff configurabili](#)
- Applicazione di criteri di riprova diversi in base ai tipi di errore
- Instradamento di diversi tipi di errori agli stati di fallback appropriati
- Mantenimento dello stato e della cronologia di gestione degli errori

## Example grafico del flusso di lavoro



### Note

Code-first alternativa: le funzioni durevoli forniscono la gestione degli errori try-catch con strategie di riprova configurabili. Vedi Gestione [degli errori nelle funzioni durevoli](#).

## Flussi di lavoro condizionali e approvazioni umane

Usa lo stato Step Functions [Choice](#) per indirizzare i flussi di lavoro in base all'output delle funzioni e al ForTaskToken suffisso di [attesa per mettere in pausa i flussi di lavoro](#) per prendere decisioni umane. Ad esempio, per elaborare una richiesta di aumento del limite di credito, utilizza una funzione Lambda per valutare i fattori di rischio. Quindi, utilizza Step Functions per indirizzare le richieste ad alto rischio all'approvazione manuale e le richieste a basso rischio all'approvazione automatica.

Per implementare un flusso di lavoro di esempio che utilizza un pattern di integrazione del token di callback task, vedi [Callback with Task Token in The Workshop](#). AWS Step Functions

## Anti-pattern esempio

Una singola funzione Lambda gestisce un flusso di lavoro di approvazione complesso mediante:

- Implementazione di una logica condizionale annidata per valutare le richieste di credito
- Richiamo di diverse funzioni di approvazione in base agli importi delle richieste

- Gestione di più percorsi di approvazione e punti decisionali
- Monitoraggio dello stato delle approvazioni in sospeso
- Implementazione della logica di timeout e notifica per le approvazioni

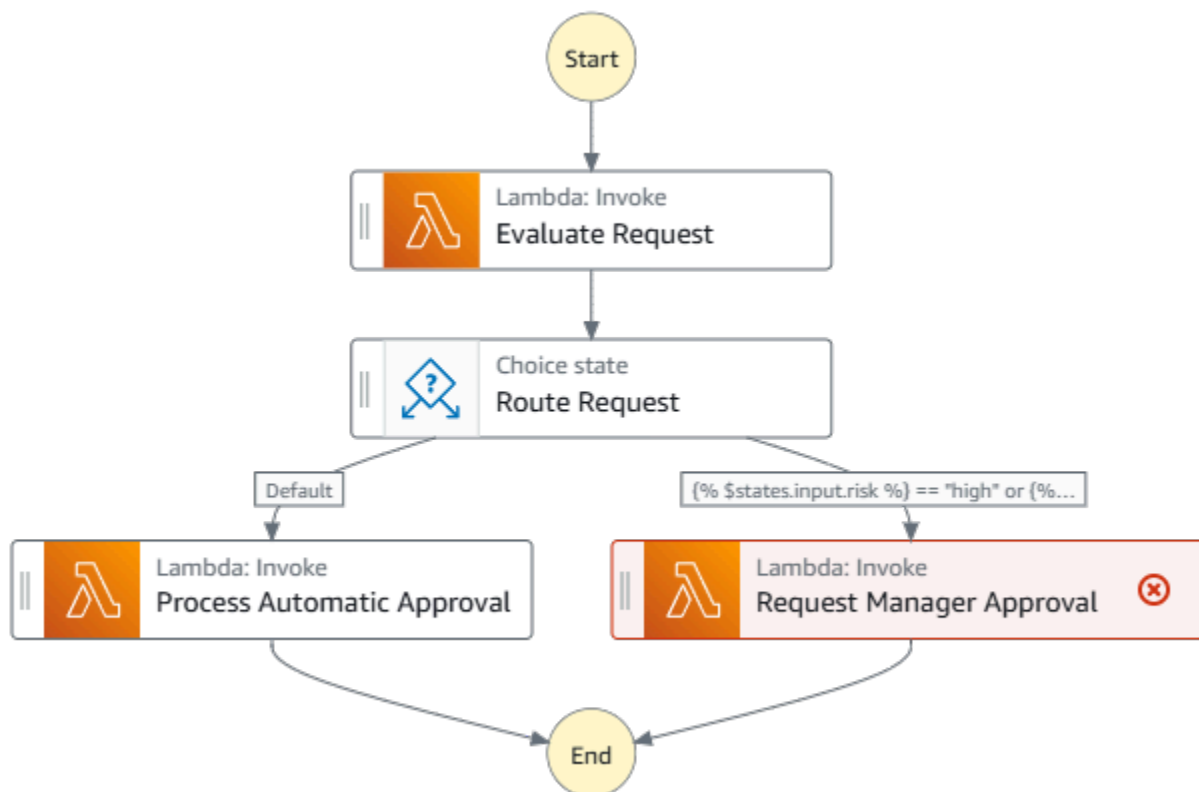
### Approccio consigliato

Utilizza tre funzioni Lambda: una per valutare il rischio di ogni richiesta, una per approvare le richieste a basso rischio e una per indirizzare le richieste ad alto rischio a un responsabile per la revisione.

Step Functions gestisce il flusso di lavoro tramite:

- Utilizzo [degli stati](#) Choice per indirizzare le richieste in base all'importo e al livello di rischio
- Sospensione dell'esecuzione in attesa dell'approvazione umana
- Gestione dei timeout per le approvazioni in sospeso
- Fornire visibilità sullo stato corrente di ogni richiesta

### Example grafico del flusso di lavoro



 Note

Code-first alternativa: le funzioni durevoli supportano i callback per flussi di lavoro «human-in-the-loop». Vedi [Callback nelle funzioni durevoli](#).

## Elaborazione parallela

Step Functions offre tre modi per gestire l'elaborazione parallela:

- Lo stato [Parallel](#) esegue più rami del flusso di lavoro contemporaneamente. Usalo quando devi eseguire diverse funzioni in parallelo, come la generazione di miniature durante l'estrazione dei metadati delle immagini.
- Lo stato [Inline Map](#) elabora matrici di dati con un massimo di 40 iterazioni simultanee. Usalo per set di dati di piccole e medie dimensioni in cui devi eseguire la stessa operazione su ogni elemento.
- Lo stato [Distributed Map](#) gestisce l'elaborazione parallela su larga scala con un massimo di 10.000 esecuzioni simultanee, supportando sia gli array JSON che le origini dati Amazon Simple Storage Service (Amazon S3). Utilizzalo quando elabori set di dati di grandi dimensioni o quando hai bisogno di una maggiore concorrenza.

## Anti-pattern esempio

Una singola funzione Lambda tenta di gestire l'elaborazione parallela mediante:

- Richiamando contemporaneamente più funzioni di elaborazione delle immagini
- Implementazione della logica di esecuzione parallela personalizzata
- Gestione dei timeout e della gestione degli errori per ogni attività parallela
- Raccolta e aggregazione dei risultati di tutte le funzioni

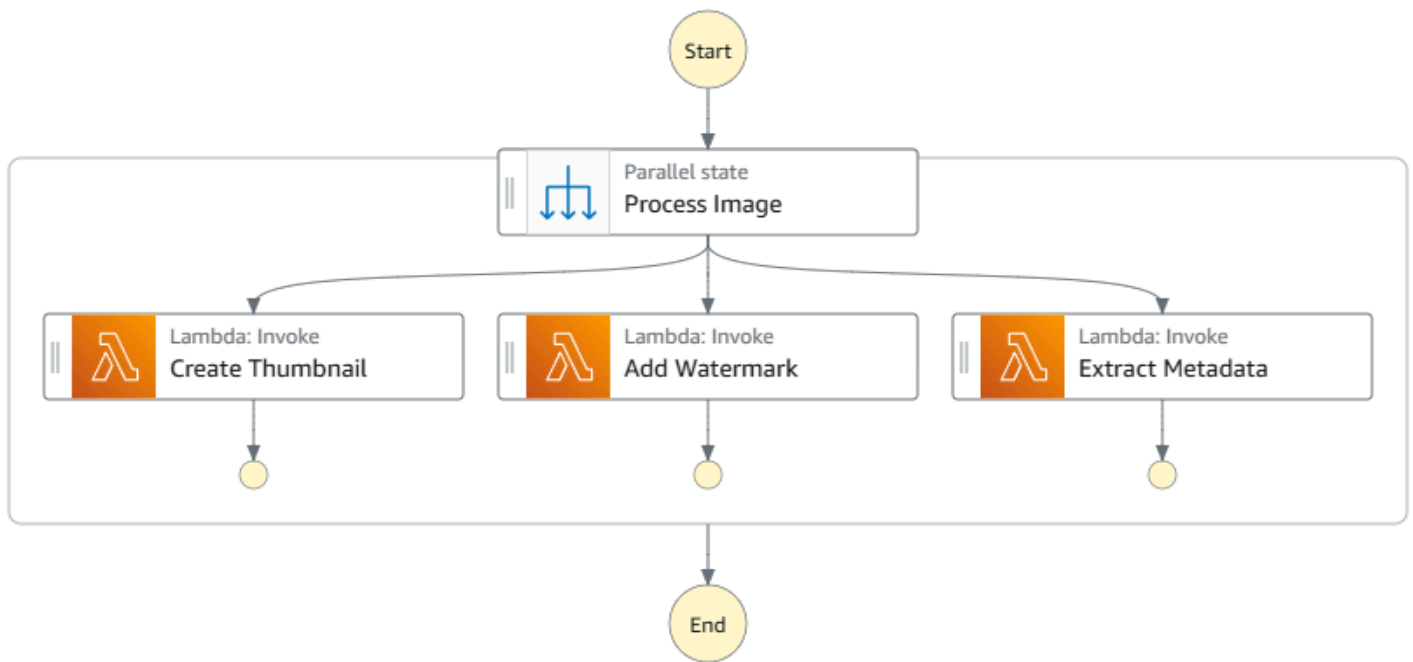
## Approccio consigliato

Utilizza tre funzioni Lambda: una per creare un'immagine in miniatura, una per aggiungere una filigrana e una per estrarre i metadati. Step Functions gestisce queste funzioni tramite:

- Esecuzione simultanea di tutte le funzioni utilizzando lo [stato](#) [Parallel](#)
- Raccolta dei risultati di ciascuna funzione in una matrice ordinata

- Gestione dei timeout e della gestione degli errori in tutte le esecuzioni parallele
- Procedendo solo quando tutti i rami paralleli sono completi

Example grafico del flusso di lavoro



#### Note

Code-first alternativa: le funzioni durevoli forniscono operazioni. `parallel()` `map()` Vedere Esecuzione [parallela](#).

## Quando non usare Step Functions con Lambda

Non tutte le Lambda-based applicazioni traggono vantaggio dall'utilizzo di Step Functions. Considerate questi scenari nella scelta dell'architettura dell'applicazione.

- [Applicazioni semplici](#)
- [Elaborazione complessa dei dati](#)
- [CPU-intensive carichi di lavoro](#)



## Applicazioni semplici

### Note

Per i flussi di lavoro che non richiedono una progettazione visiva o integrazioni estese di servizi, le funzioni durevoli di [Lambda](#) potrebbero essere un'alternativa più semplice che mantiene la logica del flusso di lavoro nel codice all'interno di Lambda.

Per le applicazioni che non richiedono un'orchestrazione complessa, l'utilizzo di Step Functions potrebbe aggiungere complessità non necessarie. Ad esempio, se stai semplicemente elaborando messaggi da una coda Amazon SQS o rispondendo a EventBridge eventi Amazon, puoi configurare questi servizi per richiamare direttamente le tue funzioni Lambda. Allo stesso modo, se la tua applicazione è composta solo da una o due funzioni Lambda con una semplice gestione degli errori, l'invocazione diretta di Lambda o le architetture basate sugli eventi potrebbero essere più semplici da implementare e mantenere.

### Elaborazione complessa dei dati

Puoi utilizzare lo [stato Step Functions](#) Distributed Map per elaborare contemporaneamente set di dati Amazon S3 di grandi dimensioni con funzioni Lambda. Ciò è efficace per molti carichi di lavoro paralleli su larga scala, inclusa l'elaborazione di dati semistrutturati come file JSON o CSV. Tuttavia, per trasformazioni di dati più complesse o analisi avanzate, considera queste alternative:

- Pipeline di trasformazione dei dati: da utilizzare AWS Glue per lavori ETL che elaborano dati strutturati o semistrutturati da più fonti. AWS Glue è particolarmente utile quando sono necessarie funzionalità integrate di gestione del catalogo dei dati e degli schemi.
- Analisi dei dati: usa Amazon EMR per l'analisi dei dati su scala di petabyte, soprattutto quando hai bisogno di strumenti Apache Hadoop ecosistemici o per carichi di lavoro di machine learning che superano i limiti di memoria di Lambda. [???](#)

### CPU-intensive carichi di lavoro

Sebbene Step Functions sia in grado di orchestrare CPU-intensive le attività, le funzioni Lambda potrebbero non essere adatte a questi carichi di lavoro a causa delle risorse CPU limitate. Per operazioni ad alta intensità di calcolo all'interno dei flussi di lavoro, considera queste alternative:

- Orchestrazione dei container: utilizza Step Functions per gestire le attività di Amazon Elastic Container Service (Amazon ECS) per risorse di calcolo più coerenti e scalabili.

- Elaborazione in batch: integra AWS Batch Step Functions per gestire processi batch ad alta intensità di calcolo che richiedono un utilizzo prolungato della CPU.

## Richiamo di una funzione Lambda con eventi batch Amazon S3

È possibile utilizzare le operazioni batch di Amazon S3 per richiamare una funzione Lambda su un set di oggetti Amazon S3 di grandi dimensioni. Amazon S3 tiene traccia dello stato di avanzamento delle operazioni batch, invia notifiche e memorizza un report di completamento che mostra lo stato di ogni operazione.

Per eseguire un'operazione batch, è possibile creare un [lavoro di operazioni batch](#) Amazon S3. Quando si crea il lavoro, si fornisce un file manifest (l'elenco degli oggetti) e si configura l'operazione da eseguire su tali oggetti.

All'avvio del lavoro batch, Amazon S3 richiama la funzione Lambda in modo [sincrono](#) per ogni oggetto nel manifest. Il parametro evento include i nomi del bucket e dell'oggetto.

Nell'esempio seguente viene illustrato l'evento che Amazon S3 invia alla funzione Lambda per un oggetto denominato `customerImage1.jpg` nel bucket `amzn-s3-demo-bucket`.

### Example Evento di richiesta batch Amazon S3

```
{
  "invocationSchemaVersion": "1.0",
  "invocationId": "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
  "job": {
    "id": "f3cc4f60-61f6-4a2b-8a21-d07600c373ce"
  },
  "tasks": [
    {
      "taskId": "dGFza2lkZ29lc2h1cmUK",
      "s3Key": "customerImage1.jpg",
      "s3VersionId": "1",
      "s3BucketArn": "arn:aws:s3:::amzn-s3-demo-bucket"
    }
  ]
}
```

La funzione Lambda deve restituire un oggetto JSON con i campi come mostrato nell'esempio seguente. È possibile copiare `invocationId` e `taskId` dal parametro evento. È possibile restituire una stringa in `resultString`. Amazon S3 salva i valori `resultString` nel report di completamento.

## Example Risposta alla richiesta batch Amazon S3

```
{
  "invocationSchemaVersion": "1.0",
  "treatMissingKeysAs" : "PermanentFailure",
  "invocationId" : "YXNkbGZqYWVmaBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
  "results": [
    {
      "taskId": "dGFza2lkZ29lc2hlcmUK",
      "resultCode": "Succeeded",
      "resultString": "[\"Alice\", \"Bob\"]"
    }
  ]
}
```

## Chiamata di funzioni Lambda dalle operazioni in batch Amazon S3

È possibile richiamare la funzione Lambda con una funzione ARN non qualificata o qualificata. Se si desidera utilizzare la stessa versione di funzione per l'intero lavoro batch, configurare una versione di funzione specifica nel parametro `FunctionARN` quando si crea il lavoro. Se si configura un alias o il qualificatore `$LATEST`, il lavoro batch inizia immediatamente a chiamare la nuova versione della funzione se l'alias o `$LATEST` viene aggiornato durante l'esecuzione del lavoro.

Si noti che non è possibile riutilizzare una funzione basata su eventi Amazon S3 esistente per le operazioni batch. Questo perché l'operazione batch Amazon S3 passa un parametro di evento diverso alla funzione Lambda e si aspetta un messaggio di ritorno con una struttura JSON specifica.

Nella [policy basata sulle risorse](#) creata per il lavoro batch Amazon S3, assicurarsi di impostare l'autorizzazione per il lavoro per richiamare la funzione Lambda.

Nel [ruolo di esecuzione](#) per la funzione, impostare una policy di attendibilità Amazon S3 per assumere il ruolo quando viene eseguita la funzione.

Se la tua funzione utilizza l' AWS SDK per gestire le risorse Amazon S3, devi aggiungere le autorizzazioni Amazon S3 nel ruolo di esecuzione.

Quando il lavoro viene eseguito, Amazon S3 avvia più istanze di funzione per elaborare gli oggetti Amazon S3 in parallelo, fino al [limite di simultaneità](#) della funzione. Amazon S3 limita l'aumento iniziale delle istanze per evitare costi eccessivi per i lavori più piccoli.

Se la funzione Lambda restituisce un codice di risposta `TemporaryFailure`, Amazon S3 riattiva l'operazione.

Per ulteriori informazioni sulla gestione delle operazioni in batch Amazon S3, consulta [Gestione dei processi di operazioni in batch](#) nella Guida per gli sviluppatori di Amazon S3.

Per un esempio di come utilizzare una funzione Lambda nelle operazioni batch di Amazon S3, consulta [Invocare una funzione Lambda da operazioni batch di Amazon S3](#) nella Guida per gli sviluppatori di Amazon S3.

## Richiamo di funzioni Lambda mediante notifiche Amazon SNS

Si può utilizzare una funzione Lambda per elaborare le notifiche Amazon Simple Notification Service (Amazon SNS). Amazon SNS supporta le funzioni Lambda come destinazione per i messaggi inviati a un argomento. Puoi sottoscrivere la funzione ad argomenti nello stesso account o in altri account AWS. Per la procedura guidata dettagliata, consulta [the section called “Tutorial”](#).

Lambda supporta i trigger SNS solo per argomenti SNS standard. Gli argomenti FIFO non sono supportati.

Lambda elabora i messaggi SNS in modo asincrono mettendo in coda i messaggi e gestendo i tentativi. Se Amazon SNS non è in grado di raggiungere Lambda o il messaggio viene rifiutato, Amazon SNS riprova a intervalli crescenti per diverse ore. Per ulteriori informazioni, consulta [Affidabilità](#) nelle domande frequenti su Amazon SNS.

### Warning

Le chiamate asincrone Lambda elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per saperne di più, consulta [Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center](#). AWS

## Utilità di idempotenza di Powertools per AWS Lambda

L'utilità di idempotenza di Powertools for Lambda rende le funzioni Lambda idempotenti. AWS È disponibile per Python, Java e .NET. TypeScript Per ulteriori informazioni, vedere [Idempotency utility](#) nella documentazione di Powertools for AWS Lambda (Python), [Idempotency Utility](#) nella documentazione di Powertools for Lambda (Java), [Idempotency Utility](#) nella documentazione di Powertools for AWS Lambda (JavaScript) e [Idempotency Utility](#) nella documentazione di Powertools for AWS Lambda (.NET). <https://docs.aws.amazon.com/powertools/dotnet/utilities/idempotency/> AWS

### Argomenti

- [Aggiunta di un trigger di argomento Amazon SNS per una funzione Lambda utilizzando la console](#)
- [Aggiunta manuale di un trigger di argomento Amazon SNS per una funzione Lambda](#)

- [Esempio di forma evento SNS](#)
- [Tutorial: Utilizzo AWS Lambda con Amazon Simple Notification Service](#)

## Aggiunta di un trigger di argomento Amazon SNS per una funzione Lambda utilizzando la console

Per aggiungere un argomento SNS come trigger per una funzione Lambda, il modo più semplice è utilizzare la console Lambda. Quando aggiungi il trigger tramite la console, Lambda imposta automaticamente le autorizzazioni e gli abbonamenti necessari per iniziare a ricevere eventi dall'argomento SNS.

Per aggiungere un argomento SNS come trigger per una funzione Lambda (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli il nome della funzione per la quale si desidera aggiungere il trigger.
3. Scegli Configurazione, quindi scegli Trigger.
4. Selezionare Add trigger (Aggiungi trigger).
5. In Configurazione del trigger, nell'elenco a discesa, scegli SNS.
6. Per Argomento SNS, scegli l'argomento SNS da sottoscrivere.

## Aggiunta manuale di un trigger di argomento Amazon SNS per una funzione Lambda

Per configurare manualmente un trigger SNS per una funzione Lambda, è necessario completare i seguenti passaggi:

- Definisci una policy basata sulle risorse per la tua funzione per consentire a SNS di richiamarla.
- Sottoscrivere la funzione Lambda all'argomento Amazon SNS.

### Note

Se l'argomento SNS e la funzione Lambda si trovano in AWS account diversi, devi anche concedere autorizzazioni aggiuntive per consentire le sottoscrizioni all'argomento SNS tra più account. Per maggiori informazioni, consulta [Concedere l'autorizzazioni tra account per la sottoscrizione di Amazon SNS](#).

Puoi usare AWS Command Line Interface (AWS CLI) per completare entrambi questi passaggi. Innanzitutto, per definire una policy basata sulle risorse per una funzione Lambda che consenta le chiamate SNS, usa il comando AWS CLI seguente. Assicurati di sostituire il valore di `--function-name` con il nome della funzione Lambda e il valore di `--source-arn` con l'ARN dell'argomento SNS.

```
aws lambda add-permission --function-name example-function \  
  --source-arn arn:aws:sns:us-east-1:123456789012:sns-topic-for-lambda \  
  --statement-id function-with-sns --action "lambda:InvokeFunction" \  
  --principal sns.amazonaws.com
```

Per iscrivere la tua funzione all'argomento SNS, usa il seguente AWS CLI comando. Assicurati di sostituire il valore di `--topic-arn` con l'ARN dell'argomento SNS e il valore di `--notification-endpoint` con l'ARN della funzione Lambda.

```
aws sns subscribe --protocol lambda \  
  --region us-east-1 \  
  --topic-arn arn:aws:sns:us-east-1:123456789012:sns-topic-for-lambda \  
  --notification-endpoint arn:aws:lambda:us-east-1:123456789012:function:example-function
```

## Esempio di forma evento SNS

Amazon SNS richiama la funzione in modo [asincrono](#) con un evento che contiene un messaggio e dei metadati.

### Example Evento messaggio di Amazon SNS

```
{  
  "Records": [  
    {  
      "EventVersion": "1.0",  
      "EventSubscriptionArn": "arn:aws:sns:us-east-1:123456789012:sns-lambda:21be56ed-a058-49f5-8c98-aedd2564c486",  
      "EventSource": "aws:sns",  
      "Sns": {  
        "SignatureVersion": "1",  
        "Timestamp": "2019-01-02T12:45:07.000Z",  
        "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEKai6RibDsvpi+tE/1+82j...65r==",  
        "SigningCertURL": "https://sns.us-east-1.amazonaws.com/SimpleNotificationService-ac565b8b1a6c5d002d285f9598aa1d9b.pem",
```



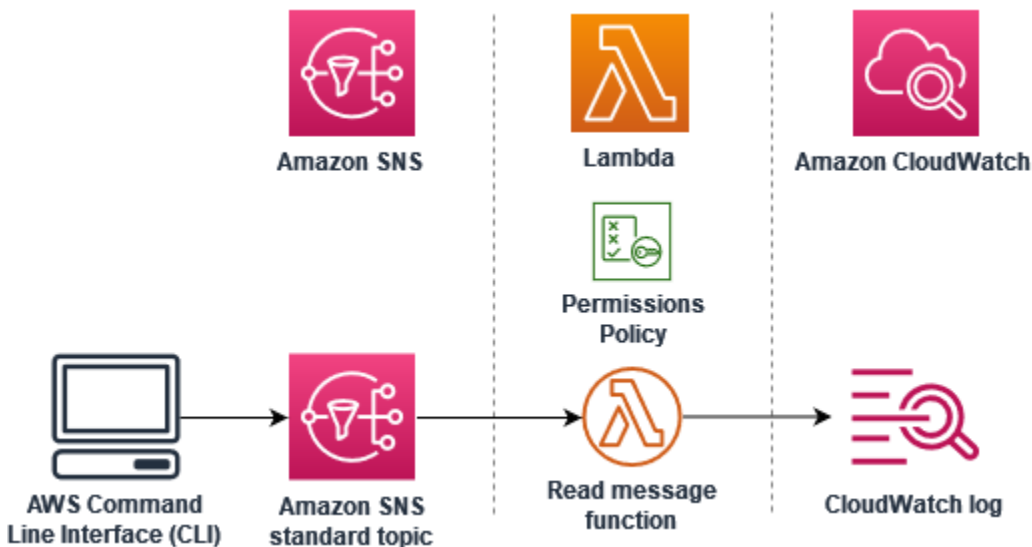
```

    "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
    "Message": "Hello from SNS!",
    "MessageAttributes": {
      "Test": {
        "Type": "String",
        "Value": "TestString"
      },
      "TestBinary": {
        "Type": "Binary",
        "Value": "TestBinary"
      }
    },
    "Type": "Notification",
    "UnsubscribeUrl": "https://sns.us-east-1.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:123456789012:test-
lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
    "TopicArn": "arn:aws:sns:us-east-1:123456789012:sns-lambda",
    "Subject": "TestInvoke"
  }
}
]
}

```

## Tutorial: Utilizzo AWS Lambda con Amazon Simple Notification Service

In questo tutorial, utilizzi una funzione Lambda in uno Account AWS per abbonarti a un argomento separato di Amazon Simple Notification Service (Amazon SNS). Account AWS Quando pubblichi messaggi sul tuo argomento Amazon SNS, la tua funzione Lambda legge il contenuto del messaggio e lo invia ad Amazon Logs. CloudWatch Per completare questo tutorial, usi (). AWS Command Line Interface AWS CLI



Per completare questo tutorial, esegui i passaggi riportati:

- Nell'account A, crea un argomento Amazon SNS.
- Nell'account B, crea una funzione Lambda che legge i messaggi dall'argomento.
- Nell'account B, crea una sottoscrizione all'argomento.
- Pubblica messaggi sull'argomento Amazon SNS nell'account A e conferma che la funzione Lambda nell'account B li trasmetta in Logs. CloudWatch

Completando questi passaggi, impari a configurare un argomento di Amazon SNS per richiamare una funzione Lambda. Scopri anche come creare una policy AWS Identity and Access Management (IAM) che autorizzi una risorsa in un'altra a Account AWS invocare Lambda.

Nel tutorial, vengono utilizzati due Account AWS separati. I AWS CLI comandi lo illustrano utilizzando due profili denominati `accountA` e `accountB`, ciascuno configurato per l'uso con un profilo diverso. Account AWS Per informazioni su come configurare l' AWS CLI utilizzo di profili diversi, vedere [Configurazione e impostazioni dei file di credenziali](#) nella Guida per l'AWS Command Line Interface utente della versione 2. Assicurati di configurare la stessa impostazione predefinita Regione AWS per entrambi i profili.

Se i AWS CLI profili creati per entrambi Account AWS utilizzano nomi diversi o se utilizzi il profilo predefinito e un profilo con nome, modifica i AWS CLI comandi nei passaggi seguenti, se necessario.

## Prerequisiti

### Installa AWS Command Line Interface

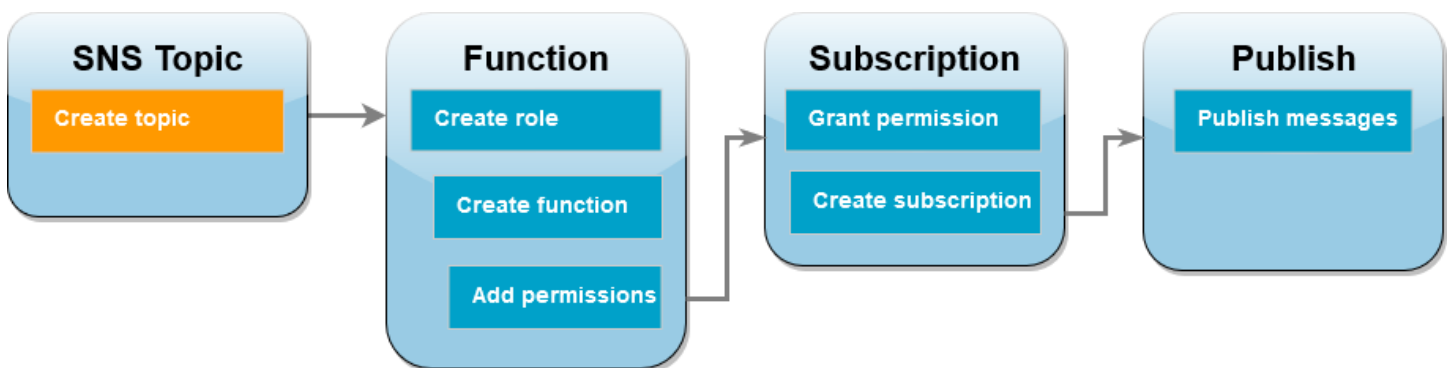
Se non hai ancora installato il AWS Command Line Interface, segui i passaggi in [Installazione o aggiornamento della versione più recente AWS CLI](#) per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

#### Note

Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere una Windows-integrated versione di Ubuntu e Bash, [installa il sottosistema Windows per Linux](#).

### Creare un argomento Amazon SNS (account A)



### Per creare l'argomento

- Nell'account A, crea un argomento standard di Amazon SNS utilizzando il seguente AWS CLI comando.

```
aws sns create-topic --name sns-topic-for-lambda --profile accountA
```

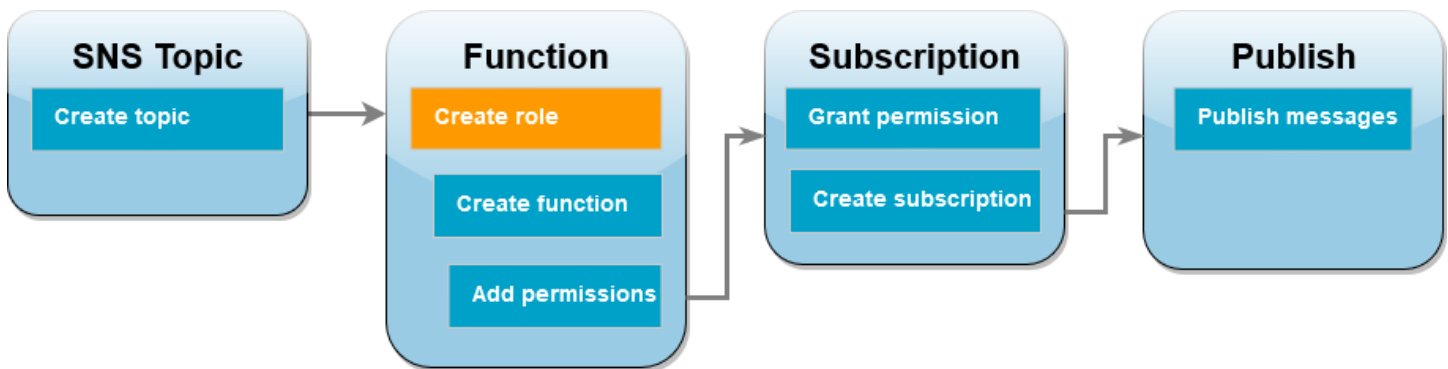
Verrà visualizzato un output simile al seguente.

```
{
```

```
"TopicArn": "arn:aws:sns:us-west-2:123456789012:sns-topic-for-lambda"
}
```

Prendi nota del nome della risorsa Amazon (ARN) dell'argomento. Ne avrai bisogno più avanti nel tutorial quando aggiungerai le autorizzazioni alla tua funzione Lambda per iscriverti all'argomento.

#### Creazione di un ruolo di esecuzione della funzione (account B)



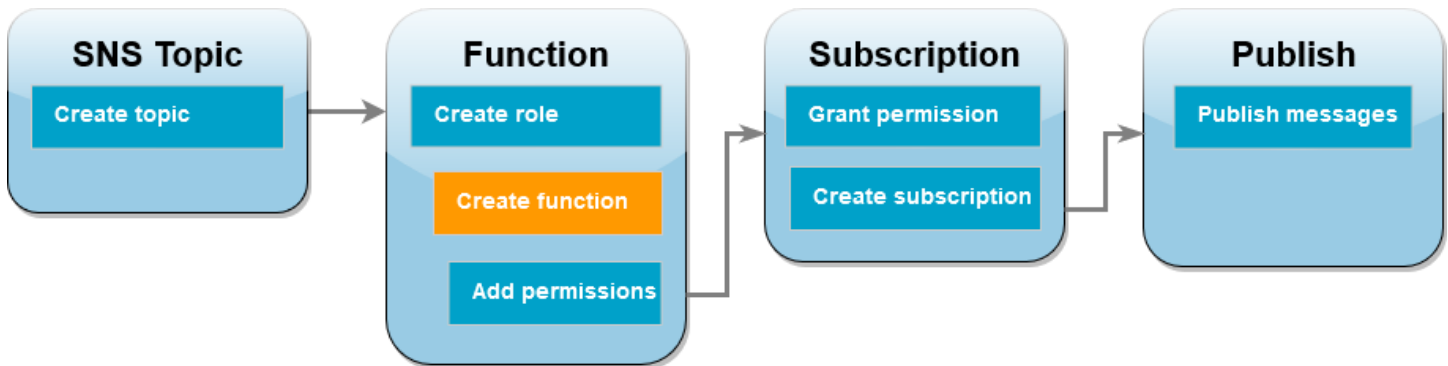
Un ruolo di esecuzione è un ruolo IAM che concede a una funzione Lambda l'autorizzazione all'accesso Servizi AWS e alle risorse. Prima di creare la funzione nell'account B, si crea un ruolo che concede alla funzione le autorizzazioni di base per scrivere i log nei log. CloudWatch Aggiungeremo le autorizzazioni di lettura dal tuo argomento Amazon SNS in una fase successiva.

#### Creazione di un ruolo di esecuzione

1. Nell'account B apri la [pagina dei ruoli](#) nella console IAM.
2. Scegliere Crea ruolo.
3. Per Tipo di entità attendibile, scegli Servizio AWS .
4. In Caso d'uso, scegli Lambda.
5. Scegli Avanti.
6. Aggiungi una policy di autorizzazioni di base al ruolo completando le seguenti operazioni:
  - a. Nella casella di ricerca Policy di autorizzazione, inserisci **AWSLambdaBasicExecutionRole**.
  - b. Scegli Avanti.
7. Completa la creazione del ruolo effettuando le seguenti operazioni:
  - a. In Dettagli ruolo, immetti **lambda-sns-role** per Nome ruolo.

## b. Scegliere Crea ruolo.

Creare una funzione Lambda (account B)



Crea una funzione Lambda che elabora i messaggi Amazon SNS. Il codice della funzione registra il contenuto dei messaggi di ogni record in Amazon CloudWatch Logs.

Questo tutorial utilizza il runtime Node.js 24, ma abbiamo anche fornito codice di esempio in altri linguaggi di runtime. Per visualizzare il codice per il runtime che ti interessa, seleziona la scheda corrispondente nella casella seguente. Il JavaScript codice che utilizzerai in questo passaggio si trova nel primo esempio mostrato nella [JavaScript](#) scheda.

.NET

SDK per .NET

### Note

C'è altro su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
into a .NET class.
```

```
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record,
        ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
            Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

## Go

## SDK per Go V2

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
    for _, record := range snsEvent.Records {
        processMessage(record)
    }
    fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
    message := record.SNS.Message
    fmt.Printf("Processed message: %s\n", message)
    // TODO: Process your record here
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import com.amazonaws.services.lambda.runtime.events.SNSEvent.SNSRecord;

import java.util.Iterator;
import java.util.List;

public class SNSEventHandler implements RequestHandler<SNSEvent, Boolean> {
    LambdaLogger logger;

    @Override
    public Boolean handleRequest(SNSEvent event, Context context) {
        logger = context.getLogger();
        List<SNSRecord> records = event.getRecords();
        if (!records.isEmpty()) {
            Iterator<SNSRecord> recordsIter = records.iterator();
            while (recordsIter.hasNext()) {
                processRecord(recordsIter.next());
            }
        }
        return Boolean.TRUE;
    }

    public void processRecord(SNSRecord record) {
```



```
        try {
            String message = record.getSNS().getMessage();
            logger.log("message: " + message);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Utilizzo di un evento SNS utilizzando Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    for (const record of event.Records) {
        await processMessageAsync(record);
    }
    console.info("done");
};

async function processMessageAsync(record) {
    try {
        const message = JSON.stringify(record.Sns.Message);
        console.log(`Processed message ${message}`);
        await Promise.resolve(1); //Placeholder for actual async work
    } catch (err) {
        console.error("An error occurred");
    }
}
```

```
    throw err;
  }
}
```

## Consumo di un evento SNS con Lambda using TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

## PHP

### SDK per PHP

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Utilizzo di un evento SNS con Lambda tramite PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

/*
Since native PHP support for AWS Lambda is not available, we are utilizing Bref's
PHP functions runtime for AWS Lambda.
For more information on Bref's PHP runtime for Lambda, refer to: https://bref.sh/docs/runtimes/function

Another approach would be to create a custom runtime.
A practical example can be found here: https://aws.amazon.com/blogs/apn/aws-lambda-custom-runtime-for-php-a-practical-example/
*/

// Additional composer packages may be required when using Bref or any other PHP
functions runtime.
// require __DIR__ . '/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Sns\SnsEvent;
use Bref\Event\Sns\SnsHandler;

class Handler extends SnsHandler
{
    public function handleSns(SnsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $message = $record->getMessage();

            // TODO: Implement your custom processing logic here
            // Any exception thrown will be logged and the invocation will be
            marked as failed

            echo "Processed Message: $message" . PHP_EOL;
        }
    }
}

return new Handler();
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for record in event['Records']:
        process_message(record)
    print("done")

def process_message(record):
    try:
        message = record['Sns']['Message']
        print(f"Processed message {message}")
        # TODO; Process your record here

    except Exception as e:
        print("An error occurred")
        raise e
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Utilizzo di un evento SNS con Lambda tramite Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].map { |record| process_message(record) }
end

def process_message(record)
  message = record['Sns']['Message']
  puts("Processing message: #{message}")
rescue StandardError => e
  puts("Error processing message: #{e}")
  raise
end
```

## Rust

### SDK per Rust

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Utilizzo di un evento SNS con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
// aws_lambda_events = { version = "0.10.0", default-features = false, features
//   = ["sns"] }
// lambda_runtime = "0.8.1"
// tokio = { version = "1", features = ["macros"] }
```

```
// tracing = { version = "0.1", features = ["log"] }
// tracing-subscriber = { version = "0.3", default-features = false, features =
  ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for event in event.payload.records {
        process_record(&event)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);

    // Implement your record handling code here.

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

## Creazione della funzione

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir sns-tutorial
cd sns-tutorial
```

2. Copia il JavaScript codice di esempio in un nuovo file denominato `index.js`.
3. Crea un pacchetto di implementazione utilizzando il seguente comando `zip`.

```
zip function.zip index.js
```

4. Esegui il AWS CLI comando seguente per creare la tua funzione Lambda nell'account B.

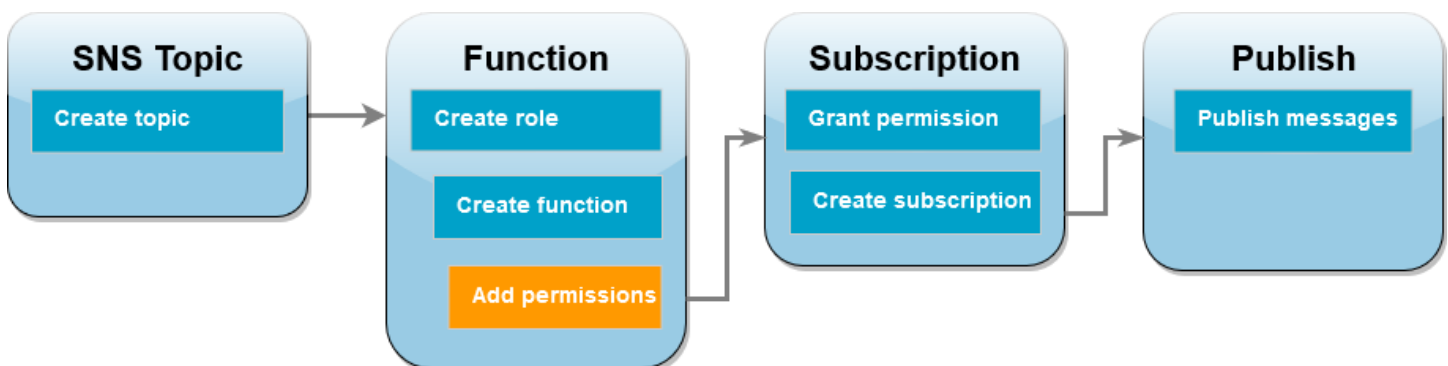
```
aws lambda create-function --function-name Function-With-SNS \  
  --zip-file fileb://function.zip --handler index.handler --runtime nodejs24.x \  
  --role arn:aws:iam::<AccountB_ID>:role/lambda-sns-role \  
  --timeout 60 --profile accountB
```

Verrà visualizzato un output simile al seguente.

```
{  
  "FunctionName": "Function-With-SNS",  
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:Function-With-SNS",  
  "Runtime": "nodejs24.x",  
  "Role": "arn:aws:iam::123456789012:role/lambda_basic_role",  
  "Handler": "index.handler",  
  ...  
  "RuntimeVersionConfig": {  
    "RuntimeVersionArn": "arn:aws:lambda:us-west-2::runtime:7d5f06b69c951da8a48b926ce280a9daf2e8bb1a74fc4a2672580c787d608206"  
  }  
}
```

5. Registra il nome della risorsa Amazon (ARN) della funzione. Ne avrai bisogno più avanti nel tutorial quando aggiungerai le autorizzazioni per consentire ad Amazon SNS di richiamare la tua funzione.

Aggiunta di autorizzazioni alla funzione (account B)



Perché Amazon SNS richiami la tua funzione, è necessario concedergli l'autorizzazione in una istruzione su una [policy basata sulle risorse](#). Aggiungi questa dichiarazione usando il AWS CLI `add-permission` comando.

Concessione dell'autorizzazione di Amazon SNS per richiamare la tua funzione

- Nell'account B, esegui il AWS CLI comando seguente utilizzando l'ARN per l'argomento Amazon SNS che hai registrato in precedenza.

```
aws lambda add-permission --function-name Function-With-SNS \
  --source-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
  --statement-id function-with-sns --action "lambda:InvokeFunction" \
  --principal sns.amazonaws.com --profile accountB
```

Verrà visualizzato un output simile al seguente.

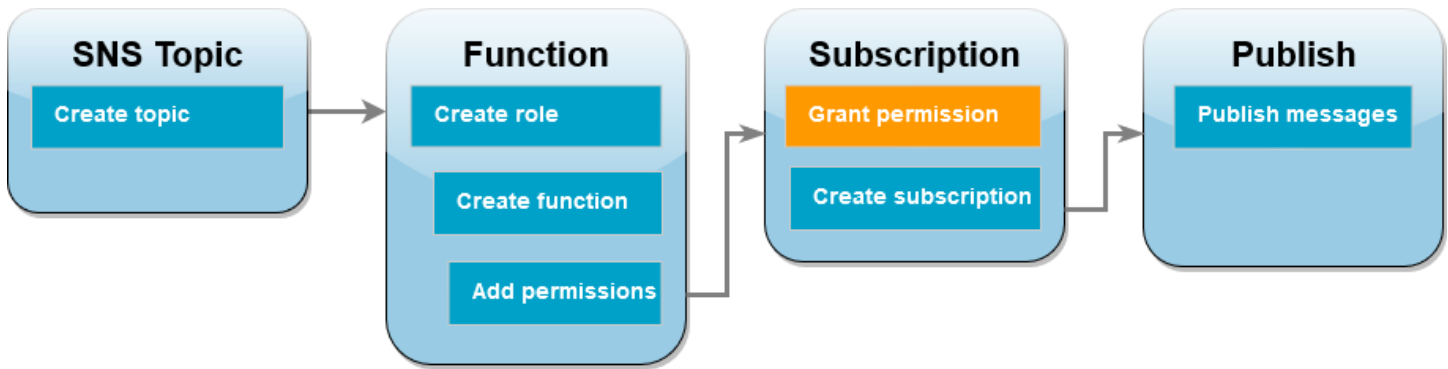
```
{
  "Statement": "{\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\n    \"arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda\"}},\n    \"Action\":[\"lambda:InvokeFunction\"],\n    \"Resource\":\"arn:aws:lambda:us-east-1:<AccountB_ID>:function:Function-With-SNS\"},\n    \"Effect\":\"Allow\", \"Principal\":{\"Service\":\"sns.amazonaws.com\"},\n    \"Sid\":\"function-with-sns\"}"
}
```

#### Note

Se l'account con l'argomento Amazon SNS è ospitato in un [opt-in Regione AWS](#), devi specificare la regione nell'indirizzo principale. Ad esempio, se stai lavorando con un argomento Amazon SNS nella regione Asia Pacifico (Hong Kong), per il principale devi specificare `sns.ap-east-1.amazonaws.com` invece di `sns.amazonaws.com`.



Concessione dell'autorizzazione tra più account per la sottoscrizione ad Amazon SNS (account A)



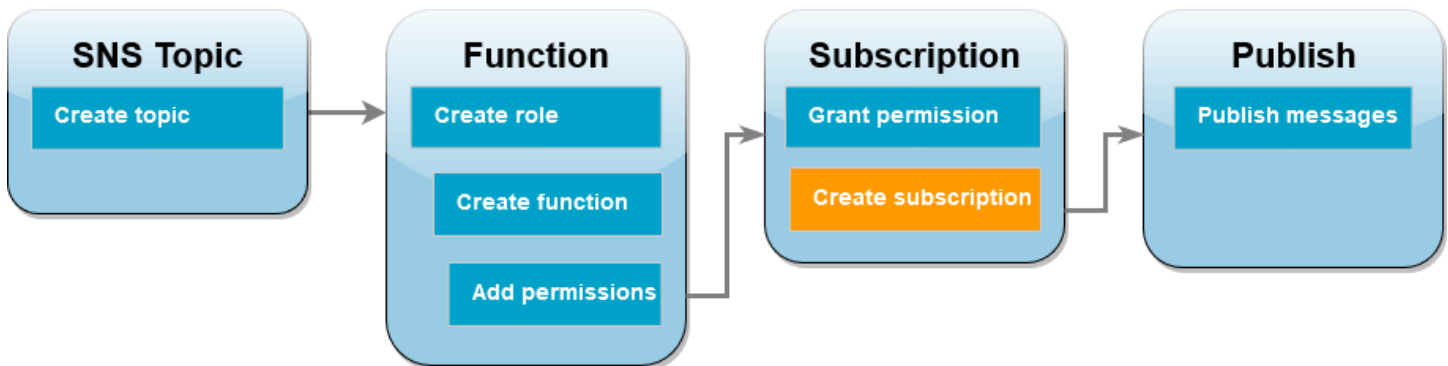
Perché la funzione Lambda nell'account B sottoscriva l'argomento Amazon SNS che hai creato nell'account A, è necessario concedere l'autorizzazione per l'account B in modo da sottoscrivere il tuo argomento. Concedi questa autorizzazione usando il AWS CLI `add-permission` comando.

Concessione dell'autorizzazione per consentire all'account B la sottoscrizione all'argomento

- Nell'account A, esegui il AWS CLI comando seguente. Usa l'ARN per l'argomento Amazon SNS registrato in precedenza.

```
aws sns add-permission --label lambda-access --aws-account-id <AccountB_ID> \
  --topic-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
  --action-name Subscribe ListSubscriptionsByTopic --profile accountA
```

Creare una sottoscrizione (account B)



Nell'account B, ora sottoscrivi la tua funzione Lambda all'argomento Amazon SNS che hai creato all'inizio del tutorial nell'account A. Quando viene inviato un messaggio a questo argomento (`sns-topic-for-lambda`), Amazon SNS richiama la funzione Lambda `Function-With-SNS` nell'account B.

## Creazione di una sottoscrizione

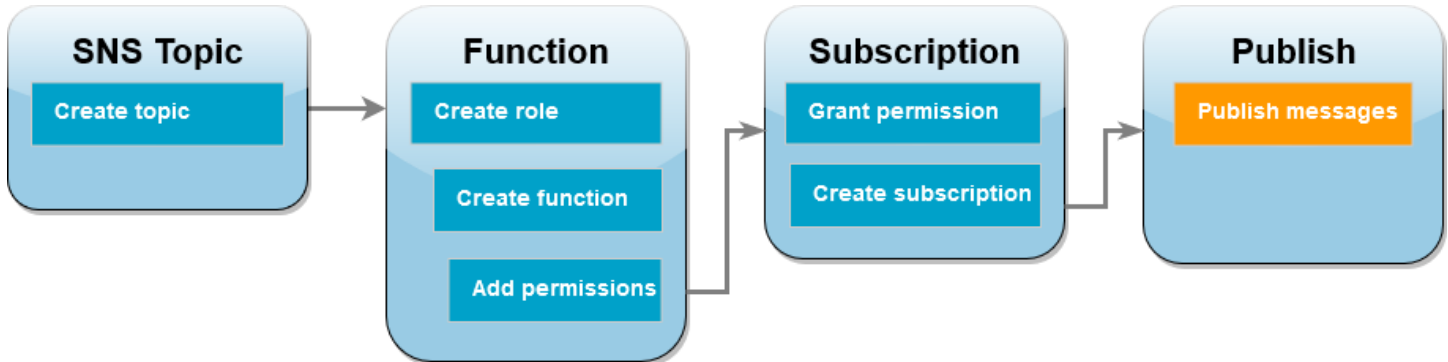
- Nell'account B, esegui il seguente AWS CLI comando. Usa la regione predefinita in cui hai creato l'argomento e gli ARN per l'argomento e la funzione Lambda.

```
aws sns subscribe --protocol lambda \  
  --region us-east-1 \  
  --topic-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \  
  --notification-endpoint arn:aws:lambda:us-east-1:<AccountB_ID>:function:Function-With-SNS \  
  --profile accountB
```

Verrà visualizzato un output simile al seguente.

```
{  
  "SubscriptionArn": "arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda:5d906xxxx-7c8x-45dx-a9dx-0484e31c98xx"  
}
```

## Pubblicazione di messaggi sull'argomento (account A e account B)



Ora che la tua funzione Lambda nell'account B è iscritta all'argomento Amazon SNS nell'account A, è il momento di testare la configurazione pubblicando messaggi sull'argomento. Per confermare che Amazon SNS abbia richiamato la tua funzione Lambda, utilizzi CloudWatch Logs per visualizzarne l'output.

Pubblicazione di un messaggio sull' argomento e visualizzazione dell'output della tua funzione

1. Digita `Hello World` in un file di testo e salvalo come `message.txt`.

2. Dalla stessa directory in cui hai salvato il file di testo, esegui il seguente AWS CLI comando nell'account A. Usa l'ARN per il tuo argomento.

```
aws sns publish --message file://message.txt --subject Test \  
  --topic-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \  
  --profile accountA
```

Ciò restituisce un ID del messaggio con un identificatore univoco, a indicare che Amazon SNS ha accettato il messaggio. Amazon SNS tenta quindi di recapitare il messaggio agli abbonati dell'argomento. Per confermare che Amazon SNS ha richiamato la tua funzione Lambda, utilizza CloudWatch Logs per visualizzare l'output della tua funzione:

3. Nell'account B, apri la [pagina dei gruppi di log](#) della console Amazon. CloudWatch
4. Scegli il nome del gruppo di log per la funzione (/aws/lambda/Function-With-SNS).
5. Scegli il flusso di log più recente.
6. Se la tua funzione è stata richiamata correttamente, vedrai un risultato simile al seguente che mostra il contenuto del messaggio che hai pubblicato sul tuo argomento.

```
2023-07-31T21:42:51.250Z c1cba6b8-ade9-4380-aa32-d1a225da0e48 INFO Processed  
message Hello World  
2023-07-31T21:42:51.250Z c1cba6b8-ade9-4380-aa32-d1a225da0e48 INFO done
```

## Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti inutili addebiti al tuo Account AWS

Nell'Account A, elimina l'argomento Amazon SNS.

Per eliminare l'argomento Amazon SNS

1. Aprire la [pagina Topics \(Argomenti\)](#) nella console Amazon SNS.
2. Selezionare l'argomento creato.
3. Scegliere Elimina.
4. Inserisci **delete me** nel campo di immissione del testo.
5. Scegliere Elimina.

Nell'Account B, elimina il ruolo di esecuzione, la funzione Lambda e la sottoscrizione Amazon SNS.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina [Ruoli](#) della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **confirm** nel campo di immissione testo e scegli Delete (Elimina).

Per eliminare la sottoscrizione Amazon SNS

1. Aprire la [pagina Subscriptions \(Sottoscrizioni\)](#) nella console Amazon SNS.
2. Selezionare la sottoscrizione creata.
3. Scegli Delete (Elimina), poi Delete (Elimina).

## Funzioni durevoli Lambda

Con funzioni AWS Lambda durevoli, puoi creare applicazioni resilienti in più fasi e flussi di lavoro AI che possono essere eseguiti per un massimo di un anno, mantenendo progressi affidabili nonostante le interruzioni. Quando viene eseguita una funzione duratura, questo ciclo di vita completo è chiamato esecuzione durevole, che utilizza dei checkpoint per tenere traccia dei progressi e ripristinare automaticamente gli errori tramite la ripetizione, rieseguendola dall'inizio saltando il lavoro completato.

All'interno di ogni funzione, si utilizzano operazioni durature come elementi costitutivi fondamentali. Le fasi eseguono la logica aziendale con tentativi e monitoraggio dei progressi integrati, mentre le attese sospendono l'esecuzione senza incorrere in costi di elaborazione, il che le rende ideali per processi di lunga durata come i flussi di lavoro umani o il polling delle dipendenze esterne. Che tu stia elaborando ordini, coordinando microservizi o orchestrando applicazioni AI agentiche, le funzioni

durevoli mantengono automaticamente lo stato e si ripristinano dagli errori mentre scrivi codice in linguaggi di programmazione familiari.

#### Note

Questa guida tratta argomenti relativi ai servizi Lambda come distribuzione, configurazione, autorizzazioni, invocazione, monitoraggio e quote. Per riferimenti alla programmazione SDK, tutorial di avvio rapido e guide specifiche per lingua, consulta la Durable Execution SDK Developer Guide. [AWS](#)

## Vantaggi principali

Scrivi codice resiliente in modo naturale: con costrutti di programmazione familiari, scrivi codice che gestisce automaticamente gli errori. Built-in grazie al checkpoint, ai tentativi trasparenti e al ripristino automatico, la logica aziendale rimane pulita e concentrata.

Pagate solo per ciò che utilizzate: durante le operazioni di attesa, la funzione viene sospesa senza incorrere in costi di elaborazione. Per i flussi di lavoro di lunga durata che attendono ore o giorni, paghi solo per il tempo di elaborazione effettivo, non per le attese inattive.

Semplicità operativa: con il modello serverless di Lambda, è possibile ottenere una scalabilità automatica, inclusa la scalabilità a zero, senza dover gestire l'infrastruttura. Le funzioni durevoli gestiscono automaticamente la gestione dello stato, la logica dei tentativi e il ripristino dei guasti, riducendo il sovraccarico operativo.

## Quando utilizzare le funzioni durevoli

Short-lived coordinamento: coordina i pagamenti, l'inventario e la spedizione tra più servizi con il rollback automatico in caso di guasti. Elabora gli ordini tramite convalida, autorizzazione al pagamento, allocazione dell'inventario ed evasione degli ordini con completamento garantito.

Elabora i pagamenti con sicurezza: crea flussi di pagamento resilienti che mantengono lo stato delle transazioni in caso di errori e gestiscono automaticamente i nuovi tentativi. Coordina l'autorizzazione in più fasi, i controlli antifrode e il regolamento tra i fornitori di servizi di pagamento con una completa verificabilità in tutte le fasi.

Crea flussi di lavoro IA affidabili: crea flussi di lavoro di intelligenza artificiale in più fasi che concatenano le chiamate di modelli, incorporano il feedback umano e gestiscono le attività di lunga

durata in modo deterministico durante gli errori. Riprendi automaticamente dopo la sospensione e paga solo per il tempo di esecuzione attivo.

Orchestra l'evasione complessa degli ordini: coordina l'elaborazione degli ordini nei sistemi di inventario, pagamento, spedizione e notifica con resilienza integrata. Gestisci automaticamente gli errori parziali, preserva lo stato degli ordini nonostante le interruzioni e attendi in modo efficiente gli eventi esterni senza consumare risorse di calcolo.

Automatizza i flussi di lavoro aziendali in più fasi: crea flussi di lavoro affidabili per l'onboarding dei dipendenti, l'approvazione dei prestiti e i processi di conformità che durano giorni o settimane. Mantieni lo stato del flusso di lavoro attraverso le approvazioni umane, le integrazioni di sistema e le attività pianificate, fornendo al contempo una visibilità completa sullo stato e sulla cronologia dei processi.

## In che modo le funzioni durevoli si confrontano con Step Functions

Sia le funzioni durevoli che le Step Functions forniscono l'orchestrazione del flusso di lavoro con una gestione automatica dello stato. Le principali differenze sono il luogo in cui vengono eseguiti e il modo in cui si definiscono i flussi di lavoro:

- Funzioni durevoli: eseguite all'interno di Lambda, utilizzano linguaggi di programmazione standard, gestite all'interno dell'ambiente Lambda
- Step Functions: servizio autonomo, DSL o visual designer basato su grafici, completamente gestito senza manutenzione

Le funzioni durevoli sono ideali per lo sviluppo di applicazioni in Lambda, dove i flussi di lavoro sono strettamente associati alla logica aziendale. Step Functions eccelle nell'orchestrazione del flusso di lavoro tra AWS servizi in cui è necessaria una progettazione visiva, integrazioni native con oltre 220 servizi e un'infrastruttura a manutenzione zero.

Per un confronto dettagliato, consulta Durable functions o Step Functions. [???](#)

## Come funziona

Sotto il cofano, le funzioni durevoli sono normali funzioni Lambda che utilizzano un checkpoint/replay meccanismo per tracciare i progressi e supportare operazioni di lunga durata attraverso punti di sospensione definiti dall'utente, comunemente denominati esecuzione duratura. Dopo la ripresa della funzione dopo una pausa o un'interruzione, il sistema esegue la riproduzione. Durante

la riproduzione, il codice viene eseguito dall'inizio ma salta i checkpoint completati, utilizzando i risultati memorizzati invece di rieseguire le operazioni completate. Questo meccanismo di ripetizione garantisce la coerenza e consente esecuzioni di lunga durata.

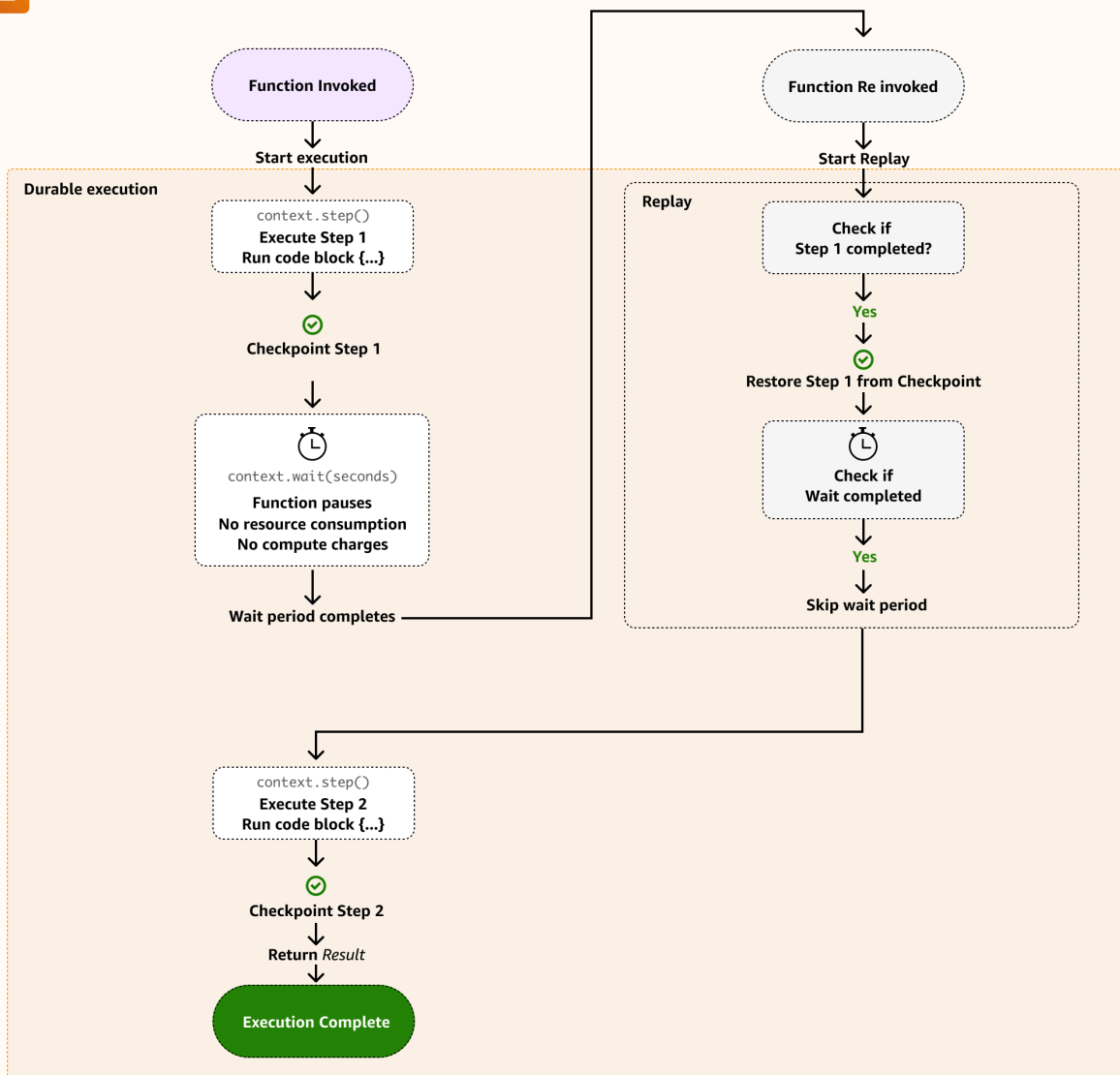
Per sfruttare questo meccanismo di checkpoint e replay nelle tue applicazioni, Lambda fornisce un SDK di esecuzione durevole. L'SDK elimina la complessità della gestione dei checkpoint e della riproduzione, esponendo semplici primitive chiamate operazioni durevoli che utilizzi nel codice. L'SDK è disponibile per Python e Java e si JavaScript integra TypeScript perfettamente con il flusso di lavoro di sviluppo Lambda esistente.

Con l'SDK, puoi incorporare il tuo gestore di eventi Lambda, che a sua volta fornisce un supporto all'evento. `DurableContext` Questo contesto consente di accedere a operazioni durature come passaggi e attese. La logica delle funzioni viene scritta come un normale codice sequenziale, ma invece di chiamare direttamente i servizi, si raggruppano le chiamate in fasi che consentono il checkpoint e i tentativi automatici. Quando è necessario sospendere l'esecuzione, si aggiungono delle attese che sospendono la funzione senza incorrere in addebiti. L'SDK gestisce tutta la complessità della gestione dello stato e della riproduzione dietro le quinte, in modo che il codice rimanga pulito e leggibile.

## Your application



## AWS Lambda durable function



## Fasi successive

- [AWS Guida per gli sviluppatori di Durable Execution SDK](#): guide di riferimento, guide introduttive, di test e linguistiche all'SDK
- [Inizia con funzioni durevoli](#)
- [Funzioni durevoli o Step Functions](#)



- [Monitora ed esegui il debug delle funzioni durature](#)
- [Rivedi la sicurezza e le autorizzazioni](#)
- [Crittografia AWS Lambda dati di esecuzione durevoli](#)
- [Segui le migliori pratiche](#)

## Concetti di base

Il AWS Durable Execution SDK è la base per la creazione di funzioni durature, fornendo le primitive necessarie per verificare l'avanzamento, gestire i tentativi e gestire il flusso di esecuzione. L'SDK è disponibile per JavaScript/TypeScript Python e Java. Per la documentazione completa dell'SDK, che include riferimenti alle API, tutorial di avvio rapido e guide specifiche per il linguaggio, consulta la Durable Execution SDK Developer Guide. [AWS](#)

### Esecuzione duratura

Un'esecuzione duratura rappresenta l'intero ciclo di vita di una funzione durevole Lambda, che utilizza un meccanismo di checkpoint e replay per tracciare l'avanzamento della logica aziendale, sospendere l'esecuzione e ripristinare i dati in caso di guasti. Quando le funzioni riprendono dopo una sospensione o un'interruzione, i checkpoint completati in precedenza vengono ripetuti e l'esecuzione della funzione continua.

Il ciclo di vita potrebbe includere più invocazioni di una funzione Lambda per completare l'esecuzione, in particolare dopo sospensioni o ripristino in caso di guasto. Questo approccio consente alla funzione di funzionare per periodi prolungati (fino a un anno) mantenendo un progresso affidabile nonostante le interruzioni.

#### Come funziona il replay

Lambda mantiene un registro di tutte le operazioni durature (passaggi, attese e altre operazioni) durante l'esecuzione della funzione. Quando la funzione deve essere messa in pausa o subisce un'interruzione, Lambda salva questo registro dei checkpoint e ne interrompe l'esecuzione. Quando è il momento di riprendere, Lambda richiama nuovamente la funzione dall'inizio e riproduce il registro dei checkpoint, sostituendo i valori memorizzati per le operazioni completate. Ciò significa che il codice viene eseguito nuovamente, ma i passaggi completati in precedenza non vengono rieseguiti. Al loro posto vengono utilizzati i risultati memorizzati.

Questo meccanismo di riproduzione è fondamentale per comprendere le funzioni durature. Il codice deve essere deterministico durante la riproduzione, vale a dire che produce gli stessi risultati con gli

stessi input. Evita le operazioni con effetti collaterali (come generare numeri casuali o ottenere l'ora corrente) al di fuori delle fasi, poiché queste possono produrre valori diversi durante la riproduzione e causare comportamenti non deterministici.

## DurableContext

DurableContext è l'oggetto di contesto che riceve la tua funzione durevole. Fornisce metodi per operazioni durature come passaggi e attese che creano punti di controllo e gestiscono il flusso di esecuzione.

La tua funzione durevole riceve un contesto Lambda DurableContext anziché predefinito:

### TypeScript

```
import {
  DurableContext,
  withDurableExecution,
} from "@aws/durable-execution-sdk-js";

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const result = await context.step(async () => {
      return "step completed";
    });
    return result;
  },
);
```

### Python

```
from aws_durable_execution_sdk_python import (
    DurableContext,
    durable_execution,
    durable_step,
)

@durable_step
def my_step(step_context, data):
    # Your business logic
    return result

@durable_execution
```

```
def handler(event, context: DurableContext):  
    result = context.step(my_step(event["data"]))  
    return result
```

Tramite `DurableContext`, la tua funzione ha accesso a operazioni durature che controllano l'avanzamento e coordinano il lavoro:

- `Steps` esegue la logica aziendale con tentativi integrati e checkpoint automatico. Ogni passaggio salva il risultato, in modo che la funzione riprenda dall'ultimo passaggio completato dopo un'interruzione.
- `Le attese` sono pause pianificate in cui la funzione smette di funzionare e interrompe la ricarica fino al momento di continuare. Usali per periodi di tempo, callback esterni o sondaggi per una condizione.
- `Invoke` chiama altre funzioni Lambda e controlla il risultato. La funzione richiamata può essere una funzione standard o duratura. Cross-account le invocazioni non sono supportate.

Per il riferimento linguistico completo a tutte le operazioni durevoli disponibili, consulta [SDK Reference](#) nella AWS Durable Execution SDK Developer Guide.

## Configurazione delle funzioni durevoli

Le funzioni durevoli hanno impostazioni di configurazione specifiche che controllano il comportamento di esecuzione e la conservazione dei dati. Queste impostazioni sono separate dalla configurazione standard delle funzioni Lambda e si applicano all'intero ciclo di vita dell'esecuzione durevole.

L'`DurableConfig` oggetto definisce la configurazione per le funzioni durevoli:

```
{  
  "ExecutionTimeout": Integer,  
  "RetentionPeriodInDays": Integer  
}
```

### Execution timeout (Timeout di esecuzione)

Il timeout di esecuzione controlla per quanto tempo può durare un'esecuzione duratura dall'inizio al completamento. Questo è diverso dal timeout della funzione Lambda, che controlla per quanto tempo può essere eseguita una singola chiamata di funzione.

Un'esecuzione duratura può includere più invocazioni di funzioni Lambda man mano che procede attraverso checkpoint, attese e replay. Il timeout di esecuzione si applica al tempo totale trascorso dell'esecuzione duratura, non alle chiamate di singole funzioni.

## Comprendere la differenza

Il timeout della funzione Lambda (massimo 15 minuti) limita ogni singola chiamata della funzione. Il timeout di esecuzione duratura (massimo 1 anno) limita il tempo totale dall'inizio dell'esecuzione fino al completamento, al fallimento o al timeout. Durante questo periodo, la funzione potrebbe essere richiamata più volte durante l'elaborazione dei passaggi, le attese e il ripristino in caso di errori.

Ad esempio, se imposti un timeout di esecuzione duratura di 24 ore e un timeout per una funzione Lambda di 5 minuti:

- Ogni chiamata di funzione deve essere completata entro 5 minuti
- L'intera esecuzione duratura può durare fino a 24 ore
- La tua funzione può essere richiamata molte volte durante queste 24 ore
- Le operazioni di attesa non vengono conteggiate nel timeout della funzione Lambda ma nel timeout di esecuzione

È possibile configurare il timeout di esecuzione quando si crea una funzione durevole utilizzando la console Lambda, o. AWS CLI AWS SAM Nella console Lambda, scegli la tua funzione, quindi Configurazione, Esecuzione durevole. Imposta il valore del timeout di esecuzione in secondi (predefinito: 86400 secondi/24 ore, minimo: 60 secondi, massimo: 31536000 secondi/1 anno).

### Note

Il timeout di esecuzione e il timeout della funzione Lambda sono impostazioni diverse. Il timeout della funzione Lambda controlla la durata di ogni singola chiamata (massimo 15 minuti). Il timeout di esecuzione controlla il tempo totale trascorso per l'intera esecuzione duratura (massimo 1 anno).

## Periodo di conservazione

Il periodo di conservazione controlla per quanto tempo Lambda conserva la cronologia delle esecuzioni e i dati dei checkpoint dopo il completamento di un'esecuzione duratura. Questi dati includono i risultati delle fasi, lo stato di esecuzione e il registro completo dei checkpoint.

Alla scadenza del periodo di conservazione, Lambda elimina la cronologia delle esecuzioni e i dati dei checkpoint. Non è più possibile recuperare i dettagli dell'esecuzione o ripetere l'esecuzione. Il periodo di conservazione inizia quando l'esecuzione raggiunge lo stato terminale (SUCCEEDED, FAILED, STOPPED o TIMED\_OUT).

È possibile configurare il periodo di conservazione durante la creazione di una funzione durevole utilizzando la console Lambda, o. AWS CLI AWS SAM Nella console Lambda, scegli la tua funzione, quindi Configurazione, Esecuzione durevole. Imposta il valore del periodo di conservazione in giorni (predefinito: 14 giorni, minimo: 1 giorno, massimo: 90 giorni).

Scegli un periodo di conservazione in base ai requisiti di conformità, alle esigenze di debug e alle considerazioni sui costi. Periodi di conservazione più lunghi offrono più tempo per il debug e l'audit, ma aumentano i costi di storage.

## Consulta anche

- [AWS Guida per gli sviluppatori di Durable Execution SDK](#): riferimenti completi all'SDK, tutorial di avvio rapido, framework di test e guide specifiche per la lingua.
- [Funzioni durevoli o Step Functions](#): confronta le funzioni durevoli con Step Functions per capire quando ogni approccio è più efficace.

## Creazione di funzioni durevoli Lambda

Per iniziare a usare le funzioni Lambda durable, usa la console Lambda per creare una funzione durevole. In pochi minuti, puoi creare e implementare una funzione durevole che utilizza passaggi e attese per dimostrare l'esecuzione basata su checkpoint.

Puoi anche creare funzioni durevoli utilizzando [AWS CLI](#) or [infrastructure as code \(IaC\)](#).

Man mano che svolgerai il tutorial, imparerai i concetti fondamentali delle funzioni durevoli, come usare l'`DurableContext` oggetto, creare checkpoint con passaggi e sospendere l'esecuzione con attese. Scoprirai anche come funziona la riproduzione quando la funzione riprende dopo un'attesa.

Per semplificare le cose, questo tutorial ti mostra come creare la tua funzione usando Python o il runtime. Node.js Con questi linguaggi interpretati, puoi modificare il codice della funzione direttamente nell'editor del codice integrato della console.

### Note

Le funzioni Durable attualmente supportano i runtime Python, Node.js (JavaScript/TypeScript) e Java, nonché le immagini dei contenitori (OCI). Per un elenco completo delle versioni di runtime supportate e delle opzioni di immagine del contenitore, vedi Runtime [supportati per funzioni durevoli](#). Per ulteriori informazioni sull'utilizzo delle immagini dei container con Lambda, vedi [Creazione di immagini di container Lambda](#) nella Lambda Developer Guide.

### Tip

Per scoprire come creare soluzioni serverless, consulta la [Guida allo sviluppo serverless](#).

## Crea una funzione durevole Lambda con la console

In questo esempio, la tua funzione durevole elabora un ordine in più passaggi con checkpoint automatico. La funzione accetta un oggetto JSON contenente un ID dell'ordine, convalida l'ordine, elabora il pagamento e conferma l'ordine. Ogni passaggio viene controllato automaticamente, quindi se la funzione viene interrotta, riprende dall'ultimo passaggio completato.

La funzione dimostra anche un'operazione di attesa, sospendendo l'esecuzione per un breve periodo per simulare l'attesa di una conferma esterna.

Per creare una funzione duratura con la console

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli Crea funzione.
3. Scegli Crea da zero.
4. Nel riquadro Informazioni base, per Nome funzione inserisci **myDurableFunction**.
5. Per Runtime, scegli Node.js 24 o Python 3.14.
6. Seleziona Abilita esecuzione duratura.

Lambda crea la tua funzione durevole con un ruolo di [esecuzione](#) che include le autorizzazioni per le operazioni sui checkpoint (`lambda:CheckpointDurableExecution`).  
`lambda:GetDurableExecutionState`

## Note

I runtime Lambda includono il Durable Execution SDK, in modo da poter testare le funzioni durevoli senza dipendenze tra i pacchetti. Tuttavia, consigliamo di includere l'SDK nel pacchetto di distribuzione per la produzione. Ciò garantisce la coerenza delle versioni ed evita potenziali aggiornamenti di runtime che potrebbero influire sulla tua funzione.

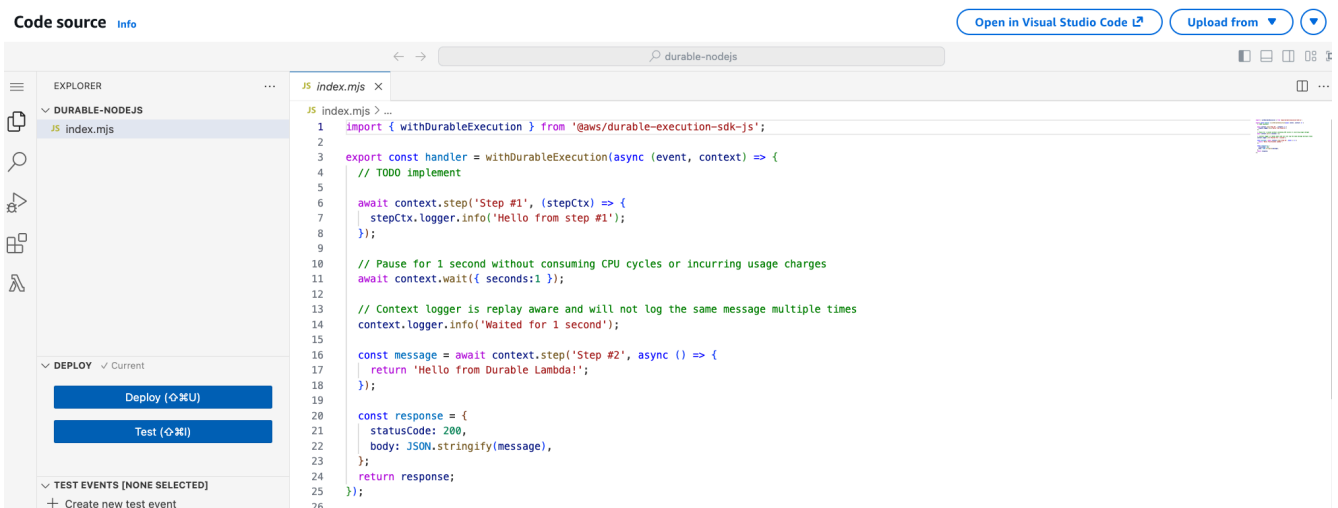
Usa l'editor di codice integrato nella console per aggiungere il tuo codice funzionale durevole.

## Node.js

Per modificare il codice nella console

1. Scegli la scheda Codice.

Nell'editor di codice integrato della console, dovresti vedere il codice della funzione creato da Lambda. Se non vedi la scheda `index.js` nell'editor di codice, seleziona `index.js` in Esplora file, come illustrato nel diagramma seguente.



2. Incolla il codice seguente nella scheda `index.js`, sostituendo il codice creato da Lambda.

```
import {
  withDurableExecution,
} from "@aws/durable-execution-sdk-js";

export const handler = withDurableExecution(
  async (event, context) => {
    const orderId = event.orderId;
```

```
// Step 1: Validate order
const validationResult = await context.step(async (stepContext) => {
  stepContext.logger.info(`Validating order ${orderId}`);
  return { orderId, status: "validated" };
});

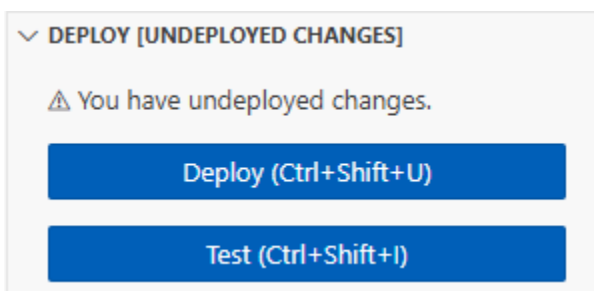
// Step 2: Process payment
const paymentResult = await context.step(async (stepContext) => {
  stepContext.logger.info(`Processing payment for order ${orderId}`);
  return { orderId, status: "paid", amount: 99.99 };
});

// Wait for 10 seconds to simulate external confirmation
await context.wait({ seconds: 10 });

// Step 3: Confirm order
const confirmationResult = await context.step(async (stepContext) => {
  stepContext.logger.info(`Confirming order ${orderId}`);
  return { orderId, status: "confirmed" };
});

return {
  orderId: orderId,
  status: "completed",
  steps: [validationResult, paymentResult, confirmationResult]
};
};
```

3. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Comprendere il codice della funzione durevole



Prima di passare alla fase successiva, esaminiamo il codice funzione e comprendiamo i concetti chiave delle funzioni durevoli.

- L'`withDurableExecution` involucro:

La tua funzione duratura è confezionata con `withDurableExecution`. Questo wrapper consente un'esecuzione duratura fornendo l'`DurableContext` oggetto e gestendo le operazioni sui checkpoint.

- L'oggetto: `DurableContext`

Invece del contesto Lambda standard, la tua funzione riceve un `DurableContext`. Questo oggetto fornisce metodi per operazioni durature come `step()` e `wait()` che creano checkpoint.

- Passaggi e punti di controllo:

Ogni `context.step()` chiamata crea un checkpoint prima e dopo l'esecuzione. Se la funzione viene interrotta, riprende dall'ultimo checkpoint completato. La funzione non riesegue i passaggi completati. Utilizza invece i risultati memorizzati.

- Operazioni di attesa:

La `context.wait()` chiamata sospende l'esecuzione senza consumare risorse di calcolo. Al termine dell'attesa, Lambda richiama nuovamente la funzione e riproduce il registro del checkpoint, sostituendo i valori memorizzati con i passaggi completati.

- Meccanismo di replay:

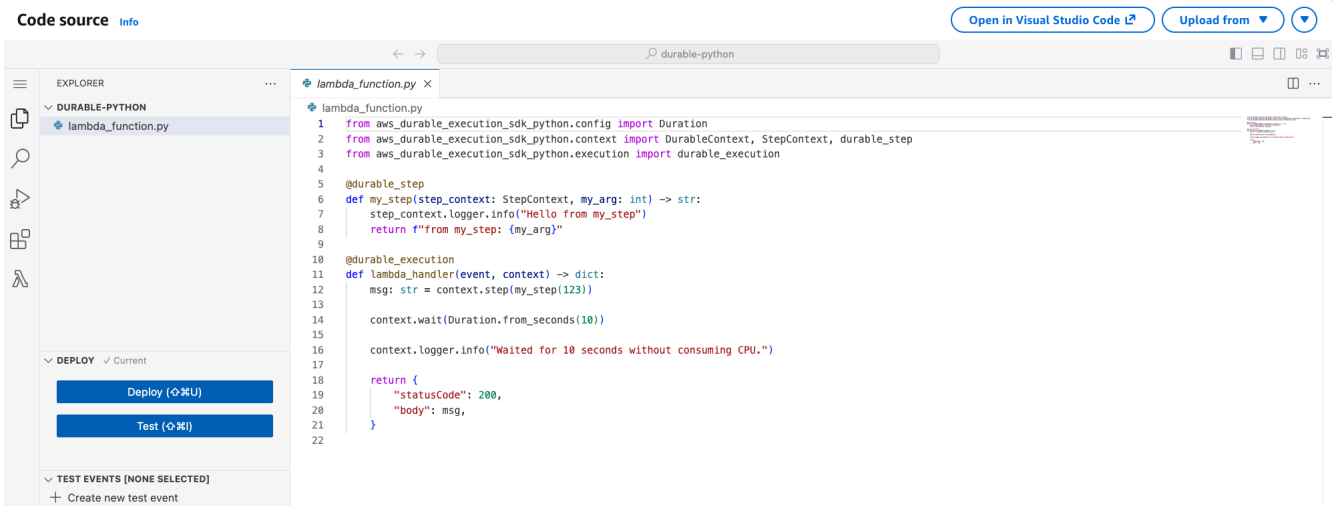
Quando la funzione riprende dopo un'attesa o un'interruzione, Lambda esegue il codice dall'inizio. Tuttavia, i passaggi completati non vengono rieseguiti. Lambda riproduce i risultati dal registro dei checkpoint. Ecco perché il codice deve essere deterministico.

## Python

Per modificare il codice nella console

1. Scegli la scheda Codice.

Nell'editor di codice integrato della console, dovresti vedere il codice della funzione creato da Lambda. Se non vedi la scheda `lambda_function.py` nell'editor di codice, seleziona `lambda_function.py` in Esplora file, come illustrato nel diagramma seguente.



2. Incolla il codice seguente nella scheda `lambda_function.py`, sostituendo il codice creato da Lambda.

```

from aws_durable_execution_sdk_python import (
    DurableContext,
    durable_execution,
    durable_step,
)
from aws_durable_execution_sdk_python.config import Duration

@durable_step
def validate_order(step_context, order_id):
    step_context.logger.info(f"Validating order {order_id}")
    return {"orderId": order_id, "status": "validated"}

@durable_step
def process_payment(step_context, order_id):
    step_context.logger.info(f"Processing payment for order {order_id}")
    return {"orderId": order_id, "status": "paid", "amount": 99.99}

@durable_step
def confirm_order(step_context, order_id):
    step_context.logger.info(f"Confirming order {order_id}")
    return {"orderId": order_id, "status": "confirmed"}

@durable_execution
def lambda_handler(event, context: DurableContext):
    order_id = event['orderId']

    # Step 1: Validate order

```

```
validation_result = context.step(validate_order(order_id))

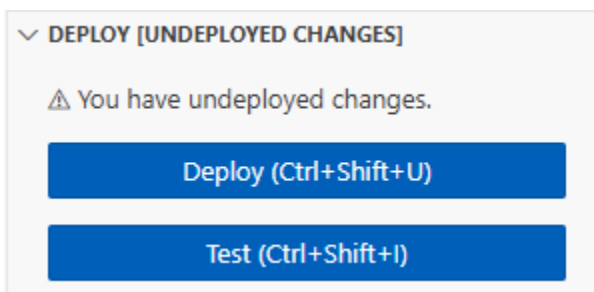
# Step 2: Process payment
payment_result = context.step(process_payment(order_id))

# Wait for 10 seconds to simulate external confirmation
context.wait(Duration.from_seconds(10))

# Step 3: Confirm order
confirmation_result = context.step(confirm_order(order_id))

return {
    "orderId": order_id,
    "status": "completed",
    "steps": [validation_result, payment_result, confirmation_result]
}
```

3. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Comprendere il codice della funzione durevole

Prima di passare alla fase successiva, esaminiamo il codice funzione e comprendiamo i concetti chiave delle funzioni durevoli.

- Il `@durable_execution` decoratore:

La tua funzione di gestione è decorata con `@durable_execution`. Questo decoratore consente un'esecuzione duratura fornendo l'`DurableContext` oggetto e gestendo le operazioni sui checkpoint.

- Il `@durable_step` decoratore:

Ogni funzione a gradino è decorata con `@durable_step`. Questo decoratore contrassegna la funzione come un passaggio duraturo che crea punti di controllo.

- L'`DurableContext` oggetto:

Invece del contesto Lambda standard, la tua funzione riceve un `DurableContext`. Questo oggetto fornisce metodi per operazioni durature come `step()` e `wait()` che creano checkpoint.

- Passaggi e punti di controllo:

Ogni `context.step()` chiamata crea un checkpoint prima e dopo l'esecuzione. Se la funzione viene interrotta, riprende dall'ultimo checkpoint completato. La funzione non riesegue i passaggi completati. Utilizza invece i risultati memorizzati.

- Operazioni di attesa:

La `context.wait()` chiamata sospende l'esecuzione senza consumare risorse di calcolo. Al termine dell'attesa, Lambda richiama nuovamente la funzione e riproduce il registro del checkpoint, sostituendo i valori memorizzati con i passaggi completati.

- Python SDK è sincrono:

Nota che Python SDK non utilizza `await`. Tutte le operazioni durevoli sono chiamate di metodo sincrone.

## Richiama la funzione durable utilizzando l'editor di codice della console

Quando non viene specificata (o pubblicata) alcuna versione esplicita, la console richiama la funzione durable utilizzando il `$LATEST` qualificatore di versione. Tuttavia, per l'esecuzione deterministica del codice, è necessario utilizzare sempre un ARN qualificato che punti a una versione stabile.

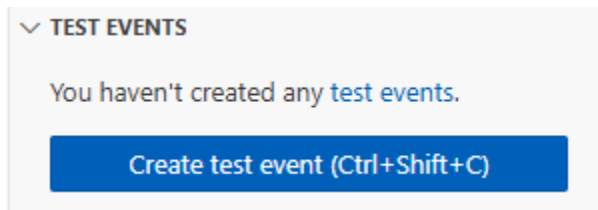
Per pubblicare una versione della tua funzione

1. Scegli la scheda **Versioni**.
2. Selezionare **Publish new version** (Pubblica nuova versione).
3. Per la descrizione della versione, inserisci **Initial version** (opzionale).
4. Seleziona **Pubblica**.
5. Lambda crea la versione 1 della funzione. Nota che la funzione ARN ora include alla fine, `:1` a indicare che si tratta della versione 1.

Ora create un evento di test da inviare alla vostra funzione. L'evento è un documento in formato JSON contenente un ID dell'ordine.

## Per creare un evento di test

1. Nella sezione TEST EVENTS dell'editor di codice della console, scegli Crea evento di test.



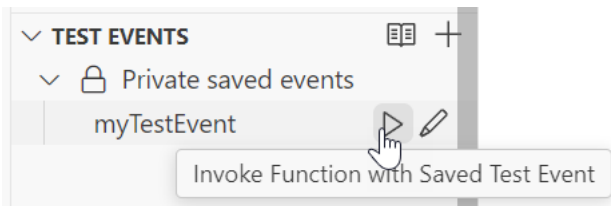
2. Per Event name (Nome evento), immettere **myTestEvent**.
3. Nella sezione JSON dell'evento, sostituisci il codice JSON predefinito con il seguente:

```
{
  "orderId": "order-12345"
}
```

4. Scegli Save (Salva).

## Per testare la tua funzione durevole e visualizzarne l'esecuzione

Nella sezione TEST EVENTS dell'editor di codice della console, scegli l'icona di esecuzione accanto all'evento di test:



La tua funzione durevole inizia l'esecuzione. Poiché include un'attesa di 10 secondi, la chiamata iniziale viene completata rapidamente e la funzione riprende dopo il periodo di attesa. È possibile visualizzare l'avanzamento dell'esecuzione nella scheda Esecuzioni durature.

## Per visualizzare l'esecuzione della funzione durevole

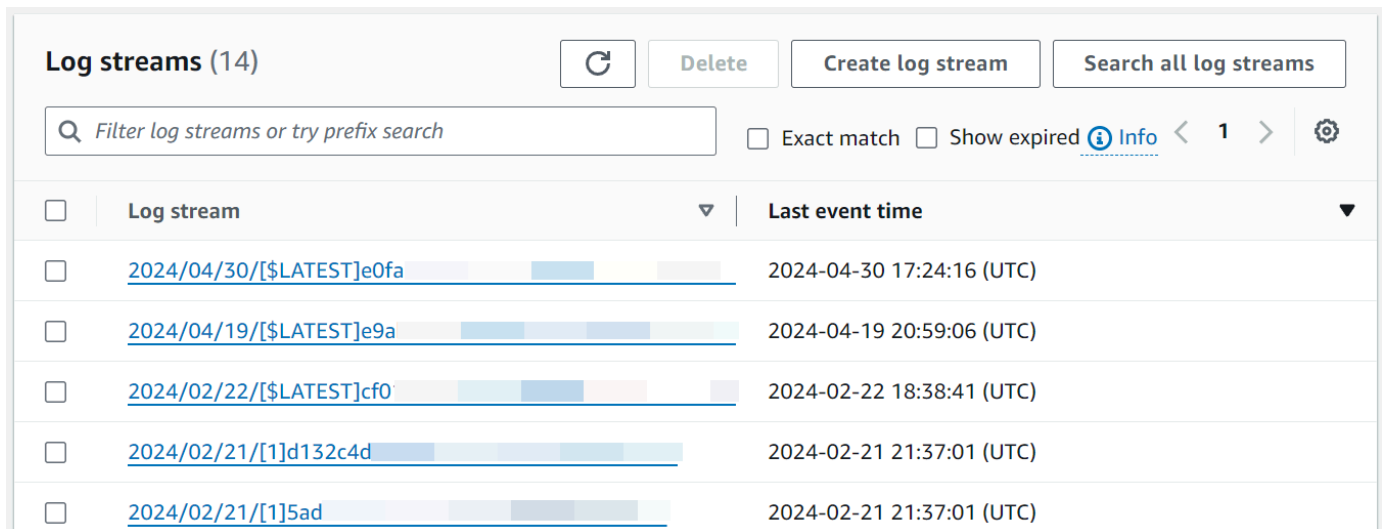
1. Scegli la scheda Esecuzioni durevoli.
2. Trova la tua esecuzione nell'elenco. L'esecuzione mostra lo stato corrente (In esecuzione, Riuscito o Non riuscito).
3. Scegli l'ID di esecuzione per visualizzare i dettagli, tra cui:
  - Cronologia di esecuzione che mostra quando ogni passaggio è stato completato

- Cronologia dei checkpoint
- Periodi di attesa
- Risultati delle fasi

Puoi anche visualizzare i log della tua funzione in CloudWatch Logs per vedere l'output della console di ogni passaggio.

Per visualizzare i record di chiamata della funzione in Logs CloudWatch

1. Apri la [pagina dei gruppi di log](#) della CloudWatch console.
2. Scegli il nome del gruppo di log per la funzione (/aws/lambda/myDurableFunction).
3. Scorri verso il basso e scegli il flusso di log per le invocazioni delle funzioni che desideri esaminare.



The screenshot shows the 'Log streams' section of the AWS CloudWatch console. At the top, there are buttons for 'Log streams (14)', 'Delete', 'Create log stream', and 'Search all log streams'. Below these is a search bar with the placeholder 'Filter log streams or try prefix search'. To the right of the search bar are checkboxes for 'Exact match' and 'Show expired', followed by an 'Info' icon and navigation arrows. The main part of the screenshot is a table with two columns: 'Log stream' and 'Last event time'. The table lists six log streams, each with a checkbox on the left and a progress bar in the 'Log stream' column. The 'Last event time' column shows the timestamp for each stream.

| <input type="checkbox"/> | Log stream                                | Last event time           |
|--------------------------|-------------------------------------------|---------------------------|
| <input type="checkbox"/> | <a href="#">2024/04/30/[\$LATEST]e0fa</a> | 2024-04-30 17:24:16 (UTC) |
| <input type="checkbox"/> | <a href="#">2024/04/19/[\$LATEST]e9a</a>  | 2024-04-19 20:59:06 (UTC) |
| <input type="checkbox"/> | <a href="#">2024/02/22/[\$LATEST]cf0</a>  | 2024-02-22 18:38:41 (UTC) |
| <input type="checkbox"/> | <a href="#">2024/02/21/[1]d132c4d</a>     | 2024-02-21 21:37:01 (UTC) |
| <input type="checkbox"/> | <a href="#">2024/02/21/[1]5ad</a>         | 2024-02-21 21:37:01 (UTC) |

Dovresti visualizzare le voci di registro per ogni chiamata della tua funzione, inclusa l'esecuzione iniziale e il replay dopo l'attesa.

#### Note

Quando si utilizza il logger da `DurableContext` (ad esempio `context.logger` o `orstepContext.logger`), i log vengono visualizzati anche nelle visualizzazioni Durable Execution e Step della console Lambda. Il caricamento di questi log può richiedere alcuni istanti.

## Eliminazione

Quando hai finito di usare la funzione durable di esempio, cancellala. Puoi anche eliminare il gruppo di log che memorizza i log della funzione e il [ruolo di esecuzione](#) creato dalla console.

Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **confirm** nel campo di immissione del testo, quindi scegli Elimina.

Come eliminare il gruppo di log

1. Apri la pagina [Log groups page \(Pagina dei gruppi di log\)](#) della console CloudWatch.
2. Scegli il gruppo di log della funzione (/aws/lambda/myDurableFunction).
3. Scegli Actions (Azioni), Delete log group(s) (Elimina gruppo/i di log).
4. Nella finestra di dialogo Delete log group(s) (Elimina gruppo/i di log) scegli Delete (Elimina).

Come eliminare il ruolo di esecuzione

1. Apri la pagina [Ruoli](#) della console AWS Identity and Access Management (IAM).
2. Seleziona il ruolo di esecuzione della funzione (ad esempio, myDurableFunction-role-*31exxmpl*).
3. Scegli Elimina.
4. Nella finestra di dialogo Elimina ruolo, immetti il nome del ruolo, quindi scegli Elimina.

## Risorse aggiuntive e fasi successive

Ora che hai creato e testato una funzione semplice e duratura utilizzando la console, procedi nel seguente modo:

- Scopri i casi d'uso più comuni delle funzioni durevoli, tra cui transazioni distribuite, elaborazione degli ordini e flussi di lavoro di revisione umana. Vedi [esempi](#).
- Scopri come monitorare le esecuzioni durature delle funzioni con CloudWatch metriche e cronologia delle esecuzioni. Vedi [Monitoraggio e debug](#).

- Scopri come richiamare funzioni durevoli in modo sincrono e asincrono e come gestire le esecuzioni di lunga durata. [Vedi Richiamo di](#) funzioni durevoli.
- Segui le best practice per scrivere codice deterministico, gestire le dimensioni dei checkpoint e ottimizzare i costi. Consulta le best practice.
- Scopri come testare funzioni durevoli localmente e nel cloud. Vedi [Test delle funzioni durevoli](#).
- Confronta le funzioni durevoli con Step Functions per capire quando ogni approccio è più efficace. Vedi Funzioni [durevoli o Funzioni Step](#).

## Implementa e richiama le funzioni durevoli Lambda con AWS CLI

Usa il AWS CLI per creare e distribuire funzioni durevoli Lambda con comandi imperativi. Questo approccio ti offre il controllo diretto su ogni fase del processo di distribuzione.

### Prerequisiti

- Installa e configura la AWS CLI. Per istruzioni, vedere [Installazione di AWS CLI](#).
- Crea un pacchetto di distribuzione con il codice della funzione e l'SDK di esecuzione durevole.
- Crea un ruolo di esecuzione IAM con autorizzazioni di checkpoint.

### Creazione del ruolo di esecuzione

Crea un ruolo IAM con autorizzazioni per l'esecuzione Lambda di base e le operazioni di checkpoint.

### Per creare il ruolo di esecuzione

1. Crea un documento sulla politica di fiducia che consenta a Lambda di assumere il ruolo. Salva questo `cometrust-policy.json`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```



```
]
}
```

## 2. Crea il ruolo:

```
aws iam create-role \
  --role-name durable-function-role \
  --assume-role-policy-document file://trust-policy.json
```

## 3. Allega la politica di esecuzione duratura per le operazioni di checkpoint e l'esecuzione di base:

```
aws iam attach-role-policy \
  --role-name durable-function-role \
  --policy-arn arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicDurableExecutionRolePolicy
```

La policy `AWSLambdaBasicDurableExecutionRolePolicy` gestita include le autorizzazioni necessarie per le operazioni di checkpoint (`lambda:CheckpointDurableExecution` e `lambda:GetDurableExecutionState`) e l'esecuzione di base di Lambda.

## Crea la funzione duratura

Crea la tua funzione duratura con il `--durable-config` parametro.

Per creare una funzione duratura

## 1. Package del codice della funzione con le dipendenze in un file.zip:

```
zip -r function.zip index.mjs node_modules/
```

### Note

Per le funzioni durevoli basate su Java, è necessario compilare il codice della funzione e le dipendenze in un unico file.zip o file Java Archive (JAR). Per ulteriori informazioni, consulta [Distribuire le funzioni Java Lambda con archivi di file.zip o JAR](#).

## 2. Crea la funzione con l'esecuzione duratura abilitata:

```
aws lambda create-function \
```

```
--function-name myDurableFunction \  
--runtime nodejs22.x \  
--role arn:aws:iam::123456789012:role/durable-function-role \  
--handler index.handler \  
--zip-file fileb://function.zip \  
--durable-config '{"ExecutionTimeout": 3600, "RetentionPeriodInDays": 7}'
```

### Note

È possibile abilitare l'esecuzione duratura solo durante la creazione della funzione. Non è possibile abilitarla su funzioni esistenti.

## Pubblica una versione

Sebbene le funzioni durevoli possano essere richiamate utilizzando il qualificatore di \$LATEST versione, è necessario utilizzare sempre un ARN qualificato che punti a una versione stabile per garantire l'esecuzione deterministica del codice.

```
aws lambda publish-version \  
--function-name myDurableFunction \  
--description "Initial version"
```

Il comando restituisce la versione ARN. Annotate il numero di versione (ad esempio, :1) alla fine dell'ARN.

Facoltativamente, create un alias che punti alla versione:

```
aws lambda create-alias \  
--function-name myDurableFunction \  
--name prod \  
--function-version 1
```

## Invoca la funzione duratura

Richiama la tua funzione duratura utilizzando l'ARN qualificato (versione o alias).

 Note

Invocazioni idempotenti: per evitare esecuzioni duplicate quando si riprova una chiamata fallita, è possibile fornire un nome di esecuzione che garantisca la semantica dell'esecuzione. at-most-once [Vedi Idempotenza per i dettagli.](#)

## Invocazione sincrona

Per le esecuzioni che vengono completate entro 15 minuti, usa la chiamata sincrona:

```
aws lambda invoke \  
  --function-name myDurableFunction:1 \  
  --payload '{"orderId": "order-12345"}' \  
  --cli-binary-format raw-in-base64-out \  
  response.json
```

O usando un alias:

```
aws lambda invoke \  
  --function-name myDurableFunction:prod \  
  --payload '{"orderId": "order-12345"}' \  
  --cli-binary-format raw-in-base64-out \  
  response.json
```

## Invocazione asincrona

Per le esecuzioni di lunga durata, usa la chiamata asincrona:

```
aws lambda invoke \  
  --function-name myDurableFunction:prod \  
  --invocation-type Event \  
  --payload '{"orderId": "order-12345"}' \  
  --cli-binary-format raw-in-base64-out \  
  response.json
```

Con la chiamata asincrona, Lambda ritorna immediatamente. L'esecuzione della funzione continua in background.

 Note

È possibile utilizzarla `$LATEST` per la prototipazione e il test nella console. Per i carichi di lavoro di produzione, utilizza una versione o un alias pubblicati.

## Gestisci esecuzioni durevoli

Utilizzate i seguenti comandi per gestire e monitorare le esecuzioni di funzioni durevoli.

### Elenca le esecuzioni

Elenca tutte le esecuzioni per una funzione durevole:

```
aws lambda list-durable-executions-by-function \  
  --function-name myDurableFunction
```

### Ottieni i dettagli dell'esecuzione

Ottieni dettagli su un'esecuzione specifica:

```
aws lambda get-durable-execution \  
  --durable-execution-arn arn:aws:lambda:us-east-1:123456789012:function:myDurableFunction:my-function-version/durable-execution/my-execution-name/my-execution-id
```

### Ottieni la cronologia delle esecuzioni

Visualizza la cronologia dei checkpoint per un'esecuzione:

```
aws lambda get-durable-execution-history \  
  --durable-execution-arn arn:aws:lambda:us-east-1:123456789012:function:myDurableFunction:my-function-version/durable-execution/my-execution-name/my-execution-id
```

### Interrompi un'esecuzione

Interrompi un'esecuzione durevole in esecuzione:

```
aws lambda stop-durable-execution \  
  --durable-execution-arn arn:aws:lambda:us-east-1:123456789012:function:myDurableFunction:my-function-version/durable-execution/my-execution-name/my-execution-id
```

```
--durable-execution-arn arn:aws:lambda:us-east-1:123456789012:function:myDurableFunction:my-function-version/durable-execution/my-execution-name/my-execution-id
```

## Aggiornamento del codice della funzione

Aggiorna il codice della funzione durevole e pubblica una nuova versione:

Per aggiornare e pubblicare una nuova versione

1. Aggiorna il codice della funzione:

```
aws lambda update-function-code \  
  --function-name myDurableFunction \  
  --zip-file fileb://function.zip
```

2. Attendi il completamento dell'aggiornamento:

```
aws lambda wait function-updated \  
  --function-name myDurableFunction
```

3. Pubblica una nuova versione:

```
aws lambda publish-version \  
  --function-name myDurableFunction \  
  --description "Updated order processing logic"
```

4. Aggiorna l'alias in modo che punti alla nuova versione:

```
aws lambda update-alias \  
  --function-name myDurableFunction \  
  --name prod \  
  --function-version 2
```

### Important

Le esecuzioni in esecuzione continuano a utilizzare la versione con cui sono iniziate. Le nuove chiamate utilizzano la versione aggiornata dell'alias.

## Visualizzare i log delle funzioni

Visualizza i log della tua funzione durevole in Logs: CloudWatch

```
aws logs tail /aws/lambda/myDurableFunction --follow
```

Filtra i log per un'esecuzione specifica:

```
aws logs filter-log-events \  
  --log-group-name /aws/lambda/myDurableFunction \  
  --filter-pattern "exec-abc123"
```

## Eeguire la pulizia delle risorse

Elimina la tua funzione durevole e le risorse associate:

```
# Delete the function  
aws lambda delete-function --function-name myDurableFunction  
  
# Delete the IAM role policies  
aws iam detach-role-policy \  
  --role-name durable-function-role \  
  --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole  
  
aws iam detach-role-policy \  
  --role-name durable-function-role \  
  --policy-arn arn:aws:iam::aws:policy/service-role/  
AWSLambdaBasicDurableExecutionRolePolicy  
  
# Delete the role  
aws iam delete-role --role-name durable-function-role
```

## Fasi successive

Dopo aver implementato la funzione duratura con: AWS CLI

- Monitora le esecuzioni utilizzando i `list-durable-executions-by-function` comandi and `get-durable-execution`
- Visualizza le operazioni dei checkpoint negli eventi relativi ai dati AWS CloudTrail
- Imposta CloudWatch allarmi per errori di esecuzione o esecuzioni di lunga durata
- Automatizza le distribuzioni utilizzando script o pipeline di shell CI/CD

Per ulteriori informazioni sui AWS CLI comandi per Lambda, consulta il [AWS CLI Command Reference](#).

## Implementa funzioni durevoli Lambda con Infrastructure as Code

Puoi implementare le funzioni durevoli Lambda utilizzando strumenti Infrastructure as Code (IaC) come AWS CloudFormation, AWS CDK AWS Serverless Application Model, o Terraform. Questi strumenti consentono di definire la funzione, il ruolo di esecuzione e le autorizzazioni nel codice, rendendo le implementazioni ripetibili e controllate dalla versione.

Tutti e tre gli strumenti richiedono di:

- Consentire un'esecuzione duratura della funzione
- Concedi le autorizzazioni di checkpoint al ruolo di esecuzione
- Pubblica una versione o crea un alias (le funzioni durevoli richiedono ARN qualificati)

Funzioni durevoli da un file ZIP

### AWS CloudFormation

Utilizzate CloudFormation per definire la vostra funzione duratura in un modello. L'esempio seguente crea una funzione durevole con le autorizzazioni richieste.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Lambda durable function example

Resources:
  DurableFunctionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicDurableExecutionRolePolicy

  DurableFunction:
```

```

Type: AWS::Lambda::Function
Properties:
  FunctionName: myDurableFunction
  Runtime: nodejs22.x
  Handler: index.handler
  Role: !GetAtt DurableFunctionRole.Arn
  Code:
    ZipFile: |
      // Your durable function code here
      export const handler = async (event, context) => {
        return { statusCode: 200 };
      };
  DurableConfig:
    ExecutionTimeout: 3600
    RetentionPeriodInDays: 7

```

```

DurableFunctionVersion:
  Type: AWS::Lambda::Version
  Properties:
    FunctionName: !Ref DurableFunction
    Description: Initial version

```

```

DurableFunctionAlias:
  Type: AWS::Lambda::Alias
  Properties:
    FunctionName: !Ref DurableFunction
    FunctionVersion: !GetAtt DurableFunctionVersion.Version
    Name: prod

```

```

Outputs:
  FunctionArn:
    Description: Durable function ARN
    Value: !GetAtt DurableFunction.Arn
  AliasArn:
    Description: Function alias ARN (use this for invocations)
    Value: !Ref DurableFunctionAlias

```

## Per distribuire il modello

```

aws cloudformation deploy \
  --template-file template.yaml \
  --stack-name my-durable-function-stack \
  --capabilities CAPABILITY_IAM

```



## AWS CDK

Con AWS CDK, è possibile definire l'infrastruttura utilizzando i linguaggi di programmazione. Gli esempi seguenti mostrano come creare una funzione duratura usando TypeScript Python.

### TypeScript

```
import * as cdk from 'aws-cdk-lib';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as iam from 'aws-cdk-lib/aws-iam';
import { Construct } from 'constructs';

export class DurableFunctionStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Create the durable function
    const durableFunction = new lambda.Function(this, 'DurableFunction', {
      runtime: lambda.Runtime.NODEJS_22_X,
      handler: 'index.handler',
      code: lambda.Code.fromAsset('lambda'),
      functionName: 'myDurableFunction',
      durableConfig: { executionTimeout: Duration.hours(1), retentionPeriod:
Duration.days(30) },
    });

    // Create version and alias
    const version = durableFunction.currentVersion;
    const alias = new lambda.Alias(this, 'ProdAlias', {
      aliasName: 'prod',
      version: version,
    });

    // Output the alias ARN
    new cdk.CfnOutput(this, 'FunctionAliasArn', {
      value: alias.functionArn,
      description: 'Use this ARN to invoke the durable function',
    });
  }
}
```

## Python

```
from aws_cdk import (
    Stack,
    aws_lambda as lambda_,
    aws_iam as iam,
    CfnOutput,
)
from constructs import Construct

class DurableFunctionStack(Stack):
    def __init__(self, scope: Construct, id: str, **kwargs):
        super().__init__(scope, id, **kwargs)

        # Create the durable function
        durable_function = lambda_.Function(
            self, 'DurableFunction',
            runtime=lambda_.Runtime.NODEJS_22_X,
            handler='index.handler',
            code=lambda_.Code.from_asset('lambda'),
            function_name='myDurableFunction',
            durable_execution={execution_timeout: Duration.hours(1),
retention_period: Duration.days(30)}
        )

        # Add durable execution managed policy for checkpoint permissions
        durable_function.role.add_managed_policy(
            iam.ManagedPolicy.from_aws_managed_policy_name('service-role/
AWSLambdaBasicDurableExecutionRolePolicy')
        )

        # Create version and alias
        version = durable_function.current_version
        alias = lambda_.Alias(
            self, 'ProdAlias',
            alias_name='prod',
            version=version
        )

        # Output the alias ARN
        CfnOutput(
            self, 'FunctionAliasArn',
            value=alias.function_arn,
            description='Use this ARN to invoke the durable function'
```

```
)
```

## Per distribuire lo stack CDK

```
cdk deploy
```

## AWS Serverless Application Model

AWS SAM semplifica i CloudFormation modelli per le applicazioni serverless. Il modello seguente crea una funzione duratura con. AWS SAM

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Lambda durable function with SAM

Resources:
  DurableFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: myDurableFunction
      Runtime: nodejs22.x
      Handler: index.handler
      CodeUri: ./src
      DurableConfig:
        ExecutionTimeout: 3600
        RetentionPeriodInDays: 7
      Policies:
        - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicDurableExecutionRolePolicy
      AutoPublishAlias: prod

Outputs:
  FunctionArn:
    Description: Durable function ARN
    Value: !GetAtt DurableFunction.Arn
  AliasArn:
    Description: Function alias ARN (use this for invocations)
    Value: !Ref DurableFunction.Alias
```

## Per distribuire il modello SAM

```
sam build
```

```
sam deploy --guided
```

## Terraform

Terraform è un popolare strumento IaC open source che supporta le risorse AWS. L'esempio seguente crea una funzione duratura con Terraform utilizzando la versione del AWS provider 6.25.0 o successiva.

```
terraform {
  required_version = ">= 1.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 6.25.0"
    }
  }
}

provider "aws" {
  region = "us-east-2"
}

# IAM Role for Lambda Function
resource "aws_iam_role" "lambda_role" {
  name = "durable-function-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Action = "sts:AssumeRole"
      Effect = "Allow"
      Principal = {
        Service = "lambda.amazonaws.com"
      }
    }]
  })
}

# Attach durable execution policy for checkpoint operations
resource "aws_iam_role_policy_attachment" "lambda_durable" {
  policy_arn = "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicDurableExecutionRolePolicy"
  role       = aws_iam_role.lambda_role.name
}
```

```
}

# Lambda Function with Durable Execution enabled
resource "aws_lambda_function" "durable_function" {
  filename      = "function.zip"
  function_name = "myDurableFunction"
  role          = aws_iam_role.lambda_role.arn
  handler       = "index.handler"
  runtime       = "nodejs22.x"
  timeout       = 30
  memory_size   = 512

  durable_config {
    execution_timeout = 900
    retention_period  = 7
  }
}

# Publish a version
resource "aws_lambda_alias" "prod" {
  name           = "prod"
  function_name  = aws_lambda_function.durable_function.function_name
  function_version = aws_lambda_function.durable_function.version
}

output "function_arn" {
  description = "ARN of the Lambda function"
  value       = aws_lambda_function.durable_function.arn
}

output "alias_arn" {
  description = "ARN of the function alias (use this for invocations)"
  value       = aws_lambda_alias.prod.arn
}
```

Da distribuire con Terraform

```
terraform init
terraform plan
terraform apply
```

 Note

Il supporto di Terraform per le funzioni durature Lambda richiede la versione del AWS provider 6.25.0 o successiva. Aggiorna la versione del tuo provider se stai utilizzando una versione precedente.

## Funzioni durevoli da un'immagine del contenitore OCI

È inoltre possibile creare funzioni durevoli basate sulle immagini dei contenitori. Per istruzioni su come creare un'immagine del contenitore, consulta [Runtime supportati per funzioni](#) durevoli.

## AWS CDK

Con AWS CDK, è possibile definire l'infrastruttura utilizzando i linguaggi di programmazione. Gli esempi seguenti mostrano come creare una funzione duratura utilizzando l'immagine TypeScript di un contenitore.

```
import * as cdk from 'aws-cdk-lib';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as iam from 'aws-cdk-lib/aws-iam';
import { Construct } from 'constructs';

export class DurableFunctionStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Create the durable function
    const durableFunction = new lambda.DockerImageFunction(this, 'DurableFunction', {
      code: lambda.DockerImageCode.fromImageAsset('./lambda', {
        platform: cdk.aws_ecr_assets.Platform.LINUX_AMD64,
      }),
      functionName: 'myDurableFunction',
      memorySize: 512,
      timeout: cdk.Duration.seconds(30),
      durableConfig: { executionTimeout: cdk.Duration.hours(1), retentionPeriod:
cdk.Duration.days(30) },
    });

    // Create version and alias
    const version = durableFunction.currentVersion;
    const alias = new lambda.Alias(this, 'ProdAlias', {
```

```

        aliasName: 'prod',
        version: version,
    });

    // Output the alias ARN
    new cdk.CfnOutput(this, 'FunctionAliasArn', {
        value: alias.functionArn,
        description: 'Use this ARN to invoke the durable function',
    });
}
}

```

Per distribuire lo stack CDK

```
cdk deploy
```

## AWS Serverless Application Model

AWS SAM semplifica i CloudFormation modelli per le applicazioni serverless. Il modello seguente crea una funzione duratura con AWS SAM

```

AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Lambda durable function with SAM

Resources:
  DurableFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: myDurableFunction
      PackageType: Image
      ImageUri: ./src
      DurableConfig:
        ExecutionTimeout: 3600
        RetentionPeriodInDays: 7
      Policies:
        - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicDurableExecutionRolePolicy
      AutoPublishAlias: prod
    Metadata:
      DockerTag: latest
      DockerContext: ./src
      Dockerfile: Dockerfile

```

**Outputs:****FunctionArn:**

Description: Durable function ARN

Value: !GetAtt DurableFunction.Arn

**AliasArn:**

Description: Function alias ARN (use this for invocations)

Value: !Ref DurableFunction.Alias

## Per distribuire il modello SAM

```
sam build
sam deploy --guided
```

## Modelli di configurazione comuni

Indipendentemente dallo strumento IaC utilizzato, segui questi schemi per funzioni durature:

### Abilita un'esecuzione duratura

Imposta la `DurableConfig` proprietà sulla tua funzione per consentire un'esecuzione duratura. Questa proprietà è disponibile solo durante la creazione della funzione. Non è possibile abilitare l'esecuzione duratura su funzioni esistenti.

### Concedi le autorizzazioni per i checkpoint

`AWSLambdaBasicDurableExecutionRolePolicy` Associa la policy gestita al ruolo di esecuzione. Questa politica include i requisiti `lambda:CheckpointDurableExecution` e `lambda:GetDurableExecutionState` le autorizzazioni.

### Utilizza ARN qualificati

Crea una versione o un alias per la tua funzione. Le funzioni durevoli richiedono ARN qualificati (con versione o alias) per l'invocazione. Usa `AutoPublishAlias` in AWS SAM o crea versioni esplicite in CloudFormation, e Terraform. AWS CDK

### Dipendenze dei pacchetti

Includi il Durable Execution SDK nel tuo pacchetto di distribuzione. Per Node.js, installa. `@aws/durable-execution-sdk-js` Per Python, installa `aws-durable-execution-sdk-python`.



## Fasi successive

Dopo aver implementato la tua funzione durevole:

- Verifica la tua funzione utilizzando l'ARN (versione o alias) qualificato
- Monitora l'avanzamento dell'esecuzione nella console Lambda nella scheda Durable executions
- Visualizza le operazioni sui checkpoint negli eventi relativi ai dati AWS CloudTrail
- Esamina CloudWatch i registri per l'output delle funzioni e il comportamento di riproduzione

Per ulteriori informazioni sull'implementazione delle funzioni Lambda con gli strumenti IaC, consulta:

- [CloudFormation AWS::Lambda: :Riferimento alla funzione](#)
- [AWS CDK Documentazione del modulo Lambda](#)
- [AWS SAM Guida per gli sviluppatori](#)

## Configura le funzioni Lambda durable

Le impostazioni di esecuzione durature controllano per quanto tempo può essere eseguita la AWS Lambda funzione, per quanto tempo il servizio conserva la cronologia delle esecuzioni e quale AWS KMS chiave crittografa i dati di esecuzione. Configura queste impostazioni per consentire un'esecuzione duratura della tua funzione.

### Abilita un'esecuzione duratura

Configura l'`DurableConfig` oggetto durante la creazione della funzione per impostare il timeout di esecuzione e la conservazione della cronologia. È possibile abilitare l'esecuzione duratura solo durante la creazione di una funzione. Non è possibile abilitarlo su funzioni esistenti.

### AWS CLI

```
aws lambda create-function \
  --function-name my-durable-function \
  --runtime nodejs24.x \
  --role arn:aws:iam::123456789012:role/my-durable-role \
  --handler index.handler \
  --zip-file fileb://function.zip \
  --durable-config '{"ExecutionTimeout": 3600, "RetentionPeriodInDays": 30,
  "KMSKeyArn": "arn:aws:kms:us-east-1:111122223333:key/key-id"}'
```

## CloudFormation

```
Resources:
  MyDurableFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: my-durable-function
      Runtime: nodejs24.x
      Handler: index.handler
      Code:
        ZipFile: |
          // Your durable function code
      DurableConfig:
        ExecutionTimeout: 3600
        RetentionPeriodInDays: 30
        KMSKeyArn: "arn:aws:kms:us-east-1:111122223333:key/key-id"
```

### Parametri di configurazione:

- **ExecutionTimeout**: obbligatorio Il tempo massimo in secondi che può essere eseguita un'esecuzione duratura prima che Lambda interrompa l'esecuzione. Questo timeout si applica all'intera esecuzione duratura, non alle singole invocazioni di funzioni. È necessario specificare **ExecutionTimeout** ogni volta che si fornisce **DurableConfig**; la creazione di una funzione con una configurazione duratura ma senza timeout di esecuzione ha esito negativo con un. **InvalidParameterValueException** Intervallo valido: 1—31622400.
- **RetentionPeriodInDays** : Opzionale. Il numero di giorni in cui conservare la cronologia delle esecuzioni dopo il completamento di un'esecuzione duratura. Dopo questo periodo, la cronologia delle esecuzioni non è più disponibile tramite l'**GetDurableExecutionHistoryAPI**. Intervallo valido: 1—90. Predefinito: 14.
- **KMSKeyArn** : Opzionale. L'ARN di una chiave gestita dal cliente per crittografare i dati di esecuzione durevoli. Per ulteriori informazioni, vedere [Crittografia dei dati di esecuzione duratura di Lambda](#).

Per il riferimento completo alle API, consulta [DurableConfig](#) il Lambda API Reference.

## Best practice per la configurazione

Segui queste best practice per configurare funzioni durevoli per l'uso in produzione:

- Imposta i timeout di esecuzione appropriati: configura `ExecutionTimeout` in base alla durata massima prevista del flusso di lavoro. Non impostate timeout inutilmente lunghi in quanto influiscono sui costi e sull'allocazione delle risorse.
- Equilibra la conservazione con i costi di archiviazione: impostali `RetentionPeriodInDays` in base ai requisiti di debug e audit. Periodi di conservazione più lunghi aumentano i costi di storage.
- Dimensioni dello stato del monitor: gli oggetti a stato di grandi dimensioni aumentano i costi di storage e possono influire sulle prestazioni. Mantieni lo stato minimo e utilizza l'archiviazione esterna per dati di grandi dimensioni.
- Configura la registrazione appropriata: abilita la registrazione dettagliata per la risoluzione dei problemi dei flussi di lavoro a lunga durata, ma considera l'impatto sul volume e sui costi dei log.
- Crittografa con la tua AWS KMS chiave: imposta una chiave gestita dal cliente `KMSKeyArn` per controllare la rotazione delle chiavi, verificare l'accesso e soddisfare i requisiti di conformità. Per informazioni, consulta [Crittografia AWS Lambda dati di esecuzione durevoli](#).

Esempio di configurazione di produzione:

```
{
  "ExecutionTimeout": 86400,
  "RetentionPeriodInDays": 7,
  "KMSKeyArn": "arn:aws:kms:us-east-1:111122223333:key/key-id"
}
```

Questo esempio imposta un timeout di esecuzione di 24 ore (86.400 secondi), un periodo di conservazione di 7 giorni e una chiave gestita dal cliente per la crittografia.

## Funzioni durevoli o Step Functions

Entrambe le funzioni di Lambda durano a lungo e AWS Step Functions consentono un'orchestrazione affidabile del flusso di lavoro con gestione automatica dello stato e ripristino degli errori. Soddisfano diverse preferenze e modelli architettonici degli sviluppatori. Le funzioni durevoli sono ottimizzate per lo sviluppo di applicazioni in Lambda, mentre Step Functions è progettato per l'orchestrazione del flusso di lavoro tra i servizi. AWS

## Quando utilizzare le funzioni durevoli

Utilizza funzioni durevoli quando:

- Il tuo team preferisce linguaggi di programmazione standard e strumenti di sviluppo familiari
- La logica dell'applicazione si trova principalmente all'interno delle funzioni Lambda
- Desiderate un controllo granulare sullo stato di esecuzione nel codice
- Stai creando applicazioni incentrate su Lambda con una stretta combinazione tra flusso di lavoro e logica aziendale
- Vuoi iterare rapidamente senza passare da codice a designer e viceversa visual/JSON

## Quando usare Step Functions

Usa Step Functions quando:

- È necessaria una rappresentazione visiva del flusso di lavoro per la visibilità tra i team
- Stai orchestrando più AWS servizi e desideri integrazioni native senza codice SDK personalizzato
- Hai bisogno di un'infrastruttura che non richiede manutenzione (nessuna applicazione di patch, aggiornamenti in fase di esecuzione)
- Le parti interessate non tecniche devono comprendere e convalidare la logica del flusso di lavoro

## Quadro decisionale

Utilizza le seguenti domande per determinare quale servizio si adatta al tuo caso d'uso:

- Qual è il tuo obiettivo principale? Sviluppo di applicazioni in Lambda → funzioni durevoli. Orchestrazione del flusso di lavoro attraverso AWS → Step Functions.
- Qual è il tuo modello di programmazione preferito? Linguaggi di programmazione standard → funzioni durevoli. DSL o visual designer basato su grafici → Step Functions.
- Quanti AWS servizi sono coinvolti? Principalmente Lambda → funzioni durevoli. AWS Servizi multipli → Step Functions.
- Quali strumenti di sviluppo utilizzi? Esperienza per sviluppatori Lambda, IDE con agente LLM, framework di unit test specifici del linguaggio di programmazione,, Toolkit → funzioni durevoli. AWS SAM AWS CDK AWS Generatore di flussi di lavoro visivo, AWS CDK per modellare i flussi di lavoro → Step Functions.

- Chi gestisce l'infrastruttura? Desideri flessibilità all'interno di Lambda → funzioni durevoli. Vuoi una gestione completa e senza manutenzione → Step Functions.

## Confronto delle funzionalità

La tabella seguente confronta le caratteristiche chiave tra le funzioni durevoli Step Functions e Lambda:

| Funzionalità              | AWS Step Functions                                                                                                                       | Funzioni durevoli Lambda                                                                              |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| Obiettivo primario        | Orchestrazione del flusso di lavoro su tutto AWS                                                                                         | Sviluppo di applicazioni in Lambda                                                                    |
| Tipo di servizio          | Servizio di workflow autonomo e dedicato                                                                                                 | Funziona in Lambda                                                                                    |
| Modello di programmazione | DSL basato su grafici, Amazon States Language o AWS CDK                                                                                  | Linguaggi di programmazione standard (JavaScript/TypeScript, Python)                                  |
| Strumenti di sviluppo     | Visual Builder nell'estensione IDE AWS Console/Toolkit, AWS CDK                                                                          | Lambda DX all'interno di agenti IDE e LLM, framework di unit test, estensione IDE Toolkit AWS SAM AWS |
| Integrazioni              | Oltre 220 AWS servizi, 16.000 APIs                                                                                                       | Estensione del modello di programmazione Lambda basato sugli eventi (sorgenti di eventi)              |
| Gestione                  | Completamente gestita, indipendente dal runtime, senza manutenzione (nessuna applicazione di patch, aggiornamenti in fase di esecuzione) | Gestito in ambiente Lambda                                                                            |
| Ideale per                | Automazione dei processi aziendali e IT, elaborazione dei dati, flussi di lavoro AI                                                      | Transazioni distribuite, logica applicativa stateful, orchestrazione                                  |

| Funzionalità | AWS Step Functions | Funzioni durevoli Lambda                                   |
|--------------|--------------------|------------------------------------------------------------|
|              |                    | delle funzioni, elaborazione dei dati, flussi di lavoro AI |

## Architetture ibride

Molte applicazioni traggono vantaggio dall'utilizzo di entrambi i servizi. Un modello comune prevede l'utilizzo di funzioni durevoli per la logica a livello di applicazione all'interno di Lambda, mentre Step Functions coordina i flussi di lavoro di alto livello su più servizi oltre alle funzioni Lambda. AWS

## Considerazioni sulla migrazione

Iniziare da un complesso semplice e in continua evoluzione: inizia con funzioni durevoli per flussi di lavoro incentrati su Lambda. Aggiungi Step Functions quando hai bisogno di un'orchestrazione multiservizio o di una progettazione visiva del flusso di lavoro.

Utenti Step Functions esistenti: mantieni Step Functions per flussi di lavoro cross-service consolidati. Prendi in considerazione funzioni durevoli per la nuova logica applicativa Lambda che richiede affidabilità.

## Risorse correlate

- [Funzioni durevoli Lambda](#)
- [Orchestrazione delle funzioni Lambda con Step Functions](#)
- [Guida introduttiva alle funzioni durevoli](#)

## Esempi e casi d'uso

Le funzioni Lambda durable consentono di creare applicazioni in più fasi e tolleranti ai guasti utilizzando operazioni durevoli come passaggi e attese. Grazie al checkpoint automatico e a un modello checkpoint-replay, in cui l'esecuzione viene riavviata dall'inizio dopo un errore ma salta i checkpoint completati, le funzioni possono riprendersi dai guasti e riprendere l'esecuzione senza perdere i progressi.

 Note

Per il riferimento all'API SDK, i tutorial di avvio rapido, le guide specifiche della lingua e i modelli di flusso di lavoro aggiuntivi, consulta la [Durable Execution SDK Developer Guide](#).

[AWS](#)

## Short-lived processi tolleranti ai guasti

Utilizza funzioni durevoli per creare operazioni affidabili che in genere vengono completate in pochi minuti. Sebbene questi processi siano più brevi rispetto ai flussi di lavoro a lunga durata, beneficiano comunque del checkpoint automatico e della tolleranza ai guasti nei sistemi distribuiti. Le funzioni durevoli aiutano a garantire il corretto completamento dei processi in più fasi anche in caso di esito negativo delle singole chiamate di servizio, senza richiedere una complessa gestione degli errori o un codice di gestione dello stato.

Gli scenari più comuni includono sistemi di prenotazione alberghiera, piattaforme di prenotazione di ristoranti, richieste di viaggi in condivisione delle corse, acquisto di biglietti per eventi e upgrade degli abbonamenti SaaS. Questi scenari hanno caratteristiche comuni: più chiamate di servizio che devono essere completate insieme, la necessità di riprovare automaticamente in caso di guasti transitori e l'esigenza di mantenere uno stato coerente tra i sistemi distribuiti.

### Transazioni distribuite tra microservizi

Coordina i pagamenti, l'inventario e la spedizione tra più servizi con il rollback automatico degli errori. Ogni operazione di servizio è suddivisa in una fase, garantendo che la transazione possa essere ripristinata da qualsiasi momento in caso di guasto di un servizio.

### TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-js";

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { orderId, amount, items } = event;

    // Reserve inventory across multiple warehouses
    const inventory = await context.step("reserve-inventory", async () => {
```

```

        return await inventoryService.reserve(items);
    });

    // Process payment
    const payment = await context.step("process-payment", async () => {
        return await paymentService.charge(amount);
    });

    // Create shipment
    const shipment = await context.step("create-shipment", async () => {
        return await shippingService.createShipment(orderId, inventory);
    });

    return { orderId, status: 'completed', shipment };
}
);

```

## Python

```

from aws_durable_execution_sdk_python import DurableContext, durable_execution

@durable_execution
def lambda_handler(event, context: DurableContext):
    order_id = event['orderId']
    amount = event['amount']
    items = event['items']

    # Reserve inventory across multiple warehouses
    inventory = context.step(
        lambda _: inventory_service.reserve(items),
        name='reserve-inventory'
    )

    # Process payment
    payment = context.step(
        lambda _: payment_service.charge(amount),
        name='process-payment'
    )

    # Create shipment
    shipment = context.step(

```



```

        lambda _: shipping_service.create_shipment(order_id, inventory),
        name='create-shipment'
    )

    return {'orderId': order_id, 'status': 'completed', 'shipment': shipment}

```

Se un passaggio fallisce, la funzione riprova automaticamente dall'ultimo checkpoint riuscito. La prenotazione dell'inventario persiste anche se l'elaborazione del pagamento fallisce temporaneamente. Quando la funzione riprova, salta la fase di inventario completata e procede direttamente all'elaborazione del pagamento. Ciò elimina le prenotazioni duplicate e garantisce uno stato coerente in tutto il sistema distribuito.

### Elaborazione degli ordini con più passaggi

Elabora gli ordini tramite convalida, autorizzazione al pagamento, allocazione dell'inventario ed evasione con riprova e ripristino automatici. Ogni passaggio viene controllato, assicurando che l'ordine proceda anche se i singoli passaggi falliscono e riprovano.

### TypeScript

```

import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-
js";

export const handler = withDurableExecution(
    async (event: any, context: DurableContext) => {
        const { orderId, customerId, items } = event;

        // Validate order details
        const validation = await context.step("validate-order", async () => {
            const customer = await customerService.validate(customerId);
            const itemsValid = await inventoryService.validateItems(items);
            return { customer, itemsValid };
        });

        if (!validation.itemsValid) {
            return { orderId, status: 'rejected', reason: 'invalid_items' };
        }

        // Authorize payment
        const authorization = await context.step("authorize-payment", async () => {

```

```
        return await paymentService.authorize(
            validation.customer.paymentMethod,
            calculateTotal(items)
        );
    });

    // Allocate inventory
    const allocation = await context.step("allocate-inventory", async () => {
        return await inventoryService.allocate(items);
    });

    // Fulfill order
    const fulfillment = await context.step("fulfill-order", async () => {
        return await fulfillmentService.createShipment({
            orderId,
            items: allocation.allocatedItems,
            address: validation.customer.shippingAddress
        });
    });

    return {
        orderId,
        status: 'completed',
        trackingNumber: fulfillment.trackingNumber
    };
}
);
```

## Python

```
from aws_durable_execution_sdk_python import DurableContext, durable_execution

@durable_execution
def lambda_handler(event, context: DurableContext):
    order_id = event['orderId']
    customer_id = event['customerId']
    items = event['items']

    # Validate order details
    def validate_order(_):
        customer = customer_service.validate(customer_id)
```

```
        items_valid = inventory_service.validate_items(items)
        return {'customer': customer, 'itemsValid': items_valid}

    validation = context.step(validate_order, name='validate-order')

    if not validation['itemsValid']:
        return {'orderId': order_id, 'status': 'rejected', 'reason':
'invalid_items'}

    # Authorize payment
    authorization = context.step(
        lambda _: payment_service.authorize(
            validation['customer']['paymentMethod'],
            calculate_total(items)
        ),
        name='authorize-payment'
    )

    # Allocate inventory
    allocation = context.step(
        lambda _: inventory_service.allocate(items),
        name='allocate-inventory'
    )

    # Fulfill order
    fulfillment = context.step(
        lambda _: fulfillment_service.create_shipment({
            'orderId': order_id,
            'items': allocation['allocatedItems'],
            'address': validation['customer']['shippingAddress']
        })),
        name='fulfill-order'
    )

    return {
        'orderId': order_id,
        'status': 'completed',
        'trackingNumber': fulfillment['trackingNumber']
    }
```

Questo schema garantisce che gli ordini non rimangano mai bloccati in stati intermedi. Se la convalida fallisce, l'ordine viene rifiutato prima dell'autorizzazione al pagamento. Se l'autorizzazione al pagamento fallisce, l'inventario non viene allocato. Ogni passaggio si basa su quello precedente con riprova e ripristino automatici.

## Nota

Il controllo condizionale `if (!validation.itemsValid)` è esterno a una fase e viene rieseguito durante il replay. Questo è sicuro perché è deterministico: produce sempre lo stesso risultato con lo stesso oggetto di convalida.

## Long-running processi

Usa funzioni durevoli per processi che durano ore, giorni o settimane. Le operazioni di attesa sospendono l'esecuzione senza incorrere in costi di elaborazione, rendendo i processi a lunga durata convenienti. Durante i periodi di attesa, la funzione smette di funzionare e Lambda ricicla l'ambiente di esecuzione. Quando è il momento di riprendere, Lambda richiama nuovamente la funzione e la riproduce dall'ultimo checkpoint.

Questo modello di esecuzione rende le funzioni durevoli ideali per i processi che devono essere interrotti per lunghi periodi, che si tratti di attendere decisioni umane, risposte di sistema esterne, finestre di elaborazione pianificate o ritardi basati sul tempo. Paghi solo per il tempo di elaborazione attivo, non per l'attesa.

Gli scenari più comuni includono processi di approvazione dei documenti, elaborazione in batch pianificata, processi di onboarding di più giorni, processi di prova degli abbonamenti e sistemi di notifica ritardata. Questi scenari hanno caratteristiche comuni: periodi di attesa prolungati misurati in ore o giorni, necessità di mantenere lo stato di esecuzione durante tali attese e requisiti sensibili ai costi, in cui il pagamento per i tempi di inattività di elaborazione è proibitivo.

## Human-in-the-loop approvazioni

Sospendi l'esecuzione per la revisione, l'approvazione o le decisioni dei documenti mantenendo lo stato di esecuzione. La funzione attende i callback esterni senza consumare risorse e riprende automaticamente quando viene ricevuta l'approvazione.

Questo modello è essenziale per i processi che richiedono il giudizio umano o la convalida esterna. La funzione si sospende al punto di callback, senza incorrere in costi di elaborazione durante l'attesa. Quando qualcuno invia la propria decisione tramite l'API, Lambda richiama nuovamente la funzione e riparte dal checkpoint, continuando con il risultato dell'approvazione.

## TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-js";

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { documentId, reviewers } = event;

    // Step 1: Prepare document for review
    const prepared = await context.step("prepare-document", async () => {
      return await documentService.prepare(documentId);
    });

    // Step 2: Request approval with callback
    const approval = await context.waitForCallback(
      "approval-callback",
      async (callbackId) => {
        await notificationService.sendApprovalRequest({
          documentId,
          reviewers,
          callbackId,
          expiresIn: 86400
        });
      },
      {
        timeout: { seconds: 86400 }
      }
    );

    // Function resumes here when approval is received
    if (approval?.approved) {
      const finalized = await context.step("finalize-document", async () => {
        return await documentService.finalize(documentId, approval.comments);
      });

      return {
        status: 'approved',
        documentId,
        finalizedAt: finalized.timestamp
      };
    }
  }
);
```

```
// Handle rejection
await context.step("archive-rejected", async () => {
    await documentService.archive(documentId, approval?.reason);
});

return {
    status: 'rejected',
    documentId,
    reason: approval?.reason
};
}
);
```

## Python

```
from aws_durable_execution_sdk_python import DurableContext, durable_execution,
WaitConfig

@durable_execution
def lambda_handler(event, context: DurableContext):
    document_id = event['documentId']
    reviewers = event['reviewers']

    # Step 1: Prepare document for review
    prepared = context.step(
        lambda _: document_service.prepare(document_id),
        name='prepare-document'
    )

    # Step 2: Request approval with callback
    def send_approval_request(callback_id):
        notification_service.send_approval_request({
            'documentId': document_id,
            'reviewers': reviewers,
            'callbackId': callback_id,
            'expiresIn': 86400
        })

    approval = context.wait_for_callback(
        send_approval_request,
        name='approval-callback',
```

```

        config=WaitConfig(timeout=86400)
    )

    # Function resumes here when approval is received
    if approval and approval.get('approved'):
        finalized = context.step(
            lambda _: document_service.finalize(document_id,
approval.get('comments')),
            name='finalize-document'
        )

        return {
            'status': 'approved',
            'documentId': document_id,
            'finalizedAt': finalized['timestamp']
        }

    # Handle rejection
    context.step(
        lambda _: document_service.archive(document_id, approval.get('reason') if
approval else None),
        name='archive-rejected'
    )

    return {
        'status': 'rejected',
        'documentId': document_id,
        'reason': approval.get('reason') if approval else None
    }

```

Quando la richiamata viene ricevuta e la funzione viene ripresa, viene riprodotta dall'inizio. La fase di preparazione del documento restituisce immediatamente il risultato indicato. Inoltre, l'ForCallback operazione di attesa restituisce istantaneamente il risultato dell'approvazione memorizzato anziché attendere nuovamente. L'esecuzione prosegue quindi con le fasi di finalizzazione o archiviazione.

### Multi-stage pipeline di dati

Elabora set di dati di grandi dimensioni attraverso fasi di estrazione, trasformazione e caricamento con punti di controllo tra le fasi. Il completamento di ogni fase può richiedere ore e i checkpoint consentono alla pipeline di riprendere da qualsiasi fase in caso di interruzione.

Questo modello è ideale per i flussi di lavoro ETL, le migrazioni dei dati o i processi di elaborazione in batch in cui è necessario elaborare i dati in più fasi con punti di ripristino intermedi. Se una fase fallisce, la pipeline riprende dall'ultima fase completata anziché ricominciare dall'inizio. Puoi anche utilizzare le operazioni di attesa per fare delle pause tra una fase e l'altra, rispettando i limiti di velocità, aspettando che i sistemi a valle siano pronti o programmando l'elaborazione nelle ore non di punta.

## TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-js";

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { datasetId, batchSize } = event;

    // Stage 1: Extract data from source
    const extracted = await context.step("extract-data", async () => {
      const records = await sourceDatabase.extractRecords(datasetId);
      return { recordCount: records.length, records };
    });

    // Wait 5 minutes to respect source system rate limits
    await context.wait({ seconds: 300 });

    // Stage 2: Transform data in batches
    const transformed = await context.step("transform-data", async () => {
      const batches = chunkArray(extracted.records, batchSize);
      const results = [];

      for (const batch of batches) {
        const transformed = await transformService.processBatch(batch);
        results.push(transformed);
      }

      return { batchCount: batches.length, results };
    });

    // Wait until off-peak hours (e.g., 2 AM)
    const now = new Date();
    const targetHour = 2;
    const msUntilTarget = calculateMsUntilHour(now, targetHour);
```



```

    await context.wait({ seconds: Math.floor(msUntilTarget / 1000) });

    // Stage 3: Load data to destination
    const loaded = await context.step("load-data", async () => {
        let loadedCount = 0;

        for (const result of transformed.results) {
            await destinationDatabase.loadBatch(result);
            loadedCount += result.length;
        }

        return { loadedCount };
    });

    // Stage 4: Verify and finalize
    const verified = await context.step("verify-pipeline", async () => {
        const verification = await destinationDatabase.verifyRecords(datasetId);
        await pipelineService.markComplete(datasetId, verification);
        return verification;
    });

    return {
        datasetId,
        recordsProcessed: extracted.recordCount,
        batchesProcessed: transformed.batchCount,
        recordsLoaded: loaded.loadedCount,
        verified: verified.success
    };
}
);

```

## Python

```

from aws_durable_execution_sdk_python import DurableContext, durable_execution
from datetime import datetime

@durable_execution
def lambda_handler(event, context: DurableContext):
    dataset_id = event['datasetId']
    batch_size = event['batchSize']

```

```
# Stage 1: Extract data from source
def extract_data(_):
    records = source_database.extract_records(dataset_id)
    return {'recordCount': len(records), 'records': records}

extracted = context.step(extract_data, name='extract-data')

# Wait 5 minutes to respect source system rate limits
context.wait(Duration.from_seconds(300))

# Stage 2: Transform data in batches
def transform_data(_):
    batches = chunk_array(extracted['records'], batch_size)
    results = []

    for batch in batches:
        transformed = transform_service.process_batch(batch)
        results.append(transformed)

    return {'batchCount': len(batches), 'results': results}

transformed = context.step(transform_data, name='transform-data')

# Wait until off-peak hours (e.g., 2 AM)
now = datetime.now()
target_hour = 2
ms_until_target = calculate_ms_until_hour(now, target_hour)
context.wait(ms_until_target // 1000)

# Stage 3: Load data to destination
def load_data(_):
    loaded_count = 0

    for result in transformed['results']:
        destination_database.load_batch(result)
        loaded_count += len(result)

    return {'loadedCount': loaded_count}

loaded = context.step(load_data, name='load-data')

# Stage 4: Verify and finalize
def verify_pipeline(_):
    verification = destination_database.verify_records(dataset_id)
```

```
        pipeline_service.mark_complete(dataset_id, verification)
        return verification

    verified = context.step(verify_pipeline, name='verify-pipeline')

    return {
        'datasetId': dataset_id,
        'recordsProcessed': extracted['recordCount'],
        'batchesProcessed': transformed['batchCount'],
        'recordsLoaded': loaded['loadedCount'],
        'verified': verified['success']
    }
```

Ogni fase è suddivisa in una fase, creando un checkpoint che consente alla pipeline di riprendere da qualsiasi fase in caso di interruzione. L'attesa di 5 minuti tra l'estrazione e la trasformazione rispetta i limiti di velocità del sistema di origine senza consumare risorse di calcolo, mentre l'attesa fino alle 2 del mattino pianifica le costose operazioni di caricamento nelle ore non di punta.

#### Nota

La `new Date()` chiamata e la `calculateMsUntilHour()` funzione sono esterne e vengono eseguite nuovamente durante la riproduzione. Per le operazioni basate sul tempo che devono essere coerenti tra i replay, calcola il timestamp all'interno di un passaggio o utilizzalo solo per le durate di attesa (che sono soggette a checkpoint).

## Schemi avanzati

Usa funzioni durevoli per creare applicazioni complesse in più fasi che combinano più operazioni durevoli, esecuzione parallela, elaborazione di array, logica condizionale e polling. Questi modelli consentono di creare applicazioni sofisticate che coordinano molte attività mantenendo la tolleranza agli errori e il ripristino automatico.

I modelli avanzati vanno oltre i semplici passaggi sequenziali. È possibile eseguire operazioni contemporaneamente `parallel()`, elaborare array con `conmap()`, attendere condizioni esterne e combinare queste primitive per creare applicazioni affidabili. `waitForCondition()` Ogni operazione duratura crea i propri punti di controllo, in modo che l'applicazione possa essere ripristinata da qualsiasi punto in caso di interruzione.

## Processi di onboarding degli utenti

Guida gli utenti nella registrazione, nella verifica delle e-mail, nella configurazione del profilo e nella configurazione iniziale con la gestione dei nuovi tentativi. Questo esempio combina passaggi sequenziali, callback e logica condizionale per creare un processo di onboarding completo.

### TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-js";

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { userId, email } = event;

    // Step 1: Create user account
    const user = await context.step("create-account", async () => {
      return await userService.createAccount(userId, email);
    });

    // Step 2: Send verification email
    await context.step("send-verification", async () => {
      return await emailService.sendVerification(email);
    });

    // Step 3: Wait for email verification (up to 48 hours)
    const verified = await context.waitForCallback(
      "email-verification",
      async (callbackId) => {
        await notificationService.sendVerificationLink({
          email,
          callbackId,
          expiresIn: 172800
        });
      },
      {
        timeout: { seconds: 172800 }
      }
    );

    if (!verified) {
      await context.step("send-reminder", async () => {
```

```
        await emailService.sendReminder(email);
    });

    return {
        status: "verification_timeout",
        userId,
        message: "Email verification not completed within 48 hours"
    };
}

// Step 4: Initialize user profile in parallel
const setupResults = await context.parallel("profile-setup", [
    async (ctx: DurableContext) => {
        return await ctx.step("create-preferences", async () => {
            return await preferencesService.createDefaults(userId);
        });
    },

    async (ctx: DurableContext) => {
        return await ctx.step("setup-notifications", async () => {
            return await notificationService.setupDefaults(userId);
        });
    },

    async (ctx: DurableContext) => {
        return await ctx.step("create-welcome-content", async () => {
            return await contentService.createWelcome(userId);
        });
    }
]);

// Step 5: Send welcome email
await context.step("send-welcome", async () => {
    const [preferences, notifications, content] = setupResults.getResults();
    return await emailService.sendWelcome({
        email,
        preferences,
        notifications,
        content
    });
});

return {
    status: "onboarding_complete",
```

```
        userId,  
        completedAt: new Date().toISOString()  
    };  
}  
);
```

## Python

```
from aws_durable_execution_sdk_python import DurableContext, durable_execution,  
    WaitConfig  
from datetime import datetime  
  
@durable_execution  
def lambda_handler(event, context: DurableContext):  
    user_id = event['userId']  
    email = event['email']  
  
    # Step 1: Create user account  
    user = context.step(  
        lambda _: user_service.create_account(user_id, email),  
        name='create-account'  
    )  
  
    # Step 2: Send verification email  
    context.step(  
        lambda _: email_service.send_verification(email),  
        name='send-verification'  
    )  
  
    # Step 3: Wait for email verification (up to 48 hours)  
    def send_verification_link(callback_id):  
        notification_service.send_verification_link({  
            'email': email,  
            'callbackId': callback_id,  
            'expiresIn': 172800  
        })  
  
    verified = context.wait_for_callback(  
        send_verification_link,  
        name='email-verification',  
        config=WaitConfig(timeout=172800)
```

```
)

if not verified:
    context.step(
        lambda _: email_service.send_reminder(email),
        name='send-reminder'
    )

    return {
        'status': 'verification_timeout',
        'userId': user_id,
        'message': 'Email verification not completed within 48 hours'
    }

# Step 4: Initialize user profile in parallel
def create_preferences(ctx: DurableContext):
    return ctx.step(
        lambda _: preferences_service.create_defaults(user_id),
        name='create-preferences'
    )

def setup_notifications(ctx: DurableContext):
    return ctx.step(
        lambda _: notification_service.setup_defaults(user_id),
        name='setup-notifications'
    )

def create_welcome_content(ctx: DurableContext):
    return ctx.step(
        lambda _: content_service.create_welcome(user_id),
        name='create-welcome-content'
    )

setup_results = context.parallel(
    [create_preferences, setup_notifications, create_welcome_content],
    name='profile-setup'
)

# Step 5: Send welcome email
def send_welcome(_):
    results = setup_results.get_results()
    preferences, notifications, content = results[0], results[1], results[2]
    return email_service.send_welcome({
        'email': email,
```

```
        'preferences': preferences,  
        'notifications': notifications,  
        'content': content  
    })  
  
    context.step(send_welcome, name='send-welcome')  
  
    return {  
        'status': 'onboarding_complete',  
        'userId': user_id,  
        'completedAt': datetime.now().isoformat()  
    }
```

Il processo combina passaggi sequenziali con punti di controllo per la creazione dell'account e l'invio di e-mail, quindi si interrompe fino a 48 ore in attesa della verifica delle e-mail senza consumare risorse. La logica condizionale gestisce percorsi diversi in base al completamento o al timeout della verifica. Le attività di configurazione del profilo vengono eseguite contemporaneamente utilizzando operazioni parallele per ridurre il tempo totale di esecuzione e ogni passaggio viene riprovato automaticamente in caso di errori transitori per garantire che l'onboarding venga completato in modo affidabile.

### Richiami concatenati tra le funzioni

Richiama altre funzioni Lambda dall'interno di una funzione durevole utilizzando.

`context.invoke()` La funzione chiamante si sospende in attesa del completamento della funzione richiamata, creando un checkpoint che preserva il risultato. Se la funzione chiamante viene interrotta dopo il completamento della funzione richiamata, riprende con il risultato memorizzato senza richiamare nuovamente la funzione.

Utilizza questo modello quando disponi di funzioni specializzate che gestiscono domini specifici (convalida dei clienti, elaborazione dei pagamenti, gestione dell'inventario) e devi coordinarli in un flusso di lavoro. Ogni funzione mantiene la propria logica e può essere richiamata da più funzioni di orchestrazione, evitando la duplicazione del codice.

### TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-js";
```



```
// Main orchestrator function
export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { orderId, customerId } = event;

    // Step 1: Validate customer by invoking customer service function
    const customer = await context.invoke(
      "validate-customer",
      "arn:aws:lambda:us-east-1:123456789012:function:customer-service:1",
      { customerId }
    );

    if (!customer.isValid) {
      return { orderId, status: "rejected", reason: "invalid_customer" };
    }

    // Step 2: Check inventory by invoking inventory service function
    const inventory = await context.invoke(
      "check-inventory",
      "arn:aws:lambda:us-east-1:123456789012:function:inventory-service:1",
      { orderId, items: event.items }
    );

    if (!inventory.available) {
      return { orderId, status: "rejected", reason: "insufficient_inventory" };
    }

    // Step 3: Process payment by invoking payment service function
    const payment = await context.invoke(
      "process-payment",
      "arn:aws:lambda:us-east-1:123456789012:function:payment-service:1",
      {
        customerId,
        amount: inventory.totalAmount,
        paymentMethod: customer.paymentMethod
      }
    );

    // Step 4: Create shipment by invoking fulfillment service function
    const shipment = await context.invoke(
      "create-shipment",
      "arn:aws:lambda:us-east-1:123456789012:function:fulfillment-service:1",
      {
```

```

        orderId,
        items: inventory.allocatedItems,
        address: customer.shippingAddress
    }
);

return {
    orderId,
    status: "completed",
    trackingNumber: shipment.trackingNumber,
    estimatedDelivery: shipment.estimatedDelivery
};
}
);

```

## Python

```

from aws_durable_execution_sdk_python import DurableContext, durable_execution

# Main orchestrator function
@durable_execution
def lambda_handler(event, context: DurableContext):
    order_id = event['orderId']
    customer_id = event['customerId']

    # Step 1: Validate customer by invoking customer service function
    customer = context.invoke(
        'arn:aws:lambda:us-east-1:123456789012:function:customer-service:1',
        {'customerId': customer_id},
        name='validate-customer'
    )

    if not customer['isValid']:
        return {'orderId': order_id, 'status': 'rejected', 'reason':
'invalid_customer'}

    # Step 2: Check inventory by invoking inventory service function
    inventory = context.invoke(
        'arn:aws:lambda:us-east-1:123456789012:function:inventory-service:1',
        {'orderId': order_id, 'items': event['items']},
        name='check-inventory'
    )

```

```

    )

    if not inventory['available']:
        return {'orderId': order_id, 'status': 'rejected', 'reason':
'insufficient_inventory'}

    # Step 3: Process payment by invoking payment service function
    payment = context.invoke(
        'arn:aws:lambda:us-east-1:123456789012:function:payment-service:1',
        {
            'customerId': customer_id,
            'amount': inventory['totalAmount'],
            'paymentMethod': customer['paymentMethod']
        },
        name='process-payment'
    )

    # Step 4: Create shipment by invoking fulfillment service function
    shipment = context.invoke(
        'arn:aws:lambda:us-east-1:123456789012:function:fulfillment-service:1',
        {
            'orderId': order_id,
            'items': inventory['allocatedItems'],
            'address': customer['shippingAddress']
        },
        name='create-shipment'
    )

    return {
        'orderId': order_id,
        'status': 'completed',
        'trackingNumber': shipment['trackingNumber'],
        'estimatedDelivery': shipment['estimatedDelivery']
    }

```

Ogni chiamata crea un checkpoint nella funzione di orchestrator. Se l'orchestrator viene interrotto dopo il completamento della convalida del cliente, riprende da quel checkpoint con i dati del cliente memorizzati, saltando la chiamata di convalida. In questo modo si evitano chiamate duplicate ai servizi a valle e si garantisce un'esecuzione coerente anche dopo le interruzioni.

Le funzioni richiamate possono essere funzioni Lambda durevoli o standard. Se richiami una funzione durevole, può avere un proprio flusso di lavoro in più fasi con attese e checkpoint. L'orchestratore attende semplicemente che l'esecuzione duratura completa sia completata, ricevendo il risultato finale.

#### Note

Cross-account le invocazioni non sono supportate. Tutte le funzioni richiamate devono trovarsi nello stesso AWS account della funzione chiamante.

### Elaborazione in batch con checkpoint

Elabora milioni di record con ripristino automatico dall'ultimo checkpoint riuscito dopo i fallimenti. Questo esempio dimostra come le funzioni durevoli combinino le `map()` operazioni con il chunking e la limitazione della velocità per gestire l'elaborazione di dati su larga scala.

### TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-js";

interface Batch {
  batchIndex: number;
  recordIds: string[];
}

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { datasetId, batchSize = 1000 } = event;

    // Step 1: Get all record IDs to process
    const recordIds = await context.step("fetch-record-ids", async () => {
      return await dataService.getRecordIds(datasetId);
    });

    // Step 2: Split into batches
    const batches: Batch[] = [];
    for (let i = 0; i < recordIds.length; i += batchSize) {
      batches.push({
        batchIndex: Math.floor(i / batchSize),
```

```

        recordIds: recordIds.slice(i, i + batchSize)
    });
}

// Step 3: Process batches with controlled concurrency
const batchResults = await context.map(
    "process-batches",
    batches,
    async (ctx: DurableContext, batch: Batch, index: number) => {
        const processed = await ctx.step(`batch-${batch.batchIndex}`, async () => {
            const results = [];
            for (const recordId of batch.recordIds) {
                const result = await recordService.process(recordId);
                results.push(result);
            }
            return results;
        });

        const validated = await ctx.step(`validate-${batch.batchIndex}`, async () =>
{
            return await validationService.validateBatch(processed);
        });

        return {
            batchIndex: batch.batchIndex,
            recordCount: batch.recordIds.length,
            successCount: validated.successCount,
            failureCount: validated.failureCount
        };
    },
    {
        maxConcurrency: 5
    }
);

// Step 4: Aggregate results
const summary = await context.step("aggregate-results", async () => {
    const results = batchResults.getResults();
    const totalSuccess = results.reduce((sum, r) => sum + r.successCount, 0);
    const totalFailure = results.reduce((sum, r) => sum + r.failureCount, 0);

    return {
        datasetId,
        totalRecords: recordIds.length,
    }
});

```

```

        batchesProcessed: batches.length,
        successCount: totalSuccess,
        failureCount: totalFailure,
        completedAt: new Date().toISOString()
    };
});

return summary;
}
);

```

## Python

```

from aws_durable_execution_sdk_python import DurableContext, durable_execution,
    MapConfig
from datetime import datetime
from typing import List, Dict

@durable_execution
def lambda_handler(event, context: DurableContext):
    dataset_id = event['datasetId']
    batch_size = event.get('batchSize', 1000)

    # Step 1: Get all record IDs to process
    record_ids = context.step(
        lambda _: data_service.get_record_ids(dataset_id),
        name='fetch-record-ids'
    )

    # Step 2: Split into batches
    batches = []
    for i in range(0, len(record_ids), batch_size):
        batches.append({
            'batchIndex': i // batch_size,
            'recordIds': record_ids[i:i + batch_size]
        })

    # Step 3: Process batches with controlled concurrency
    def process_batch(ctx: DurableContext, batch: Dict, index: int):
        batch_index = batch['batchIndex']

```

```
def process_records(_):
    results = []
    for record_id in batch['recordIds']:
        result = record_service.process(record_id)
        results.append(result)
    return results

processed = ctx.step(process_records, name=f'batch-{batch_index}')

validated = ctx.step(
    lambda _: validation_service.validate_batch(processed),
    name=f'validate-{batch_index}'
)

return {
    'batchIndex': batch_index,
    'recordCount': len(batch['recordIds']),
    'successCount': validated['successCount'],
    'failureCount': validated['failureCount']
}

batch_results = context.map(
    process_batch,
    batches,
    name='process-batches',
    config=MapConfig(max_concurrency=5)
)

# Step 4: Aggregate results
def aggregate_results(_):
    results = batch_results.get_results()
    total_success = sum(r['successCount'] for r in results)
    total_failure = sum(r['failureCount'] for r in results)

    return {
        'datasetId': dataset_id,
        'totalRecords': len(record_ids),
        'batchesProcessed': len(batches),
        'successCount': total_success,
        'failureCount': total_failure,
        'completedAt': datetime.now().isoformat()
    }

summary = context.step(aggregate_results, name='aggregate-results')
```

```
return summary
```

I record vengono suddivisi in batch gestibili per evitare di sovraccaricare la memoria o i servizi a valle, quindi vengono elaborati più batch contemporaneamente al controllo del parallelismo. `maxConcurrency` Ogni batch ha un proprio checkpoint, pertanto gli errori eseguono solo un nuovo tentativo sul batch fallito anziché rielaborare tutti i record. Questo modello è ideale per processi ETL, migrazioni di dati o operazioni in blocco in cui l'elaborazione può richiedere ore.

## Fasi successive

- Esplora i concetti [di base](#) da comprendere `DurableContext`, i passaggi e le attese
- Consulta le [best practice](#) per scrivere codice deterministico e ottimizzare le prestazioni
- Scopri come [testare funzioni durevoli](#) localmente e nel cloud
- Confronta le funzioni durevoli con Step Functions per capire quando ogni approccio è più efficace. Vedi Funzioni [durevoli o Funzioni Step](#).

## Sicurezza e autorizzazioni per le funzioni durevoli Lambda

AWS Lambda le funzioni durevoli richiedono autorizzazioni IAM specifiche per eseguire e gestire esecuzioni durevoli. Segui il principio del privilegio minimo concedendo solo le autorizzazioni necessarie a ciascun principale.

### Autorizzazioni IAM per funzioni durevoli

Per la maggior parte delle azioni API di esecuzione duratura, Lambda autorizza la richiesta in base all'ARN di esecuzione durevole, non all'ARN della funzione. Un'esecuzione duratura è una risorsa secondaria della versione di una funzione e il relativo ARN estende l'ARN della funzione:

```
arn:aws:lambda:region:account-id:function:function-name:qualifier/durable-execution/execution-name/execution-id
```

Il qualificatore in un ARN di esecuzione durevole è sempre `$LATEST` o un numero di versione.

Se la policy specifica un ARN di funzione non qualificato nell'elemento Resource, Lambda rifiuta la richiesta perché un ARN non qualificato non corrisponde mai a un ARN di esecuzione durevole. Per concedere l'accesso, aggiungi uno schema di qualificazione alla funzione ARN:



- `arn:aws:lambda:us-east-1:123456789012:function:my-function:*`— Corrisponde alle esecuzioni durevoli di ogni versione.
- `arn:aws:lambda:us-east-1:123456789012:function:my-function:1/durable-execution/*`— Corrisponde solo alle esecuzioni durevoli della versione 1.

La maggior parte delle policy utilizza il `:*` modulo perché i numeri di versione possono cambiare a ogni distribuzione.

**⚠** Gli alias non si applicano ai permessi di esecuzione duraturi

È possibile richiamare una funzione durevole tramite un alias, ma l'ARN dell'esecuzione risultante contiene il numero di versione risolta.

### Autorizzazioni del ruolo di esecuzione

Il ruolo di esecuzione della funzione durevole deve disporre delle autorizzazioni per creare checkpoint e recuperare lo stato di esecuzione. La seguente politica mostra le autorizzazioni minime richieste:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:CheckpointDurableExecution",
        "lambda:GetDurableExecutionState"
      ],
      "Resource": "arn:aws:lambda:region:account-id:function:function-name:*"
    }
  ]
}
```

Quando si crea una funzione durevole utilizzando la console, Lambda aggiunge automaticamente queste autorizzazioni al ruolo di esecuzione. Se crei la funzione usando AWS CLI o AWS CloudFormation, aggiungi queste autorizzazioni al tuo ruolo di esecuzione.

Affinché Lambda possa assumere il tuo ruolo di esecuzione, la politica di fiducia del ruolo deve specificare Lambda service principal (`lambda.amazonaws.com`) come servizio affidabile. L'esempio seguente mostra una politica di fiducia che concede a Lambda l'autorizzazione ad assumere il ruolo.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

### Principio del privilegio minimo

Limita l'accesso agli ARN di funzioni specifiche invece di usare caratteri jolly. Ciò limita il ruolo di esecuzione alle operazioni di checkpoint solo per le funzioni che ne hanno bisogno.

### Example Autorizzazioni con ambito per più funzioni

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:CheckpointDurableExecution",
        "lambda:GetDurableExecutionState"
      ],
      "Resource": [
        "arn:aws:lambda:us-east-1:123456789012:function:orderProcessor:*",
        "arn:aws:lambda:us-east-1:123456789012:function:paymentHandler:*"
      ]
    }
  ]
}
```

In alternativa, puoi utilizzare la policy AWS gestita

`AWSLambdaBasicDurableExecutionRolePolicy` che include le autorizzazioni di esecuzione durature richieste insieme alle autorizzazioni di esecuzione Lambda di base per Amazon Logs. CloudWatch

## Autorizzazioni per la gestione di esecuzioni durevoli

Lambda autorizza il `lambda:GetDurableExecution` `lambda:GetDurableExecutionHistory`, e le `lambda:StopDurableExecution` azioni contro l'ARN di esecuzione durevole.

Per elencare le esecuzioni durevoli di una funzione, usa.

`lambda:ListDurableExecutionsByFunction` Lambda autorizza questa azione contro la funzione ARN indicata nella richiesta. Le seguenti regole si applicano alla corrispondenza degli ARN:

- Una politica che concede la funzione non qualificata ARN soddisfa solo le richieste che utilizzano il nome della funzione non qualificata.
- Una policy che concede un ARN qualificato corrisponde solo alle richieste che utilizzano un nome di funzione qualificato.

Per ulteriori informazioni, consulta [the section called “Funzioni di riferimento nella sezione Risorse delle policy”](#).

Per abbinare le richieste effettuate con entrambe le forme del nome della funzione, includi sia l'ARN della funzione non qualificata che quella qualificata, come mostra la seguente politica:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ManageDurableExecutions",
      "Effect": "Allow",
      "Action": [
        "lambda:GetDurableExecution",
        "lambda:GetDurableExecutionHistory",
        "lambda:StopDurableExecution"
      ],
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:myDurableFunction:*"
    },
    {
      "Sid": "ListDurableExecutions",
      "Effect": "Allow",
      "Action": "lambda:ListDurableExecutionsByFunction",
      "Resource": [
        "arn:aws:lambda:us-east-1:123456789012:function:myDurableFunction",
        "arn:aws:lambda:us-east-1:123456789012:function:myDurableFunction:*"
      ]
    }
  ]
}
```

```

    ]
  }
]
}

```

## Autorizzazioni per l'invio di callback

Il sistema esterno che invia un risultato di callback utilizza le azioni

`lambda:SendDurableExecutionCallbackSuccess`, `lambda:SendDurableExecutionCallbackFailure` e `lambda:SendDurableExecutionCallbackHeartbeat`

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SendCallbacks",
      "Effect": "Allow",
      "Action": [
        "lambda:SendDurableExecutionCallbackSuccess",
        "lambda:SendDurableExecutionCallbackFailure",
        "lambda:SendDurableExecutionCallbackHeartbeat"
      ],
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:myDurableFunction:*"
    }
  ]
}

```

Lambda autorizza le richieste di callback rispetto all'ARN dell'esecuzione durevole che ha creato il callback.

## Crittografia statale

Le funzioni Lambda durable crittografano i dati di esecuzione a riposo. L'esecuzione di ogni funzione mantiene uno stato isolato a cui le altre esecuzioni non possono accedere. È possibile configurare una chiave gestita dal cliente per crittografare i dati di esecuzione durevoli. Per ulteriori informazioni, consulta [Crittografia AWS Lambda dati di esecuzione durevoli](#).

I dati del checkpoint includono:

- Risultati ed errori delle fasi

- Input di richiamo concatenati

Per l'elenco completo dei dati di esecuzione durevoli crittografati a riposo, vedere. [Cosa è crittografato](#)

Tutti i dati vengono crittografati in transito utilizzando TLS quando Lambda legge o scrive i dati dei checkpoint.

Crittografia personalizzata con serializzatori e deserializzatori personalizzati

Per i requisiti di sicurezza critici, puoi implementare il tuo meccanismo di crittografia e decrittografia utilizzando serializzatori e deserializzatori personalizzati () utilizzando un SDK durevole. SerDer Questo approccio offre il pieno controllo sulle chiavi di crittografia e sugli algoritmi utilizzati per proteggere i dati dei checkpoint.

#### Important

Quando si utilizza la crittografia personalizzata, si perde la visibilità dei risultati delle operazioni nella console Lambda e nelle risposte API. I dati del checkpoint appaiono crittografati nella cronologia delle esecuzioni e non possono essere esaminati senza decrittografia.

Requisiti kms:Encrypt e kms:Decrypt autorizzazioni per il ruolo di esecuzione della funzione per la AWS KMS chiave utilizzata nell'implementazione personalizzata. SerDer

## CloudTrail registrazione

Lambda registra le operazioni dei checkpoint come eventi di dati in. AWS CloudTrailÈ possibile utilizzarlo CloudTrail per verificare la creazione dei checkpoint, tenere traccia delle modifiche allo stato di esecuzione e monitorare l'accesso a dati di esecuzione durevoli.

Le operazioni relative ai checkpoint vengono CloudTrail visualizzate nei log con i seguenti nomi di evento:

- CheckpointDurableExecution- Registrato quando viene completato un passaggio e viene creato un checkpoint
- GetDurableExecutionState- Registrato quando Lambda recupera lo stato di esecuzione durante la riproduzione

Per abilitare la registrazione degli eventi di dati per funzioni durature, configura un CloudTrail percorso per registrare gli eventi dei dati Lambda. Per ulteriori informazioni, consulta [Registrazione degli eventi relativi ai dati](#) nella Guida per l' CloudTrail utente.

Esempio: immissione del CloudTrail registro per il funzionamento del checkpoint

```
{
  "eventVersion": "1.08",
  "eventTime": "2024-11-16T10:30:45Z",
  "eventName": "CheckpointDurableExecution",
  "eventSource": "lambda.amazonaws.com",
  "requestParameters": {
    "functionName": "myDurableFunction",
    "executionId": "exec-abc123",
    "stepId": "step-1"
  },
  "responseElements": null,
  "eventType": "AwsApiCall"
}
```

## Cross-account considerazioni

Se si richiamano funzioni permanenti su più AWS account, l'account chiamante richiede l'`lambda:InvokeFunction` autorizzazione, ma le operazioni di checkpoint utilizzano sempre il ruolo di esecuzione nell'account della funzione. L'account chiamante non può accedere direttamente ai dati del checkpoint o allo stato di esecuzione.

Questo isolamento garantisce che i dati dei checkpoint rimangano protetti all'interno dell'account della funzione, anche se richiamati da account esterni.

## Funzionalità di sicurezza Lambda ereditate

Le funzioni durevoli ereditano tutte le funzionalità di sicurezza, governance e conformità da Lambda. Queste includono la connettività VPC, la crittografia delle variabili di ambiente, le code di lettere morte, la concorrenza riservata, gli URL delle funzioni, la firma del codice e le certificazioni di conformità come SOC, PCI DSS e HIPAA.

Per informazioni dettagliate sulle funzionalità di sicurezza di Lambda, consulta Security in nella Lambda Developer Guide. [AWS Lambda](#) Le uniche considerazioni aggiuntive sulla sicurezza per le funzioni durevoli sono le autorizzazioni IAM documentate in questa guida.

# Crittografia AWS Lambda dati di esecuzione durevoli

Grazie a funzioni [AWS Lambda durevoli](#), è possibile creare applicazioni resilienti in più fasi che durano fino a un anno, utilizzando i checkpoint per ripristinare ogni esecuzione dagli errori riproducendo il lavoro completato.

Lambda crittografa sempre i dati di esecuzione durevoli a riposo. Puoi inoltre configurare la tua chiave gestita AWS KMS dal cliente sulla funzione per il controllo della rotazione, la visibilità degli audit o la conformità. Con una chiave gestita dal cliente, solo i committenti autorizzati possono leggere i dati di esecuzione.

## Come funziona la crittografia

Un'esecuzione durevole Lambda utilizza la stessa chiave KMS con cui è iniziata per tutta la sua durata. La modifica o la rimozione della chiave sulla funzione ha effetto solo sulle esecuzioni che iniziano dopo la modifica. Scopri [Quando la chiave gestita dal cliente non è disponibile](#) come si comporta un'esecuzione duratura quando la sua chiave non è disponibile.

Le chiavi gestite dal cliente sono soggette a costi standard AWS KMS . Per i dettagli sui prezzi, vedere [Prezzi di AWS Key Management Service](#).

## Cosa è crittografato

Quando si configura una chiave gestita dal cliente su una funzione durevole, Lambda utilizza tale chiave per crittografare i seguenti dati di esecuzione duratura a riposo:

- Il payload di input che trasmetti con ogni richiesta. Invoke
- I dati dei checkpoint sono stati mantenuti dall'CheckpointDurableExecutionAPI, inclusi i risultati delle fasi, gli errori dei passaggi e gli input di invocazione concatenati.
- Risultati ed errori di esecuzione.
- Risultati ed errori di callback inviati tramite SendDurableExecutionCallbackSuccess e SendDurableExecutionCallbackFailure.

### Function-level e le chiavi di esecuzione durevoli sono indipendenti

Il livello di funzione KMSKeyArn che crittografa le variabili di [ambiente](#), i pacchetti [di distribuzione.zip](#) e le [SnapStart](#) istantanee è separato da quello KMSKeyArn in DurableConfig che crittografa i dati di esecuzione durevoli. L'impostazione di uno non

imposta l'altro. È possibile utilizzare la stessa chiave KMS per entrambi oppure è possibile utilizzare chiavi KMS diverse.

## Configura la crittografia delle chiavi gestita dal cliente

L'impostazione della crittografia delle chiavi gestita dal cliente per una funzione duratura è un processo in tre fasi. Completa i seguenti passaggi in ordine.

### Creazione di una chiave gestita dal cliente

Le funzioni durevoli supportano le chiavi KMS di crittografia simmetrica nella stessa AWS regione della funzione. Cross-Region le chiavi non sono supportate. Per creare una chiave simmetrica gestita dal cliente, segui i passaggi per la [creazione di chiavi](#) KMS di crittografia simmetrica nella Guida per gli sviluppatori. AWS Key Management Service

### Permissions

AWS KMS Le autorizzazioni vengono concesse tramite la politica chiave della chiave KMS.

La configurazione di una chiave gestita dal cliente richiede AWS KMS le autorizzazioni sul principale che crea o aggiorna la funzione. L'invocazione di una funzione durevole non richiede AWS KMS autorizzazioni sul chiamante; Lambda esegue la crittografia con il suo service principal.

### Policy della chiave

Le [policy della chiave](#) controllano l'accesso alla chiave gestita dal cliente. Ogni chiave gestita dal cliente deve avere esattamente una policy chiave. In una politica chiave, Resource: "\*" si riferisce solo alla chiave KMS a cui è associata la politica, non a tutte le chiavi KMS dell'account.

La politica chiave di una funzione durevole concede a ciascun committente solo le AWS KMS azioni di cui ha bisogno, suddivise per servizio e funzione (ARN). Le sezioni seguenti descrivono le dichiarazioni, le condizioni e un esempio completo.

### Dichiarazioni politiche obbligatorie

La politica garantisce le seguenti funzionalità:

- Abilita le autorizzazioni utente IAM. Concede all'account root l'accesso incondizionato per gestire la chiave. Questa dichiarazione non utilizza condizioni perché l'ambito impedirebbe di ruotare, aggiornare o eliminare la chiave.



- Consenti a Lambda di utilizzare questo tasto per funzioni durature. Assegna il responsabile del servizio Lambda e. `kms:GenerateDataKey` `kms:Decrypt`
- Consenti al ruolo di esecuzione della funzione di decrittografare i dati di esecuzione durevoli. Assegna al ruolo `kms:Decrypt` di esecuzione la lettura dello stato e il progresso dell'esecuzione.
- Consenti all'autore della funzione di descrivere questa chiave. Consente a `kms:DescribeKey` Lambda di convalidare che la chiave è simmetrica e abilitata durante o. `CreateFunction` `UpdateFunctionConfiguration`
- Consenti all'autore della funzione di convalidare questa chiave per una funzione specifica. Concede `kms:GenerateDataKey` e `kms:Decrypt` quindi Lambda può convalidare le autorizzazioni delle chiavi e l'accesso con il contesto di crittografia della funzione durante o. `CreateFunction` `UpdateFunctionConfiguration` Ciò conferma che la policy chiave accetta le chiamate per questa funzione specifica prima che la funzione venga creata o aggiornata.
- Consenti operatori di esecuzione duraturi. Concede `kms:Decrypt` agli operatori le chiamate verso `GetDurableExecution`, `GetDurableExecutionHistory` with `IncludeExecutionData=true` `GetDurableExecutionState` `StopDurableExecution`, e le API di callback.

Come procedura consigliata, utilizzate principi separati per il ruolo di esecuzione, l'autore della funzione e l'operatore di esecuzione durevole in modo che ogni identità abbia solo le funzionalità necessarie.

Condizioni politiche consigliate

La politica utilizza le seguenti condizioni per limitare l'accesso:

- [kms: ViaService](#) limita l'uso delle chiavi alle richieste inoltrate tramite Lambda. Applicandolo al ruolo di esecuzione, all'autore della funzione e alle istruzioni dell'operatore si impedisce loro di utilizzare la chiave direttamente. AWS KMS
- [aws: SourceArn](#) e [aws: SourceAccount](#) proteggono dal problema [dei vice confusi](#) tra servizi diversi. Lambda inoltra questi valori quando chiama AWS KMS utilizzando le proprie credenziali di servizio. Questa condizione si applica solo alla dichiarazione principale del servizio Lambda. Le istruzioni relative al ruolo di esecuzione, all'autore della funzione e all'operatore utilizzano AWS KMS le credenziali della sessione di accesso inoltrato del chiamante, che non contengono queste intestazioni.

- [kms: definisce l'EncryptionContext:aws:lambda:FunctionArn](#)ambito della chiave di una funzione specifica. Lambda lo aggiunge al contesto di crittografia su ogni AWS KMS chiamata per dati di esecuzione durevoli.

L'esempio seguente combina le dichiarazioni e le condizioni precedenti.

Example Politica chiave per le funzioni durevoli

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:root" },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "Allow Lambda to use this key for durable functions",
      "Effect": "Allow",
      "Principal": { "Service": "lambda.amazonaws.com" },
      "Action": [ "kms:GenerateDataKey", "kms:Decrypt" ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333",
          "aws:SourceArn": "arn:aws:lambda:us-east-1:111122223333:function:myDurableFunction",
          "kms:EncryptionContext:aws:lambda:FunctionArn": "arn:aws:lambda:us-east-1:111122223333:function:myDurableFunction"
        }
      }
    },
    {
      "Sid": "Allow the function execution role to decrypt durable execution data",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/myDurableFunctionRole" },
      "Action": "kms:Decrypt",
      "Resource": "*",
      "Condition": {
```

```

        "StringEquals": {
            "kms:ViaService": "lambda.us-east-1.amazonaws.com",
            "kms:EncryptionContext:aws:lambda:FunctionArn": "arn:aws:lambda:us-
east-1:111122223333:function:myDurableFunction"
        }
    },
    {
        "Sid": "Allow the function author to describe this key",
        "Effect": "Allow",
        "Principal": { "AWS": "arn:aws:iam::111122223333:role/FunctionAuthor" },
        "Action": "kms:DescribeKey",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "kms:ViaService": "lambda.us-east-1.amazonaws.com"
            }
        }
    },
    {
        "Sid": "Allow the function author to validate this key for a specific
function",
        "Effect": "Allow",
        "Principal": { "AWS": "arn:aws:iam::111122223333:role/FunctionAuthor" },
        "Action": [ "kms:GenerateDataKey", "kms:Decrypt" ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "kms:ViaService": "lambda.us-east-1.amazonaws.com",
                "kms:EncryptionContext:aws:lambda:FunctionArn": "arn:aws:lambda:us-
east-1:111122223333:function:myDurableFunction"
            }
        }
    },
    {
        "Sid": "Allow durable execution operators",
        "Effect": "Allow",
        "Principal": { "AWS": "arn:aws:iam::111122223333:role/
DurableExecutionOperator" },
        "Action": "kms:Decrypt",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "kms:ViaService": "lambda.us-east-1.amazonaws.com",

```

```
        "kms:EncryptionContext:aws:lambda:FunctionArn": "arn:aws:lambda:us-east-1:111122223333:function:myDurableFunction"
    }
}
]
```

Per ulteriori informazioni sulle politiche AWS KMS chiave, consulta [How to change a key policy](#) nella AWS Key Management Service Developer Guide.

Configura una chiave gestita dal cliente su una funzione duratura

È possibile configurare la chiave gestita dal cliente per dati di esecuzione duraturi tramite il `KMSKeyArn` campo dell'`DurableConfig` oggetto. Impostala quando crei la funzione con [CreateFunction](#); aggiornala su una funzione durevole esistente con [UpdateFunctionConfiguration](#).

Lambda console

Per configurare una chiave gestita dal cliente su una funzione durevole esistente

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli una funzione duratura.
3. Scegli Configurazione, quindi scegli Esecuzione duratura dalla barra di navigazione a sinistra.
4. Scegli Modifica.
5. In Crittografia, scegli Usa una chiave gestita dal cliente, quindi scegli una chiave KMS dall'elenco.
6. Scegli Save (Salva).

AWS CLI

Per configurare una chiave gestita dal cliente quando crei una funzione

Nel seguente [esempio di](#) creazione di una funzione, `--durable-config` specifica l'ARN della chiave gestita dal cliente insieme al timeout di esecuzione duraturo e al periodo di conservazione.

```
aws lambda create-function \
```

```
--function-name myDurableFunction \
--runtime nodejs24.x \
--handler index.handler \
--role arn:aws:iam::111122223333:role/myDurableFunctionRole \
--zip-file fileb://function.zip \
--durable-config '{"KMSKeyArn":"arn:aws:kms:us-east-1:111122223333:key/key-
id","ExecutionTimeout":3600,"RetentionPeriodInDays":30}'
```

Per configurare una chiave gestita dal cliente su una funzione durevole esistente

Usa il comando <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/update-function-configuration.html> `update-function-configuration`. Includi tutti `DurableConfig` i campi che desideri conservare, perché `--durable-config` sostituisce l'intero oggetto.

```
aws lambda update-function-configuration \
--function-name myDurableFunction \
--durable-config '{"KMSKeyArn":"arn:aws:kms:us-east-1:111122223333:key/key-
id","ExecutionTimeout":3600,"RetentionPeriodInDays":30}'
```

Per rimuovere la chiave gestita dal cliente da una funzione durevole

Ometti `KMSKeyArn` da `--durable-config`. Le esecuzioni esistenti continuano a utilizzare la chiave gestita dal cliente con cui hanno iniziato. Le nuove esecuzioni utilizzano invece la crittografia predefinita.

```
aws lambda update-function-configuration \
--function-name myDurableFunction \
--durable-config '{"ExecutionTimeout":3600,"RetentionPeriodInDays":30}'
```

## AWS CloudFormation

In una `AWS::Lambda::Function` risorsa, imposta la `KMSKeyArn` proprietà di `DurableConfig`:

```
Resources:
  MyDurableFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: myDurableFunction
      Runtime: nodejs24.x
      Handler: index.handler
      Role: !GetAtt MyDurableFunctionRole.Arn
```

```
Code:
  ZipFile: |
    // Your durable function code
DurableConfig:
  KMSKeyArn: "arn:aws:kms:us-east-1:111122223333:key/key-id"
  ExecutionTimeout: 3600
  RetentionPeriodInDays: 30
```

### Important

Ogni esecuzione durevole utilizza la chiave gestita dal cliente che è stata configurata sulla funzione all'avvio. La modifica o la rimozione della chiave ha effetto solo sulle esecuzioni che iniziano dopo la modifica; le esecuzioni in corso continuano a utilizzare la chiave con cui sono iniziate fino al completamento o al fallimento.

## Lettura di dati di esecuzione durevoli

Due API di lettura restituiscono dati di esecuzione duraturi:

- `GetDurableExecution` restituisce lo stato corrente di una singola esecuzione, incluso il payload di input, i dati di checkpoint più recenti e le informazioni sui risultati o sugli errori.
- `GetDurableExecutionHistory` restituisce l'elenco ordinato degli eventi emessi dall'esecuzione, mostrando gli input e i risultati del checkpoint, gli input e i risultati con invocazione concatenata e il risultato finale.

`IncludeExecutionData` Entrambe le API accettano un parametro di richiesta. In tal caso `IncludeExecutionData` è `true`, Lambda utilizza le credenziali del chiamante per richiamare la chiave gestita dal `kms:Decrypt` cliente. Lambda restituisce quindi i dati di esecuzione decrittografati nella risposta. L'identità del chiamante deve figurare `kms:Decrypt` sulla chiave gestita dal cliente.

Quando `IncludeExecutionData` è `false`, Lambda non chiama AWS KMS né decrittografa i dati di esecuzione e la risposta è impostata su `ExecutionDataIncluded` `false`. La risposta include ancora `DurableConfig`, il che fa eco all'ARN chiave gestita dal cliente utilizzata dall'esecuzione. `IncludeExecutionData` l'impostazione predefinita è `false`, quindi i chiamanti che necessitano solo di metadati non necessitano di autorizzazioni. AWS KMS

**GetDurableExecution** Esempio: risposta con **IncludeExecutionData=false**

```
{
  "DurableExecutionArn": "arn:aws:lambda:us-east-1:111122223333:function:myDurableFunction:execution:exec-abc123",
  "DurableExecutionName": "exec-abc123",
  "FunctionArn": "arn:aws:lambda:us-east-1:111122223333:function:myDurableFunction",
  "StartTimestamp": 1733256000,
  "Status": "RUNNING",
  "ExecutionDataIncluded": false,
  "DurableConfig": {
    "ExecutionTimeout": 3600,
    "KMSKeyArn": "arn:aws:kms:us-east-1:111122223333:key/key-id",
    "RetentionPeriodInDays": 30
  }
}
```

## Invocazioni concatenate

Quando una funzione durevole utilizza un richiamo concatenato per chiamare un'altra funzione durevole, Lambda considera l'esecuzione secondaria come un'esecuzione durevole indipendente: ha il proprio ID di esecuzione, il proprio flusso di checkpoint e la propria chiave gestita dal cliente. La configurazione della chiave gestita dal cliente del genitore non ha alcun effetto sui dati del figlio e la configurazione della chiave gestita dal cliente del bambino non ha alcun effetto sui dati del genitore.

### Permissions

Il responsabile del servizio Lambda deve avere `kms:GenerateDataKey` e `kms:Decrypt` sulla chiave gestita dal cliente della funzione figlio. Se le funzioni padre e figlio utilizzano chiavi gestite dal cliente diverse, concedi al responsabile del servizio l'accesso su ciascuna chiave separatamente. Il ruolo di esecuzione della funzione principale non richiede autorizzazioni sulla chiave gestita dal cliente del figlio.

Identificare quale chiave ha crittografato quale esecuzione

Entrambi `GetDurableExecution` e `ListDurableExecutionsByFunction` restituiscono l'ARN della chiave gestita dal cliente che ha crittografato ogni esecuzione. Utilizza queste API per confermare quale chiave era attiva all'avvio di un'esecuzione principale o secondaria.

## Caching della chiave dei dati

Per ridurre il volume delle AWS KMS chiamate e migliorare la disponibilità durante le interruzioni del servizio, Lambda memorizza nella cache una chiave di dati generata dalla chiave gestita dal cliente

per un massimo di 15 minuti. Mentre la cache è calda, Lambda riutilizza la stessa chiave di dati tra le operazioni all'interno di un'esecuzione e tra le esecuzioni avviate durante la finestra della cache.

## Costo

Per-execution AWS KMS il costo rimane basso perché Lambda chiama `GenerateDataKey` al massimo una volta per finestra di cache, non una volta per checkpoint. A tassi di richiesta elevati, il costo per esecuzione diminuisce ulteriormente perché più esecuzioni condividono ogni chiave di dati memorizzata nella cache. Ti vengono fatturate le AWS KMS chiamate effettivamente effettuate da Lambda, non per ogni checkpoint o lettura.

## Stabilità statica

Le chiavi dati rimangono memorizzate nella cache per un massimo di 15 minuti. Le modifiche alla chiave hanno effetto al successivo aggiornamento della cache. Durante le interruzioni prolungate del servizio, Lambda continua a funzionare dalla cache, mantenendo in esecuzione le esecuzioni in corso.

## CloudTrail

La memorizzazione nella cache delle chiavi di dati consente di visualizzare meno `GenerateDataKey` eventi CloudTrail rispetto alle esecuzioni o ai checkpoint. Vedi ad esempio [Monitoraggio delle chiavi KMS per funzioni durature](#) eventi.

## Quando la chiave gestita dal cliente non è disponibile

Se la chiave gestita dal cliente con cui è iniziata un'esecuzione è disabilitata, è programmata per l'eliminazione o il suo accesso è stato revocato tramite la politica delle chiavi, l'esecuzione non può procedere ulteriormente. Al checkpoint successivo, Lambda fallisce l'esecuzione con un errore non ripetibile. AWS KMS Il ripristino dell'accesso alla chiave non riprende automaticamente l'esecuzione. È necessario iniziare una nuova esecuzione.

Le API che leggono o scrivono payload falliscono anche quando la chiave non è disponibile. Possono restituire una delle seguenti eccezioni:

- `KMSAccessDeniedException`: Lambda non è riuscita a decrittografare i dati di esecuzione durevoli perché l'accesso alla chiave KMS è stato negato.
- `KMSDisabledException`: Lambda non è riuscita a decrittografare i dati di esecuzione duratura perché la chiave KMS è disabilitata.



- `KMSInvalidStateException`: Lambda non è riuscita a decrittografare i dati di esecuzione duratura perché lo stato della chiave KMS non è valido per `Decrypt`.
- `KMSNotFoundException`: Lambda non è riuscita a decrittografare i dati di esecuzione duratura perché la chiave KMS non è stata trovata.

Il ripristino dell'accesso alla chiave KMS consente a queste API di lettura di avere nuovamente successo per le esecuzioni già fallite. Le esecuzioni non riuscite rimangono fallite; è necessario iniziare una nuova esecuzione.

Per controllare i metadati di esecuzione mentre la chiave KMS non è disponibile, chiama con `GetDurableExecution IncludeExecutionData=false`. Ciò restituisce lo stato dell'esecuzione, i timestamp e `DurableConfig` (inclusa la chiave KMS ARN) senza chiamare `AWS KMS`.

Poiché Lambda memorizza nella cache le chiavi di dati, la disabilitazione di una chiave non ha effetto immediato. Per informazioni dettagliate, vedi [Caching della chiave dei dati](#).

#### Important

L'eliminazione di una chiave KMS da cui dipendono ancora le esecuzioni in corso o conservate distrugge definitivamente tali esecuzioni e la relativa cronologia.

## Monitoraggio delle chiavi KMS per funzioni durature

Quando utilizzi una chiave gestita dal cliente con una funzione duratura, puoi utilizzarla [AWS CloudTrail](#) per tenere traccia delle AWS KMS chiamate effettuate da Lambda per tuo conto. Per indicazioni generali sulla ricerca AWS KMS degli eventi, consulta [Searching AWS KMS API activity](#).

Per confermare che Lambda stia utilizzando la tua chiave come previsto, cerca questi campi in ogni evento:

- `eventName`: uno dei `GenerateDataKeyDecrypt`, o `DescribeKey`
- `eventSource`: `kms.amazonaws.com`
- `userIdentity.invokedBy`: `lambda.amazonaws.com`
- `requestParameters.encryptionContext.aws:lambda:FunctionArn`: l'ARN della funzione durevole

Gli esempi seguenti sono CloudTrail gli eventi di una `UpdateFunctionConfiguration` chiamata `CreateFunction` o che configura una chiave gestita dal cliente su una funzione durevole. Lambda emette tre AWS KMS chiamate per convalidare la chiave: una vera e una a `DescribeKey` secco e `GenerateDataKey Decrypt`.

## GenerateDataKey

Quando imposti o modifichi l'ingresso `DurableConfig`, Lambda emette una prova `KMSKeyArn` a secco sulla chiave gestita dal cliente per convalidare che la policy delle chiavi consenta a Lambda di derivare le chiavi di dati per il `GenerateDataKey` contesto di crittografia di questa funzione. Il dry-run non esegue operazioni crittografiche ma registra un evento completo. CloudTrail L'evento di esempio seguente registra l'operazione di esecuzione a secco: `GenerateDataKey`.

```
{
  "eventVersion": "1.11",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAI23456789EXAMPLE:example",
    "arn": "arn:aws:sts::111122223333:assumed-role/FunctionAuthor/example",
    "accountId": "111122223333",
    "accessKeyId": "ASIA123456789EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAI23456789EXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/FunctionAuthor",
        "accountId": "111122223333",
        "userName": "FunctionAuthor"
      },
      "attributes": {
        "creationDate": "2026-01-15T16:54:42Z",
        "mfaAuthenticated": "false"
      }
    },
    "invokedBy": "lambda.amazonaws.com"
  },
  "eventTime": "2026-01-15T16:55:00Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "GenerateDataKey",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "lambda.amazonaws.com",
  "userAgent": "lambda.amazonaws.com",
```

```

    "errorCode": "DryRunOperationException",
    "errorMessage": "The request would have succeeded, but the DryRun option is
set.",
    "requestParameters": {
      "encryptionContext": {
        "aws:lambda:FunctionArn": "arn:aws:lambda:us-
east-1:111122223333:function:myDurableFunction"
      },
      "dryRun": true,
      "keyId": "arn:aws:kms:us-east-1:111122223333:key/key-id",
      "keySpec": "AES_256"
    },
    "responseElements": null,
    "additionalEventData": {
      "keyMaterialId":
"a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0EXAMPLE"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEebbbbb",
    "readOnly": true,
    "resources": [
      {
        "accountId": "111122223333",
        "type": "AWS::KMS::Key",
        "ARN": "arn:aws:kms:us-east-1:111122223333:key/key-id"
      }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
  }
}

```

## Decrypt

Quando imposti o modifichi l'ingresso `DurableConfig`, Lambda emette anche una prova `KMSKeyArn` a secco sulla chiave gestita dal cliente per convalidare che la policy della chiave consenta a Lambda di decrittografare i dati per il Decrypt contesto di crittografia di questa funzione. Il dry-run utilizza, quindi non è richiesto alcun testo cifrato reale. `IGNORE_CIPHertext` L'evento di esempio seguente registra l'operazione di funzionamento a secco: Decrypt

```

{
  "eventVersion": "1.11",

```

```

"userIdentity": {
  "type": "AssumedRole",
  "principalId": "AROA123456789EXAMPLE:example",
  "arn": "arn:aws:sts::111122223333:assumed-role/FunctionAuthor/example",
  "accountId": "111122223333",
  "accessKeyId": "ASIA123456789EXAMPLE",
  "sessionContext": {
    "sessionIssuer": {
      "type": "Role",
      "principalId": "AROA123456789EXAMPLE",
      "arn": "arn:aws:iam::111122223333:role/FunctionAuthor",
      "accountId": "111122223333",
      "userName": "FunctionAuthor"
    },
    "attributes": {
      "creationDate": "2026-01-15T16:54:42Z",
      "mfaAuthenticated": "false"
    }
  },
  "invokedBy": "lambda.amazonaws.com"
},
"eventTime": "2026-01-15T16:55:00Z",
"eventSource": "kms.amazonaws.com",
"eventName": "Decrypt",
"awsRegion": "us-east-1",
"sourceIPAddress": "lambda.amazonaws.com",
"userAgent": "lambda.amazonaws.com",
"errorCode": "DryRunOperationException",
"errorMessage": "The request would have succeeded, but the DryRun option is set.",
"requestParameters": {
  "encryptionContext": {
    "aws:lambda:FunctionArn": "arn:aws:lambda:us-east-1:111122223333:function:myDurableFunction"
  },
  "dryRun": true,
  "keyId": "arn:aws:kms:us-east-1:111122223333:key/key-id",
  "dryRunModifiers": ["IGNORE_CIPHertext"],
  "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
},
"responseElements": null,
"additionalEventData": {
  "keyMaterialId":
"a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0EXAMPLE"

```

```

},
"requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
"eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbbb",
"readOnly": true,
"resources": [
  {
    "accountId": "111122223333",
    "type": "AWS::KMS::Key",
    "ARN": "arn:aws:kms:us-east-1:111122223333:key/key-id"
  }
],
"eventType": "AwsApiCall",
"managementEvent": true,
"recipientAccountId": "111122223333",
"eventCategory": "Management"
}

```

## DescribeKey

Quando si imposta o si modifica l'ingresso `KMSKeyArnDurableConfig`, Lambda chiama `DescribeKey` per confermare che la chiave sia simmetrica e abilitata prima di accettare la modifica. A differenza delle `Decrypt` sonde `GenerateDataKey` and, si tratta di una chiamata reale, non di una corsa a secco. L'evento di esempio seguente registra l'operazione `DescribeKey`:

```

{
  "eventVersion": "1.11",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAI23456789EXAMPLE:example",
    "arn": "arn:aws:sts::111122223333:assumed-role/FunctionAuthor/example",
    "accountId": "111122223333",
    "accessKeyId": "ASIA123456789EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAI23456789EXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/FunctionAuthor",
        "accountId": "111122223333",
        "userName": "FunctionAuthor"
      },
      "attributes": {
        "creationDate": "2026-01-15T16:54:42Z",

```

```

        "mfaAuthenticated": "false"
      },
    },
    "invokedBy": "lambda.amazonaws.com"
  },
  "eventTime": "2026-01-15T16:55:00Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "DescribeKey",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "lambda.amazonaws.com",
  "userAgent": "lambda.amazonaws.com",
  "requestParameters": {
    "keyId": "arn:aws:kms:us-east-1:111122223333:key/key-id"
  },
  "responseElements": null,
  "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
  "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbbb",
  "readOnly": true,
  "resources": [
    {
      "accountId": "111122223333",
      "type": "AWS::KMS::Key",
      "ARN": "arn:aws:kms:us-east-1:111122223333:key/key-id"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "111122223333",
  "eventCategory": "Management"
}

```

## Argomenti correlati alla crittografia

- [Crittografia dei dati inattivi per Lambda](#)
- [Protezione delle variabili di ambiente Lambda](#)
- [Crittografia dei pacchetti di distribuzione Lambda .zip](#)
- [Modello di sicurezza per Lambda SnapStart](#)
- [Configura le funzioni Lambda durable](#)

## SDK di esecuzione durevole

L'SDK di esecuzione durevole è la base per la creazione di funzioni durature. Fornisce primitive per verificare l'avanzamento, gestire i tentativi e gestire il flusso di esecuzione. L'SDK astrae la complessità della gestione e della riproduzione dei checkpoint, consentendoti di scrivere codice sequenziale che diventa automaticamente tollerante ai guasti.

L'SDK è disponibile per, Python JavaScript, TypeScript Java e C# (.NET). Per una documentazione completa sulle API, tutorial di avvio rapido e guide specifiche per il linguaggio, consulta la Durable Execution SDK Developer Guide. [AWS](#)

### Cosa fa l'SDK

**Gestione dei checkpoint:** l'SDK crea automaticamente dei checkpoint man mano che la funzione esegue operazioni durature. Ogni checkpoint registra il tipo di operazione, gli input e i risultati. Quando la funzione completa un passaggio, l'SDK mantiene il checkpoint prima di continuare. Ciò garantisce che la funzione possa riprendere da qualsiasi operazione completata se interrotta.

**Coordinamento della riproduzione:** quando la funzione riprende dopo una pausa o un'interruzione, l'SDK esegue la riproduzione. Esegue il codice dall'inizio ma salta le operazioni completate, utilizzando i risultati dei checkpoint memorizzati invece di rieseguirli. L'SDK garantisce che la riproduzione sia deterministica. Con gli stessi input e lo stesso registro dei checkpoint, la funzione produce gli stessi risultati.

**Isolamento dello stato:** l'SDK mantiene lo stato di esecuzione separato dalla logica aziendale. Ogni esecuzione durevole ha il proprio registro dei checkpoint a cui le altre esecuzioni non possono accedere. L'SDK crittografa i dati dei checkpoint inattivi e garantisce che lo stato rimanga coerente tra i replay.

Per una spiegazione dettagliata del funzionamento del checkpoint e del comportamento di ripetizione, consulta Concetti [chiave](#) nella Durable Execution SDK Developer Guide. AWS

### Operazioni durevoli

L'SDK fornisce alla funzione un oggetto. `DurableContext` Questo contesto sostituisce il contesto Lambda standard e fornisce metodi per creare checkpoint, gestire il flusso di esecuzione e coordinarsi con sistemi esterni.

`DurableContext` Fornisce le seguenti operazioni per la creazione di flussi di lavoro duraturi:

| Operation                             | Description                                                                                                 |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------|
| <a href="#">Fase</a>                  | Esegui e controlla un'unità di lavoro con strategie di riprova configurabili e semantica di esecuzione.     |
| <a href="#">Attendi</a>               | Sospendi l'esecuzione per una durata specificata senza consumare risorse di calcolo.                        |
| <a href="#">Attendi la condizione</a> | Esegui il sondaggio per individuare una condizione con un checkpoint automatico tra un tentativo e l'altro. |
| <a href="#">Richiamata</a>            | Metti in pausa l'esecuzione e attendi che un sistema esterno fornisca l'input tramite l'API Lambda.         |
| <a href="#">Invoke</a>                | Chiama un'altra funzione Lambda e attendi il risultato, con un checkpoint automatico.                       |
| <a href="#">Parallelo</a>             | Esegui più operazioni contemporaneamente con politiche di completamento configurabili.                      |
| <a href="#">Mappa</a>                 | Elabora ogni elemento di una raccolta contemporaneamente con il controllo della concorrenza opzionale.      |
| <a href="#">Contesto secondario</a>   | Crea un contesto di esecuzione isolato per raggruppare più operazioni.                                      |

Ogni operazione duratura crea automaticamente dei checkpoint, assicurando che la funzione possa riprendere da qualsiasi punto. Per riferimenti dettagliati alle API, esempi di codice e utilizzo specifico della lingua, consulta [SDK Reference](#) nella AWS Durable Execution SDK Developer Guide.

## Come vengono misurate le operazioni durevoli

Ogni operazione duratura eseguita `DurableContext` crea punti di controllo per tenere traccia dell'avanzamento dell'esecuzione e archiviare i dati sullo stato. Queste operazioni comportano costi in base al loro utilizzo e i checkpoint possono contenere dati che contribuiscono ai costi di scrittura e conservazione dei dati. I dati archiviati includono i dati degli eventi di richiamo, i payload restituiti dalle fasi e i dati trasmessi durante il completamento dei callback. Comprendere in che modo vengono



misurare le operazioni durature consente di stimare i costi di esecuzione e ottimizzare i flussi di lavoro. Per informazioni dettagliate sui prezzi, consulta la pagina dei prezzi di [Lambda](#).

La dimensione del payload si riferisce alla dimensione dei dati serializzati che persiste in un'operazione duratura. I dati vengono misurati in byte e la dimensione può variare a seconda del serializzatore utilizzato dall'operazione. Il payload di un'operazione potrebbe essere il risultato stesso dei completamenti riusciti o l'oggetto di errore serializzato se l'operazione non è riuscita.

## Funzionamento di base

Le operazioni di base sono gli elementi costitutivi fondamentali delle funzioni durature:

| Operation                | Tempistica dei checkpoint                 | Numero di operazioni | I dati sono persistiti                                           |
|--------------------------|-------------------------------------------|----------------------|------------------------------------------------------------------|
| Esecuzione               | Avviato                                   | 1                    | Dimensione del payload di input                                  |
| Esecuzione               | Completato ()<br>Succeeded/Failed/Stopped | 0                    | Dimensione del payload in uscita                                 |
| Fase                     | Retry/Succeeded/Failed                    | 1 + N tentativi      | Dimensione del payload restituita da ogni tentativo              |
| Attendi                  | Avviato                                   | 1                    | N/A                                                              |
| WaitForCondition         | Ogni tentativo di sondaggio               | 1 + N sondaggi       | Dimensione del payload restituita da ogni tentativo di sondaggio |
| Invocation-level Riprova | Avviato                                   | 1                    | Payload per l'oggetto di errore                                  |

## Operazioni di callback

Le operazioni di callback consentono alla funzione di interrompersi e attendere che i sistemi esterni forniscano l'input. Queste operazioni creano punti di controllo quando il callback viene creato e quando è completato:

| Operation                                       | Tempistica dei checkpoint | Numero di operazioni                              | I dati sono persistiti                                                                       |
|-------------------------------------------------|---------------------------|---------------------------------------------------|----------------------------------------------------------------------------------------------|
| CreateCallback                                  | Avviato                   | 1                                                 | N/A                                                                                          |
| Completamento del callback tramite chiamata API | Completato                | 0                                                 | Payload di callback                                                                          |
| WaitForCallback                                 | Avviato                   | 3 + N tentativi (contesto + callback + passaggio) | Payload restituiti dai tentativi di fase del mittente, più due copie del payload di callback |

## Operazioni composte

Le operazioni composte combinano più operazioni durevoli per gestire modelli di coordinamento complessi come l'esecuzione parallela, l'elaborazione di array e i contesti annidati:

| Operation | Tempistica dei checkpoint | Numero di operazioni                  | I dati sono persistiti                                                                                          |
|-----------|---------------------------|---------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| Parallelo | Avviato                   | 1 + N rami (1 contesto principale + N | Fino a due copie della dimensione del payload restituita da ciascuna filiale, più gli stati di ciascuna filiale |

| Operation           | Tempistica dei checkpoint | Numero di operazioni                                      | I dati sono persistiti                                                                                              |
|---------------------|---------------------------|-----------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
|                     |                           | contesti figlio)                                          |                                                                                                                     |
| Mappa               | Avviato                   | 1 + N filiali (1 contesto principale + N contesti figlio) | Fino a due copie della dimensione e del payload restituita da ogni iterazione, più gli stati di ciascuna iterazione |
| Aiutanti di Promise | Completato                | 1                                                         | Dimensione del payload restituita dalla promessa                                                                    |
| RunInChildContext   | Succeeded/Failed          | 1                                                         | Dimensione del payload restituita dal contesto secondario                                                           |

Per i contesti, ad esempio da `runInChildContext` o utilizzati internamente da operazioni composte, i risultati inferiori a 256 KB vengono sottoposti direttamente al checkpoint. I risultati più grandi non vengono memorizzati, ma vengono ricostruiti durante la riproduzione rielaborando le operazioni del contesto.

## Runtime supportati per funzioni durature

Sono disponibili funzioni durevoli per runtime gestiti selezionati e immagini di container OCI per una maggiore flessibilità delle versioni di runtime. È possibile creare funzioni durevoli per Python Node.js, Java e C# (.NET) utilizzando runtime gestiti direttamente nella console o programmaticamente tramite `infrastructure-as-code`.

### Runtime gestiti da Lambda

I seguenti runtime gestiti supportano funzioni durevoli quando si creano funzioni nella console Lambda o si utilizza il parametro `with the AWS CLI . --durable-config`

'{"ExecutionTimeout": 3600, "RetentionPeriodInDays": 7}' Per informazioni complete sui runtime Lambda, consulta Lambda runtimes. [???](#)

| Lingua    | Runtime     |
|-----------|-------------|
| Node.js   | nodejs22.x  |
| Node.js   | nodejs24.x  |
| Python    | python 3.13 |
| Python    | python 3.14 |
| Java      | java17      |
| Java      | java 21     |
| Java      | java25      |
| C# (.NET) | dotnet 8    |
| C# (.NET) | dotnet 10   |

#### Note

I runtime Lambda Node.js e Python includono l'SDK di esecuzione durevole per test e sviluppo. Tuttavia, consigliamo di includere l'SDK nel pacchetto di distribuzione per la produzione. Ciò garantisce la coerenza delle versioni ed evita potenziali aggiornamenti di runtime che potrebbero influire sul comportamento delle funzioni. Poiché Java e C# (.NET) sono linguaggi compilati, i runtime Lambda Java e .NET non includono l'SDK di esecuzione durevole, quindi deve essere incluso nel pacchetto di distribuzione.

## Node.js

Installa l'SDK nel tuo progetto: Node.js

```
npm install @aws/durable-execution-sdk-js
```

L'SDK supporta JavaScript e TypeScript. Per i TypeScript progetti, l'SDK include le definizioni dei tipi.

## Python

Installa l'SDK nel tuo progetto Python:

```
pip install aws-durable-execution-sdk-python
```

L'SDK Python utilizza metodi sincroni e non richiede `async/await`.

## Java

Aggiungi una dipendenza a: `pom.xml`

```
<dependency>
  <groupId>software.amazon.lambda.durable</groupId>
  <artifactId>aws-durable-execution-sdk-java</artifactId>
  <version>VERSION</version>
</dependency>
```

Installa l'SDK nel tuo progetto Java:

```
mvn install
```

L'SDK Java fornisce versioni sincrone e asincrone di ciascun metodo.

## C# (.NET)

Aggiungi l'SDK al tuo progetto .NET:

```
dotnet add package Amazon.Lambda.DurableExecution
```

.NET SDK espone operazioni durevoli tramite `IDurableContext`. Il gestore riceve la busta del servizio di esecuzione durevole e delega a `DurableFunction.WrapAsync`. Ogni organo operativo richiede un `CancellationToken`.

## Immagini di container

È possibile utilizzare funzioni durevoli con immagini dei contenitori per supportare versioni di runtime aggiuntive o configurazioni di runtime personalizzate. Le immagini dei contenitori consentono di utilizzare versioni di runtime non disponibili come runtime gestiti o di personalizzare l'ambiente di runtime.

Per creare una funzione duratura utilizzando un'immagine del contenitore:

1. Crea un Dockerfile basato su un'immagine base Lambda
2. Installa il Durable Execution SDK nel tuo contenitore
3. Crea e invia l'immagine del contenitore ad Amazon Elastic Container Registry
4. Crea la funzione Lambda dall'immagine del contenitore con l'esecuzione durevole abilitata

## Esempio di contenitore

Crea un Dockerfile:

### Python

Crea un Dockerfile per Python 3.11:

```
FROM public.ecr.aws/lambda/python:3.11

# Copy requirements file
COPY requirements.txt ${LAMBDA_TASK_ROOT}/

# Install dependencies including durable SDK
RUN pip install -r requirements.txt

# Copy function code
COPY lambda_function.py ${LAMBDA_TASK_ROOT}/
```

```
# Set the handler
CMD [ "lambda_function.handler" ]
```

Crea un file: requirements.txt

```
aws-durable-execution-sdk-python
```

## Java

Crea un Dockerfile per Java 25:

```
FROM --platform=linux/amd64 public.ecr.aws/lambda/java:25

# Install Maven
RUN dnf install -y maven

WORKDIR /var/task

# Copy Maven configuration and source code
COPY pom.xml .
COPY src ./src

# Build
RUN mvn clean package -DskipTests

# Move JAR to lib directory
RUN mv target/*.jar lib/

# Set the handler
CMD ["src.path.to.lambdaFunction::handler"]
```

Crea e invia l'immagine:

```
# Build the image
```

```
docker build -t my-durable-function .

# Tag for ECR
docker tag my-durable-function:latest 123456789012.dkr.ecr.us-east-1.amazonaws.com/my-durable-function:latest

# Push to ECR
docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com/my-durable-function:latest
```

Crea la funzione con l'esecuzione duratura abilitata:

```
aws lambda create-function \
  --function-name myDurableFunction \
  --package-type Image \
  --code ImageUri=123456789012.dkr.ecr.us-east-1.amazonaws.com/my-durable-function:latest \
  --role arn:aws:iam::123456789012:role/lambda-execution-role \
  --durable-config '{"ExecutionTimeout": 3600, "RetentionPeriodInDays": 7}'
```

Per ulteriori informazioni sull'utilizzo delle immagini dei container con Lambda, vedi [Creazione di immagini di container Lambda](#) nella Lambda Developer Guide.

## Considerazioni sul runtime

**Gestione delle versioni SDK:** per mantenere il pieno controllo sulle dipendenze ed evitare possibili problemi di disallineamento delle versioni, ti consigliamo di aggiungere tutte le dipendenze della tua funzione al tuo pacchetto di distribuzione, anche se le versioni di esse sono incluse nel runtime Lambda per impostazione predefinita. Ciò include l'SDK di esecuzione durevole. Blocca l'SDK di esecuzione durevole su una versione principale nel tuo file di dipendenza. Una nuova versione principale può introdurre modifiche che possono causare errori nelle esecuzioni in corso. Utilizza versioni numerate o alias per funzioni durevoli di produzione anziché per garantire che le modifiche \$LATEST alla versione SDK non influiscano sulle esecuzioni in corso.

**Aggiornamenti di runtime:** AWS aggiorna i runtime gestiti per includere patch di sicurezza e correzioni di bug. Questi aggiornamenti potrebbero includere nuove versioni dell'SDK. Per evitare comportamenti imprevisti, includi l'SDK nel pacchetto di distribuzione ed esegui un test approfondito prima di distribuirlo in produzione.



Dimensione dell'immagine del contenitore: le immagini del contenitore hanno una dimensione massima non compressa di 10 GB. L'SDK di esecuzione durevole aggiunge dimensioni minime all'immagine. Ottimizza il tuo contenitore utilizzando build in più fasi e rimuovendo le dipendenze non necessarie.

Prestazioni di avvio a freddo: le immagini dei container potrebbero avere tempi di avvio a freddo più lunghi rispetto ai runtime gestiti. L'SDK di esecuzione durevole ha un impatto minimo sulle prestazioni di avvio a freddo. Usa la concorrenza predisposta se la latenza di avvio a freddo è fondamentale per la tua applicazione.

## Richiamo di funzioni Lambda durevoli

Le funzioni Lambda durevoli supportano gli stessi metodi di invocazione delle funzioni Lambda standard. È possibile richiamare funzioni durevoli in modo sincrono, asincrono o tramite mappature delle sorgenti degli eventi. Il processo di invocazione è identico alle funzioni standard, ma le funzioni durevoli forniscono funzionalità aggiuntive per le esecuzioni di lunga durata e la gestione automatica dello stato.

### Metodi di invocazione

Invocazione sincrona: richiama una funzione durevole e attendi la risposta. Le chiamate sincrone sono limitate dalla Lambda a 15 minuti (o meno, a seconda della funzione configurata e del timeout di esecuzione). Utilizza la chiamata sincrona quando hai bisogno di risultati immediati o quando esegui l'integrazione con servizi che prevedono una risposta. APIs È possibile utilizzare le operazioni di attesa per un calcolo efficiente senza interrompere il chiamante: la chiamata attende il completamento dell'intera esecuzione durevole. [Per avvisi di esecuzione idempotenti, utilizzate il parametro del nome dell'esecuzione come descritto in Idempotency.](#)

```
aws lambda invoke \  
  --function-name my-durable-function:1 \  
  --cli-binary-format raw-in-base64-out \  
  --payload '{"orderId": "12345"}' \  
  response.json
```

Richiamata asincrona: mette in coda un evento per l'elaborazione senza attendere una risposta. Lambda mette l'evento in coda e ritorna immediatamente. Le chiamate asincrone supportano durate di esecuzione fino a 1 anno. Utilizza la chiamata asincrona per scenari o quando l'elaborazione può

avvenire in background. fire-and-forget [Per avvisi di esecuzione idempotenti, utilizzate il parametro del nome dell'esecuzione come descritto in Idempotenza.](#)

```
aws lambda invoke \  
  --function-name my-durable-function:1 \  
  --invocation-type Event \  
  --cli-binary-format raw-in-base64-out \  
  --payload '{"orderId": "12345"}' \  
  response.json
```

Mappature delle sorgenti degli eventi: configura Lambda per richiamare automaticamente la tua funzione duratura quando i record sono disponibili da servizi basati su stream o code come Amazon SQS, Kinesis o DynamoDB. Le mappature delle sorgenti degli eventi analizzano l'origine dell'evento e richiamano la funzione con batch di record. [Per i dettagli sull'utilizzo delle mappature delle sorgenti degli eventi con funzioni durevoli, inclusi i limiti di durata dell'esecuzione, consulta Mappature delle sorgenti degli eventi con funzioni durevoli.](#)

[Per dettagli completi su ciascun metodo di invocazione, vedere invocazione sincrona e chiamata asincrona.](#)

#### Note

Le funzioni durevoli supportano dead-letter queues (DLQs) per la gestione degli errori, ma non supportano le destinazioni Lambda. Configura un DLQ per acquisire i record delle chiamate non riuscite.

## ARNs Requisito qualificato

Le funzioni durevoli richiedono identificatori qualificati per la chiamata. È necessario richiamare funzioni durevoli utilizzando un numero di versione, un alias o. \$LATEST È possibile utilizzare un ARN completo o un nome di funzione con version/alias suffisso. Non è possibile utilizzare un identificatore non qualificato (senza una versione o un suffisso di alias).

Invocazioni valide:

```
# Using full ARN with version number
arn:aws:lambda:us-east-1:123456789012:function:my-durable-function:1

# Using full ARN with alias
arn:aws:lambda:us-east-1:123456789012:function:my-durable-function:prod

# Using full ARN with $LATEST
arn:aws:lambda:us-east-1:123456789012:function:my-durable-function:$LATEST

# Using function name with version number
my-durable-function:1

# Using function name with alias
my-durable-function:prod
```

### Invocazioni non valide:

```
# Unqualified ARN (not allowed)
arn:aws:lambda:us-east-1:123456789012:function:my-durable-function

# Unqualified function name (not allowed)
my-durable-function
```

Questo requisito garantisce che le esecuzioni durevoli rimangano coerenti per tutto il loro ciclo di vita. Quando inizia un'esecuzione durevole, viene associata alla versione della funzione specifica. Se la funzione viene messa in pausa e riprende ore o giorni dopo, Lambda richiama la stessa versione che ha avviato l'esecuzione, garantendo la coerenza del codice nell'intero flusso di lavoro.

### Best practice

Utilizza versioni numerate o alias per funzioni durevoli di produzione anziché `$LATEST`. Le versioni numerate sono immutabili e supportano la riproduzione deterministica. Facoltativamente, gli alias forniscono un riferimento stabile che è possibile aggiornare per indirizzare a nuove versioni senza modificare il codice di invocazione. Quando si aggiorna un alias, le nuove esecuzioni utilizzano la nuova versione, mentre le esecuzioni in corso continuano con la versione originale. È possibile utilizzarla `$LATEST` per la prototipazione o per abbreviare i tempi di implementazione durante lo sviluppo, tenendo presente che le esecuzioni potrebbero non essere riprodotte correttamente (o addirittura fallire) se il codice sottostante cambia durante l'esecuzione.

## Comprendere il ciclo di vita dell'esecuzione

Quando richiami una funzione duratura, Lambda crea un'esecuzione durevole che può includere più chiamate di funzioni:

1. **Richiamata iniziale:** la richiesta di invocazione crea una nuova esecuzione durevole. Lambda assegna un ID di esecuzione univoco e avvia l'elaborazione.
2. **Esecuzione e checkpoint:** man mano che la funzione esegue operazioni durature, l'SDK crea checkpoint che tengono traccia dei progressi.
3. **Sospensione (se necessaria):** se la funzione utilizza attese durature, ad esempio `or`, `wait` o ripetizioni automatiche dei passaggi `waitForCallback`, Lambda sospende l'esecuzione e interrompe l'addebito del tempo di elaborazione.
4. **Ripresa:** quando è il momento di riprendere (anche dopo nuovi tentativi), Lambda richiama nuovamente la funzione. L'SDK riproduce il registro del checkpoint e continua dal punto in cui l'esecuzione si è interrotta.
5. **Completamento:** quando la funzione restituisce un risultato finale o genera un errore non gestito, l'esecuzione durevole viene completata.

Per le chiamate sincrone, il chiamante attende il completamento dell'intera esecuzione durevole, comprese le eventuali operazioni di attesa. Se l'esecuzione supera il timeout di chiamata (15 minuti o meno), la chiamata scade. Per le chiamate asincrone, Lambda ritorna immediatamente e l'esecuzione continua indipendentemente. Usa l'esecuzione durevole APIs per tenere traccia dello stato di esecuzione e recuperare i risultati finali.

## Richiamo dal codice dell'applicazione

Utilizzate il AWS SDKs per richiamare funzioni durevoli dal codice dell'applicazione. Il processo di invocazione è identico alle funzioni standard:

### TypeScript

```
import { LambdaClient, InvokeCommand } from '@aws-sdk/client-lambda';

const client = new LambdaClient({});

// Synchronous invocation
const response = await client.send(new InvokeCommand({
```

```
    FunctionName: 'arn:aws:lambda:us-east-1:123456789012:function:my-durable-  
function:1',  
    Payload: JSON.stringify({ orderId: '12345' })  
  }));  
  
const result = JSON.parse(Buffer.from(response.Payload!).toString());  
  
// Asynchronous invocation  
await client.send(new InvokeCommand({  
  FunctionName: 'arn:aws:lambda:us-east-1:123456789012:function:my-durable-  
function:1',  
  InvocationType: 'Event',  
  Payload: JSON.stringify({ orderId: '12345' })  
}));
```

## Python

```
import boto3  
import json  
  
client = boto3.client('lambda')  
  
# Synchronous invocation  
response = client.invoke(  
    FunctionName='arn:aws:lambda:us-east-1:123456789012:function:my-durable-  
function:1',  
    Payload=json.dumps({'orderId': '12345'})  
)  
  
result = json.loads(response['Payload'].read())  
  
# Asynchronous invocation  
client.invoke(  
    FunctionName='arn:aws:lambda:us-east-1:123456789012:function:my-durable-  
function:1',  
    InvocationType='Event',  
    Payload=json.dumps({'orderId': '12345'})  
)
```

## Invocazioni concatenate

Le funzioni durevoli possono richiamare altre funzioni durevoli e non durevoli utilizzando l'operazione `da.invoke DurableContext`. Questo crea una chiamata concatenata in cui la funzione chiamante attende (sospende) il completamento della funzione invocata:

### TypeScript

```
export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    // Invoke another durable function and wait for result
    const result = await context.invoke(
      'process-order',
      'arn:aws:lambda:us-east-1:123456789012:function:order-processor:1',
      { orderId: event.orderId }
    );

    return { statusCode: 200, body: JSON.stringify(result) };
  }
);
```

### Python

```
@durable_execution
def handler(event, context: DurableContext):
    # Invoke another durable function and wait for result
    result = context.invoke(
        'arn:aws:lambda:us-east-1:123456789012:function:order-processor:1',
        {'orderId': event['orderId']},
        name='process-order'
    )

    return {'statusCode': 200, 'body': json.dumps(result)}
```

Le invocazioni concatenate creano un checkpoint nella funzione chiamante. Se la funzione chiamante viene interrotta, riprende dal checkpoint con il risultato della funzione richiamata, senza richiamare nuovamente la funzione.

 Note

Le chiamate concatenate tra più account non sono supportate. La funzione richiamata deve trovarsi nello stesso AWS account della funzione chiamante.

## Mappature delle sorgenti di eventi con funzioni durevoli

Le funzioni durevoli funzionano con tutte le mappature delle sorgenti di eventi Lambda. Configura le mappature delle sorgenti di eventi per le funzioni durevoli nello stesso modo in cui le configuri per le funzioni standard. Le mappature delle sorgenti di eventi eseguono automaticamente il polling di sorgenti di eventi come Amazon SQS, Kinesis e DynamoDB Streams e richiamano la tua funzione con batch di record.

Le mappature delle sorgenti di eventi sono utili per funzioni durevoli che elaborano flussi o code con flussi di lavoro complessi in più fasi. Ad esempio, puoi creare una funzione duratura che elabori i messaggi Amazon SQS con tentativi, chiamate API esterne e approvazioni umane.

In che modo le mappature delle sorgenti degli eventi richiamano funzioni durevoli

Le mappature delle sorgenti di eventi richiamano le funzioni durevoli in modo sincrono, aspettando il completamento dell'esecuzione durevole completa prima di elaborare il batch successivo o contrassegnare i record come elaborati. Se il tempo totale di esecuzione durevole supera i 15 minuti, l'esecuzione scade e non riesce. La mappatura dell'origine dell'evento riceve un'eccezione di timeout e la gestisce in base alla configurazione del nuovo tentativo.

### Limite di esecuzione di 15 minuti

Quando le funzioni durevoli vengono richiamate dalle mappature delle sorgenti degli eventi, la durata totale dell'esecuzione durevole non può superare i 15 minuti. Questo limite si applica all'intera esecuzione duratura dall'inizio al completamento, non solo alle singole chiamate di funzioni.

Questo limite di 15 minuti è separato dal timeout della funzione Lambda (anch'esso massimo di 15 minuti). Il timeout della funzione controlla la durata di ogni singola chiamata, mentre il timeout di esecuzione durevole controlla il tempo totale trascorso dall'inizio al completamento dell'esecuzione.

Scenari di esempio:

- **Valido:** una funzione durevole elabora un messaggio Amazon SQS in tre passaggi, ciascuno dei quali richiede 2 minuti, quindi attende 5 minuti prima di completare un passaggio finale. Tempo totale di esecuzione: 11 minuti. Funziona perché il totale è inferiore a 15 minuti.

- Non valido: una funzione durevole elabora un messaggio Amazon SQS, completa l'elaborazione iniziale in 2 minuti, quindi attende 20 minuti per una richiamata esterna prima di essere completata. Tempo totale di esecuzione: 22 minuti. Questo supera il limite di 15 minuti e fallisce.
- Non valido: una funzione durevole elabora un record Kinesis con più operazioni di attesa per un totale di 30 minuti tra i passaggi. Anche se ogni singola chiamata viene completata rapidamente, il tempo totale di esecuzione supera i 15 minuti.

#### Important

Configura il tuo timeout di esecuzione duraturo a 15 minuti o meno quando usi le mappature delle sorgenti degli eventi, altrimenti la creazione della mappatura dell'origine dell'evento non riesce. Se il flusso di lavoro richiede tempi di esecuzione più lunghi, utilizza il modello di funzione intermedio descritto di seguito.

## Configurazione delle mappature delle sorgenti degli eventi

Configura le mappature delle sorgenti di eventi per funzioni durature utilizzando la console Lambda o gli SDK. AWS CLI AWS Tutte le proprietà standard di mappatura delle sorgenti di eventi si applicano alle funzioni durevoli:

```
aws lambda create-event-source-mapping \  
  --function-name arn:aws:lambda:us-east-1:123456789012:function:my-durable-function:1 \  
  \  
  --event-source-arn arn:aws:sqs:us-east-1:123456789012:my-queue \  
  --batch-size 10 \  
  --maximum-batching-window-in-seconds 5
```

Ricorda di utilizzare un ARN qualificato (con numero di versione o alias) quando configuri le mappature delle sorgenti di eventi per funzioni durevoli.

## Gestione degli errori con le mappature delle sorgenti degli eventi

Le mappature delle sorgenti degli eventi forniscono una gestione degli errori integrata che funziona con funzioni durevoli:



- **Comportamento dei tentativi:** se la chiamata iniziale ha esito negativo, la mappatura dell'origine dell'evento riprova in base alla configurazione del tentativo. Configura il numero massimo di tentativi e intervalli tra i tentativi in base alle tue esigenze.
- **Dead-letter code:** configura una coda di lettere morte per acquisire i record che hanno esito negativo dopo tutti i tentativi. Ciò impedisce la perdita di messaggi e consente l'ispezione manuale dei record non riusciti.
- **Errori parziali dei batch:** per Amazon SQS e Kinesis, utilizza la segnalazione parziale degli errori dei batch per elaborare i record singolarmente e riprovare solo i record non riusciti.
- **Bisect on error:** per Kinesis e DynamoDB Streams, abilita bisect on error per suddividere i batch non riusciti e isolare i record problematici.

#### Note

Le funzioni Durable supportano le code di lettere morte (DLQ) per la gestione degli errori, ma non supportano le destinazioni Lambda. Configura un DLQ per acquisire i record delle chiamate non riuscite.

Per informazioni complete sulla gestione degli errori di mappatura delle sorgenti degli eventi, vedere [mappature delle sorgenti degli eventi](#).

### Utilizzo di una funzione intermedia per flussi di lavoro di lunga durata

Se il completamento del flusso di lavoro richiede più di 15 minuti, utilizza una funzione Lambda standard intermedia tra la mappatura dell'origine dell'evento e la funzione durevole. La funzione intermediaria riceve gli eventi dalla mappatura dell'origine degli eventi e richiama la funzione durable in modo asincrono, rimuovendo il limite di esecuzione di 15 minuti.

Questo modello separa il modello di invocazione sincrona della mappatura dell'origine degli eventi dal modello di esecuzione a lungo termine della funzione durable. La mappatura dell'origine degli eventi richiama la funzione intermedia, che ritorna rapidamente dopo l'avvio dell'esecuzione durevole. La funzione durable viene quindi eseguita in modo indipendente per tutto il tempo necessario (fino a 1 anno).

### Architecture

Il modello di funzione intermedio utilizza tre componenti:

1. Mappatura dell'origine dell'evento: esegue il polling della sorgente dell'evento (Amazon SQS, Kinesis, DynamoDB Streams) e richiama la funzione intermedia in modo sincrono con batch di record.
2. Funzione intermedia: una funzione Lambda standard che riceve eventi dalla mappatura della sorgente dell'evento, convalida e trasforma i dati se necessario e richiama la funzione durable in modo asincrono. Questa funzione viene completata rapidamente (in genere meno di 1 secondo) e restituisce il controllo alla mappatura della sorgente dell'evento.
3. Funzione durevole: elabora l'evento con una logica complessa a più fasi che può essere eseguita per lunghi periodi. Richiamato in modo asincrono, quindi non è vincolato dal limite di 15 minuti.

## Implementazione

La funzione intermediaria riceve l'intero evento dalla mappatura della sorgente dell'evento e richiama la funzione durable in modo asincrono. Utilizzate il parametro `execution name` per assicurarvi che l'esecuzione venga avviata in modo idempotente, impedendo l'elaborazione duplicata se la mappatura della sorgente dell'evento riprova:

## TypeScript

```
import { LambdaClient, InvokeCommand } from '@aws-sdk/client-lambda';
import { SQSEvent } from 'aws-lambda';
import { createHash } from 'crypto';

const lambda = new LambdaClient({});

export const handler = async (event: SQSEvent) => {
  // Invoke durable function asynchronously with execution name
  await lambda.send(new InvokeCommand({
    FunctionName: 'arn:aws:lambda:us-east-1:123456789012:function:my-durable-
function:1',
    InvocationType: 'Event',
    Payload: JSON.stringify({
      executionName: event.Name,
      event: event
    })
  }));

  return { statusCode: 200 };
};
```

## Python

```
import boto3
import json
import hashlib

lambda_client = boto3.client('lambda')

def handler(event, context):
    # Invoke durable function asynchronously with execution name
    lambda_client.invoke(
        FunctionName='arn:aws:lambda:us-east-1:123456789012:function:my-durable-
function:1',
        InvocationType='Event',
        Payload=json.dumps({
            'executionName': execution_name,
            'event': event["name"]
        })
    )

    return {'statusCode': 200}
```

Per verificare l'idempotenza della funzione intermediaria stessa, utilizzate [Powertools per evitare invocazioni duplicate della funzione durable se la AWS Lambda](#) mappatura della sorgente dell'evento riprova la funzione intermediaria.

La funzione durable riceve il payload con il nome dell'esecuzione ed elabora tutti i record con una logica a lungo termine:

## TypeScript

```
import { withDurableExecution, DurableContext } from '@aws/durable-execution-sdk-
js';

export const handler = withDurableExecution(
  async (payload: any, context: DurableContext) => {
    const sqsEvent = payload.event;
```

```

// Process each record with complex, multi-step logic
const results = await context.map(
  sqsEvent.Records,
  async (ctx, record) => {
    const validated = await ctx.step('validate', async () => {
      return validateOrder(JSON.parse(record.body));
    });

    // Wait for external approval (could take hours or days)
    const approval = await ctx.waitForCallback(
      'approval',
      async (callbackId) => {
        await requestApproval(callbackId, validated);
      },
      { timeout: { hours: 48 } }
    );

    // Complete processing
    return await ctx.step('complete', async () => {
      return completeOrder(validated, approval);
    });
  }
);

return { statusCode: 200, processed: results.getResults().length };
};

```

## Python

```

from aws_durable_execution_sdk_python import durable_execution, DurableContext
from aws_durable_execution_sdk_python.config import Duration, WaitForCallbackConfig
from collections.abc import Sequence
import json

def validate_order(order_data: dict) -> dict:
    """Validate order data - always passes."""
    return order_data

def request_approval(callback_id: str, validated_order: dict) -> None:

```

```

    """Request approval for the order - always passes."""
    pass

def complete_order(validated_order: dict, approval_result: str) -> dict:
    """Complete the order processing - always passes."""
    return validated_order

@durable_execution
def lambda_handler(payload, context: DurableContext):
    sqs_event = payload['event']

    def process_record(
        ctx: DurableContext,
        record: dict,
        index: int,
        items: Sequence[dict]
    ) -> dict:
        validated = ctx.step(
            lambda _: validate_order(json.loads(record['body'])),
            name=f'validate-{index}'
        )

        approval = ctx.wait_for_callback(
            submitter=lambda callback_id, wait_ctx: request_approval(callback_id,
validated),
            name=f'approval-{index}',
            config=WaitForCallbackConfig(timeout=Duration.from_seconds(172800))
        )

        return ctx.step(
            lambda _: complete_order(validated, approval),
            name=f'complete-{index}'
        )

    results = context.map(
        inputs=sqs_event['Records'],
        func=process_record,
        name='process-records'
    )

    return {
        'statusCode': 200,
        'started': results.started_count,
        'completed': results.success_count,

```

```
'failed': results.failure_count,  
'total': results.total_count  
}
```

## Considerazioni chiave

Questo modello rimuove il limite di esecuzione di 15 minuti disaccoppiando la mappatura della sorgente dell'evento dall'esecuzione durevole. La funzione intermediaria ritorna immediatamente dopo l'avvio dell'esecuzione durevole, consentendo alla mappatura della sorgente dell'evento di continuare l'elaborazione. La funzione durable viene quindi eseguita in modo indipendente per tutto il tempo necessario.

La funzione intermediaria ha successo quando richiama la funzione durevole, non quando l'esecuzione durevole viene completata. Se l'esecuzione durevole fallisce in un secondo momento, la mappatura dell'origine dell'evento non verrà riprovata perché il batch è già stato elaborato correttamente. Implementa la gestione degli errori nella funzione durable e configura le code di lettere morte per le esecuzioni non riuscite.

Utilizzate il parametro `execution name` per assicurarvi che l'esecuzione abbia inizio in modo idempotente. Se la mappatura dell'origine dell'evento riprova la funzione intermediaria, la funzione durable non avvierà un'esecuzione duplicata perché il nome dell'esecuzione esiste già.

## Fonti di eventi supportate

Le funzioni Durable supportano tutte le sorgenti di eventi Lambda che utilizzano le mappature delle sorgenti di eventi:

- code Amazon SQS (standard e FIFO)
- Flussi Kinesis
- DynamoDB Streams
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Self-managed Apache Kafka
- Amazon MQ (ActiveMQ e RabbitMQ)
- Flussi di modifiche di Amazon DocumentDB

Tutti i tipi di sorgenti di eventi sono soggetti al limite di esecuzione duratura di 15 minuti quando si richiamano funzioni durevoli.

## Riprova per le funzioni durevoli Lambda

Le funzioni durevoli forniscono funzionalità di riprova automatica che rendono le applicazioni resistenti ai guasti transitori. L'SDK gestisce i nuovi tentativi a due livelli: i tentativi successivi per errori della logica aziendale e i tentativi di backend per i guasti dell'infrastruttura.

### Ripetimenti in fasi

Quando si verifica un'eccezione non rilevata all'interno di un passaggio, l'SDK riprova automaticamente il passaggio in base alla strategia di riprova configurata. I tentativi di fase sono operazioni controllate che consentono all'SDK di sospendere l'esecuzione e riprenderla in un secondo momento senza perdere i progressi.

### Comportamento dei nuovi tentativi

La tabella seguente descrive come l'SDK gestisce le eccezioni in pochi passaggi:

| Scenario                                                        | Cosa succede                                                                                                                                                  | Impatto della misurazione                       |
|-----------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| Eccezione in corso con i tentativi di nuovo tentativo rimanenti | L'SDK crea un checkpoint per il nuovo tentativo e sospende la funzione. Alla chiamata successiva, il passaggio riprova con il ritardo di backoff configurato. | 1 operazione + dimensione del payload di errore |
| Eccezione in fase senza tentativi di nuovo tentativo rimanenti  | Il passaggio non riesce e genera un'eccezione. Se il codice del gestore non rileva questa eccezione, l'intera esecuzione ha esito negativo.                   | 1 operazione + dimensione del payload di errore |

Quando un passaggio deve essere riprovato, l'SDK verifica lo stato del tentativo ed esce dalla chiamata Lambda se nessun altro lavoro è in esecuzione. Ciò consente all'SDK di implementare ritardi di backoff senza consumare risorse di calcolo. La funzione riprende automaticamente dopo il periodo di backoff.

## Configurazione delle strategie di ripetizione dei passaggi

Configura le strategie di ripetizione per controllare il modo in cui i passaggi gestiscono gli errori. È possibile specificare il numero massimo di tentativi, gli intervalli di backoff e le condizioni per riprovare. Per un riferimento completo agli helper, alle preimpostazioni e alle strategie personalizzate per i tentativi, consultate [Retries](#) nella documentazione di Durable Execution SDK.

### Eccezioni al di fuori dei passaggi

Quando si verifica un'eccezione non rilevata nel codice del gestore ma al di fuori di qualsiasi passaggio, l'SDK contrassegna l'esecuzione come fallita. Ciò garantisce che gli errori nella logica dell'applicazione vengano acquisiti e segnalati correttamente.

| Scenario                                                            | Cosa succede                                                                                                          | Impatto della misurazione        |
|---------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|----------------------------------|
| Eccezione nel codice del gestore al di fuori di qualsiasi passaggio | L'SDK contrassegna l'esecuzione come FALLITA e restituisce l'errore. L'eccezione non viene ritentata automaticamente. | Dimensione del payload di errore |

Per abilitare il ritentativo automatico per il codice soggetto a errori, racchiudi in un passaggio una strategia di riprova. I passaggi prevedono un nuovo tentativo automatico con backoff configurabile, mentre il codice esterno ha esito negativo immediatamente.

### Riprova di richiamo

I tentativi a livello di chiamata vengono gestiti in modo diverso a seconda di come si tenta di richiamare la funzione Lambda durable. La tabella seguente descrive in che modo i diversi tipi di chiamata possono influenzare i tentativi a livello di chiamata.

| Tipo di invocazione  | Cosa succede                                                                                                                                                                                                                                                         |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Invocazione sincrona | Lambda non ritenta automaticamente l'invocazione in caso di errore durante l'esecuzione di una funzione durevole. I tentativi in caso di errore di richiamo dipendono dall'origine dell'invocazione sincrona. Ad esempio, utilizzando l' AWS SDK, e per impostazione |



| Tipo di invocazione        | Cosa succede                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                            | predefinita vengono ritentati automaticamente. InternalFailure ThrottlingException                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Invocazione asincrona      | Se l'esecuzione di una funzione durevole fallisce (ad esempio, entra nello stato FAILED, STOPPED o TIMED_OUT), Lambda non riprova l'esecuzione. Ciò è diverso dalle funzioni Lambda standard, in cui Lambda riprova la funzione in caso di errori di chiamata asincrona. L'MaximumRetryAttempts impostazione per le chiamate asincrone non si applica alle esecuzioni durature. Se si configura una coda di lettere morte (DLQ) per la funzione, Lambda invia l'evento di attivazione al DLQ.                                                                                                                                                                                                        |
| ESM (Event Source Mapping) | Per impostazione predefinita, Lambda riprova l'intero batch finché non ha esito positivo. Per le origini di flusso (DynamoDB e Kinesis), è possibile configurare il numero massimo di tentativi che Lambda effettua quando la funzione restituisce un errore. Vedi la mappatura delle sorgenti <a href="#">degli eventi in batch</a> . Per Amazon SQS ESM, puoi configurare il numero massimo di tentativi tramite un DLQ sulla coda Amazon SQS originale. <a href="#">Consulta la</a> sezione Configurazione di Amazon SQS ESM. Come best practice, potresti prendere in considerazione un DLQ a livello di funzione e Lambda invia l'evento di attivazione dell'errore al DLQ. <a href="#">???</a> |

| Tipo di invocazione | Cosa succede                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Trigger diretto     | <p>Dipende dal «Trigger». Ad esempio, Lambda elabora le funzioni attivate dalle notifiche di eventi di Amazon S3 in modo asincrono. Vedi <a href="#">Elaborare le notifiche di eventi Amazon SQS con Lambda</a>. Lambda elabora le funzioni attivate dalle notifiche di eventi di Amazon SNS, in modo asincrono. Vedi <a href="#">Richiamare le funzioni Lambda con le notifiche Amazon SNS</a>. Il comportamento di ripetizione dell'invocazione asincrona è riportato sopra nella voce della tabella «Invocazione asincrona». Se Amazon SNS non è in grado di raggiungere Lambda o il messaggio viene rifiutato, Amazon SNS riprova a intervalli crescenti per diverse ore. Per ulteriori informazioni, consulta <a href="#">Affidabilità</a> nelle domande frequenti su Amazon SNS. API Gateway richiama Lambda in modo sincrono e restituisce la risposta di errore autentica al richiedente. Vedi i tentativi di richiamo. <a href="#">???</a> Il comportamento dei tentativi di chiamata sincrona è riportato sopra nella voce della tabella «Richiamo sincrono». Vedi ogni trigger diretto per maggiori dettagli.</p> |

## Tentativi nel backend

I tentativi di backend si verificano quando Lambda rileva guasti all'infrastruttura, errori di runtime o quando l'SDK non è in grado di comunicare con il servizio di esecuzione durevole. Lambda riprova automaticamente questi errori per aiutare le funzioni durevoli a riprendersi da problemi transitori dell'infrastruttura.

## Scenari di riprova nel backend

Lambda riprova automaticamente la funzione quando incontra i seguenti scenari:

- Errori interni del servizio: quando Lambda o il servizio di esecuzione durevole restituiscono un errore 5xx, che indica un problema temporaneo del servizio.
- Limitazione: quando la funzione è limitata a causa dei limiti di concorrenza o delle quote di servizio.

- **Timeout:** quando l'SDK non può raggiungere il servizio di esecuzione durevole entro il periodo di timeout.
- **Errori di inizializzazione della sandbox:** quando Lambda non è in grado di inizializzare l'ambiente di esecuzione.
- **Errori di runtime:** quando il runtime di Lambda rileva errori esterni al codice della funzione, come errori di esaurimento della memoria o arresti anomali del processo.
- **Errori del token di checkpoint non valido:** quando il token del checkpoint non è più valido, in genere a causa di modifiche allo stato del servizio.

La tabella seguente descrive come l'SDK gestisce questi scenari:

| Scenario                                                                                                                                     | Cosa succede                                                                                                                               | Impatto della misurazione                                     |
|----------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|
| Errore di runtime esterno a durable handler (OOM, timeout, crash)                                                                            | Lambda riprova automaticamente l'invocazione. L'SDK esegue il replay dall'ultimo checkpoint, saltando i passaggi completati.               | Dimensione del payload di errore + 1 operazione per tentativo |
| Errore di servizio (5xx) o timeout durante la chiamata a/API <code>CheckpointDurableExecution GetDurableExecutionState</code>                | Lambda riprova automaticamente l'invocazione. L'SDK esegue i replay dall'ultimo checkpoint.                                                | Dimensione del payload di errore + 1 operazione per tentativo |
| Throttling (429) o token di checkpoint non valido durante la chiamata a/API <code>CheckpointDurableExecution GetDurableExecutionState</code> | Lambda riprova automaticamente l'invocazione con un backoff esponenziale. L'SDK esegue i replay dall'ultimo checkpoint.                    | Dimensione del payload di errore + 1 operazione per tentativo |
| Errore del client (4xx, tranne 429 e token non valido) quando a/API <code>CheckpointDurableExecution</code>                                  | L'SDK contrassegna l'esecuzione come FALLITA. Non si verifica un nuovo tentativo automatico perché l'errore indica un problema permanente. | Dimensione del payload errato                                 |

| Scenario                               | Cosa succede | Impatto della misurazione |
|----------------------------------------|--------------|---------------------------|
| Execution GetDurabl<br>eExecutionState |              |                           |

I tentativi di backend utilizzano il backoff esponenziale e continuano fino al completamento della funzione o al raggiungimento del timeout di esecuzione. Durante la riproduzione, l'SDK salta i checkpoint completati e continua l'esecuzione dall'ultima operazione riuscita, assicurando che la funzione non riesegua il lavoro completato.

## Riprova le best practice

Segui queste best practice per configurare le strategie di ripetizione dei tentativi:

- Configura strategie di ripetizione esplicite: non affidarti al comportamento predefinito dei tentativi in produzione. Configura strategie di ripetizione esplicite con il numero massimo di tentativi e intervalli di backoff appropriati per il tuo caso d'uso.
- Usa tentativi condizionali: implementa la `shouldRetry` logica per riprovare solo gli errori transitori (limiti di frequenza, timeout) e fallire rapidamente in caso di errori permanenti (errori di convalida, non trovati).
- Imposta il numero massimo di tentativi appropriato: equilibrio tra resilienza e tempo di esecuzione. Troppi tentativi possono ritardare il rilevamento degli errori, mentre un numero insufficiente può causare errori non necessari.
- Utilizza il backoff esponenziale: il backoff esponenziale riduce il carico sui servizi a valle e aumenta la probabilità di ripristino in caso di guasti transitori.
- Raccogli il codice soggetto a errori in passaggi: il codice esterno non può essere ritentato automaticamente. Raccogli le chiamate API esterne, le interrogazioni al database e altre operazioni soggette a errori in fasi con strategie di ripetizione.
- Monitora le metriche dei nuovi tentativi: tieni traccia delle operazioni di ripetizione dei tentativi e degli errori di esecuzione in Amazon CloudWatch per identificare i modelli e ottimizzare le strategie di riprova.

## Idempotenza

Le funzioni durevoli forniscono un'idempotenza incorporata per l'avvio dell'esecuzione tramite i nomi delle esecuzioni. Quando si fornisce un nome di esecuzione, Lambda lo utilizza per evitare

esecuzioni duplicate e consentire tentativi sicuri delle richieste di invocazione. Per impostazione predefinita, i passaggi hanno una semantica di esecuzione almeno una volta. Durante la riproduzione, l'SDK restituisce risultati controllati senza rieseguire i passaggi completati, ma la logica aziendale deve essere idempotente per gestire potenziali tentativi prima del completamento.

### Note

Le mappature delle sorgenti degli eventi Lambda (ESM) non supportano l'idempotenza al momento del lancio. Pertanto, ogni invocazione (inclusi i tentativi) avvia una nuova esecuzione duratura. Per garantire un'esecuzione idempotente con le mappature delle sorgenti degli eventi, implementate la logica di idempotenza nel codice della funzione, ad esempio con [Powertools for, AWS Lambda](#) oppure utilizzate una normale funzione Lambda come proxy (dispatcher) per richiamare una funzione durevole con una chiave di idempotenza (parametro del nome dell'esecuzione).

## Nomi di esecuzione

È possibile fornire un nome di esecuzione quando si richiama una funzione durevole. Il nome di esecuzione funge da chiave di idempotenza, consentendo di riprovare in sicurezza le richieste di richiamo senza creare esecuzioni duplicate. Se non fornisci un nome, Lambda genera automaticamente un ID di esecuzione univoco.

I nomi di esecuzione devono essere univoci all'interno dell'account e della regione. Quando richiami una funzione con un nome di esecuzione già esistente, il comportamento di Lambda dipende dallo stato dell'esecuzione esistente e dalla corrispondenza del payload.

## Comportamento idempotente

La tabella seguente descrive come Lambda gestisce le richieste di invocazione in base al fatto che tu fornisca un nome di esecuzione, lo stato di esecuzione esistente e se il payload corrisponde:

| Scenario | Nome fornito? | Stato di esecuzione esistente | Payload identico? | Comportamento                                                          |
|----------|---------------|-------------------------------|-------------------|------------------------------------------------------------------------|
| 1        | No            | N/A                           | N/A               | Nuova esecuzione e iniziata: Lambda genera un ID di esecuzione univoco |

| Scenario | Nome fornito? | Stato di esecuzione esistente                   | Payload identico? | Comportamento                                                                                                                                                                             |
|----------|---------------|-------------------------------------------------|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|          |               |                                                 |                   | e avvia una nuova esecuzione                                                                                                                                                              |
| 2        | Sì            | Non è mai esistito o la conservazione è scaduta | N/A               | Nuova esecuzione e iniziata: Lambda avvia una nuova esecuzione con il nome fornito                                                                                                        |
| 3        | Sì            | In esecuzione                                   | Sì                | Avvio idempotente: Lambda restituisce le informazioni di esecuzione esistenti senza avviarne un duplicato. Per le chiamate sincrone, funge da ricollegamento all'esecuzione in esecuzione |
| 4        | Sì            | In esecuzione                                   | No                | Errore: Lambda restituisce un <code>DurableExecutionAlreadyStartedException</code> errore perché un'esecuzione con questo nome è già in esecuzione con un payload diverso                 |

| Scenario | Nome fornito? | Stato di esecuzione esistente                         | Payload identico? | Comportamento                                                                                                                                                                |
|----------|---------------|-------------------------------------------------------|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 5        | Sì            | Chiuso (riuscito, non riuscito, interrotto o scaduto) | Sì                | Avvio idempotente: Lambda restituisce le informazioni di esecuzione esistenti senza avviare una nuova esecuzione e. Viene restituito il risultato dell'esecuzione chiusa     |
| 6        | Sì            | Chiuso (riuscito, non riuscito, interrotto o scaduto) | No                | Errore: Lambda restituisce un <code>DurableExecutionAlreadyStartedException</code> errore perché un'esecuzione con questo nome è già stata completata con un payload diverso |

## Nota

Gli scenari 3 e 5 dimostrano un comportamento idempotente in cui Lambda gestisce in modo sicuro le richieste di invocazione duplicate restituendo le informazioni di esecuzione esistenti invece di creare duplicati.

## Idempotenza graduale

Per impostazione predefinita, i passaggi hanno una semantica di esecuzione almeno una volta. Quando la funzione viene riprodotta dopo un'attesa, un callback o un errore, l'SDK confronta ogni passaggio con il registro dei checkpoint. Per i passaggi già completati, l'SDK restituisce il risultato del checkpoint senza rieseguire la logica dei passaggi. Tuttavia, se un passaggio fallisce o la funzione

viene interrotta prima del completamento del passaggio, il passaggio potrebbe essere eseguito più volte.

La logica aziendale all'interno delle fasi deve essere idempotente per gestire potenziali tentativi. Usa le chiavi di idempotenza per garantire che operazioni come i pagamenti o le scritture sul database vengano eseguite una sola volta, anche se il passaggio viene ripetuto. Consulta [Idempotency and Retries](#) nella AWS Durable Execution SDK Developer Guide per dettagli su come codificare l'idempotenza.

## Test delle funzioni durevoli di Lambda

Utilizza gli SDK di test Durable Execution per eseguire e ispezionare le esecuzioni sia localmente che nel cloud. Per la creazione di test, asserzioni, cloud runner, integrazione con la CLI SAM ed esempi completi, consulta [Testing](#) in the AWS Durable Execution SDK Developer Guide.

### Autorizzazioni IAM per i test sul cloud

Quando utilizzi il cloud runner o `sam remote invoke` per testare una funzione durevole distribuita, il principale chiamante necessita dell'autorizzazione per richiamare la funzione e leggerne la cronologia di esecuzione. Allega le seguenti autorizzazioni al chiamante di prova:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction",
        "lambda:GetDurableExecution",
        "lambda:GetDurableExecutionHistory"
      ],
      "Resource": [
        "arn:aws:lambda:region:account-id:function:function-name",
        "arn:aws:lambda:region:account-id:function:function-name:*"
      ]
    }
  ]
}
```

Sostituisci *region* e *function-name* con i tuoi valori. *account-id*



## Errori di debug

Quando i test falliscono, controlla il risultato dell'esecuzione per capire cosa è andato storto. Controlla lo stato di esecuzione per vedere se la funzione è riuscita, non riuscita o è scaduta. Leggi i messaggi di errore per comprendere la causa dell'errore.

Esamina i risultati delle singole operazioni per scoprire dove il comportamento si discosta dalle aspettative. Controlla i risultati delle fasi per vedere quali valori sono stati prodotti. Verifica l'ordine delle operazioni per confermare le operazioni eseguite nella sequenza prevista. Conta le operazioni per garantire che sia stato creato il giusto numero di passaggi, attese e callback.

I problemi più comuni includono il codice non deterministico che produce risultati diversi durante la riproduzione, lo stato condiviso tramite variabili globali che si interrompe durante la riproduzione e le operazioni mancanti a causa di errori di logica condizionale. Usa i debugger e la registrazione standard per esaminare il codice delle funzioni e tenere traccia del flusso di esecuzione.

Per i test sul cloud, controlla la cronologia di esecuzione in CloudWatch Logs per visualizzare i registri operativi dettagliati. Utilizza il tracciamento per tracciare il flusso di esecuzione tra i servizi e identificare i punti deboli.

## Monitoraggio di funzioni durature

Puoi monitorare le tue funzioni durevoli utilizzando CloudWatch metriche, CloudWatch registri e tracciamento. Poiché le funzioni durevoli possono essere eseguite per lunghi periodi e comprendere più invocazioni di funzioni, il loro monitoraggio richiede la comprensione dei loro modelli di esecuzione unici, inclusi i checkpoint, le transizioni di stato e il comportamento di ripetizione.

### CloudWatch metriche

Lambda pubblica automaticamente le metriche senza costi aggiuntivi CloudWatch. Le funzioni Durable forniscono metriche aggiuntive oltre alle metriche Lambda standard per aiutarti a monitorare i flussi di lavoro di lunga durata, la gestione dello stato e l'utilizzo delle risorse.

#### Metriche di esecuzione durevoli

Lambda emette le seguenti metriche per esecuzioni durature:

| Metrica                                        | Description                                                                                             |
|------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| ApproximateRunningDurableExecutions            | Numero di esecuzioni durevoli nello stato RUNNING                                                       |
| ApproximateRunningDurableExecutionsUtilization | Percentuale della quota massima di esecuzioni durevoli in esecuzione del tuo account attualmente in uso |
| DurableExecutionDuration                       | È trascorso il tempo in millisecondi in cui un'esecuzione duratura è rimasta nello stato RUNNING        |
| DurableExecutionStarted                        | Numero di esecuzioni durature iniziate                                                                  |
| DurableExecutionStopped                        | Numero di esecuzioni durevoli interrotte utilizzando l'API StopDurableExecution                         |
| DurableExecutionSucceeded                      | Numero di esecuzioni durevoli completate con successo                                                   |
| DurableExecutionFailed                         | Numero di esecuzioni durevoli completate con un errore                                                  |
| DurableExecutionTimedOut                       | Numero di esecuzioni durevoli che hanno superato il timeout di esecuzione configurato                   |
| DurableExecutionOperations                     | Numero cumulativo di operazioni eseguite nell'ambito di un'esecuzione duratura (massimo: 3.000)         |
| DurableExecutionStorageWrittenBytes            | Quantità cumulativa di dati in byte mantenuta da un'esecuzione duratura (max: 100 MB)                   |

## CloudWatch metriche

Lambda emette metriche standard di invocazione, prestazioni e concorrenza per funzioni durature. Poiché un'esecuzione duratura può includere più invocazioni di funzioni man mano che procede attraverso checkpoint e replay, queste metriche si comportano diversamente rispetto alle funzioni standard:

- **Richiami:** conta ogni chiamata di funzione, inclusi i replay. Una singola esecuzione duratura può generare più punti dati di chiamata.

- **Durata:** misura ogni chiamata di funzione separatamente. Utilizza `DurableExecutionDuration` il tempo totale impiegato da una singola esecuzione duratura.
- **Errori:** tiene traccia degli errori di richiamo delle funzioni. Da utilizzare per errori a livello `DurableExecutionFailed` di esecuzione.

Per un elenco completo delle metriche Lambda standard, vedi [Tipi di metriche per le funzioni Lambda](#).

## Creazione di allarmi CloudWatch

Crea CloudWatch allarmi per notificarti quando le metriche superano le soglie. Gli allarmi comuni includono:

- `ApproximateRunningDurableExecutionsUtilizations` supera l'80% della tua quota
- `DurableExecutionFailed` aumenta al di sopra di una soglia
- `DurableExecutionTimedOut` indica che le esecuzioni stanno scadendo
- `DurableExecutionStorageWrittenBytes` si avvicina ai limiti di archiviazione

Per ulteriori informazioni, vedere [Utilizzo degli CloudWatch allarmi](#).

## EventBridge eventi

Lambda pubblica eventi durevoli di modifica dello stato di esecuzione in EventBridge. Puoi utilizzare questi eventi per attivare flussi di lavoro, inviare notifiche o tenere traccia delle modifiche al ciclo di vita dell'esecuzione nelle tue funzioni durevoli.

### Eventi di modifica dello stato di esecuzione duraturi

Lambda emette un evento EventBridge ogni volta che un'esecuzione durevole cambia stato. Questi eventi hanno le seguenti caratteristiche:

- **Origine:** `aws.lambda`
- **Tipo di dettaglio:** `Durable Execution Status Change`

Gli eventi di modifica dello stato vengono pubblicati per i seguenti stati di esecuzione:

- **RUNNING-** Esecuzione iniziata

- SUCCEEDED- Esecuzione completata con successo
- STOPPED- Esecuzione interrotta utilizzando l' StopDurableExecution API
- FAILED- Esecuzione non riuscita con un errore
- TIMED\_OUT- L'esecuzione ha superato il timeout configurato

L'esempio seguente mostra un evento duraturo di modifica dello stato di esecuzione:

```
{
  "version": "0",
  "id": "d019b03c-a8a3-9d58-85de-241e96206538",
  "detail-type": "Durable Execution Status Change",
  "source": "aws.lambda",
  "account": "123456789012",
  "time": "2025-11-20T13:08:22Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "durableExecutionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:$LATEST/durable-execution/090c4189-b18b-4296-9d0c-cfd01dc3a122/9f7d84c9-ea3d-3ffc-b3e5-5ec51c34ffc9",
    "durableExecutionName": "order-123",
    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:2",
    "status": "RUNNING",
    "startTimestamp": "2025-11-20T13:08:22.345Z"
  }
}
```

Per gli stati terminali (SUCCEEDED, STOPPED, FAILED, TIMED\_OUT), l'evento include un `endTimeStamp` campo che indica quando l'esecuzione è stata completata.

## Creazione di EventBridge regole

Crea EventBridge regole per indirizzare eventi duraturi di modifica dello stato di esecuzione a destinazioni come Amazon Simple Notification Service, Amazon Simple Queue Service o altre funzioni Lambda.

L'esempio seguente crea una regola che corrisponde a tutte le modifiche durature dello stato di esecuzione:

```
{
  "source": ["aws.lambda"],
  "detail-type": ["Durable Execution Status Change"]
}
```

L'esempio seguente crea una regola che corrisponde solo alle esecuzioni non riuscite:

```
{
  "source": ["aws.lambda"],
  "detail-type": ["Durable Execution Status Change"],
  "detail": {
    "status": ["FAILED"]
  }
}
```

L'esempio seguente crea una regola che corrisponde alle modifiche di stato per una funzione specifica:

```
{
  "source": ["aws.lambda"],
  "detail-type": ["Durable Execution Status Change"],
  "detail": {
    "functionArn": [{
      "prefix": "arn:aws:lambda:us-east-1:123456789012:function:my-function"
    }]
  }
}
```

Per ulteriori informazioni sulla creazione di EventBridge regole, consulta i EventBridge tutorial di [Amazon](#) nella Guida per l' EventBridge utente.

## Utilizzo di code di lettere morte con eventi EventBridge

EventBridge Gli eventi di modifica dello stato forniscono una notifica in tempo reale quando un'esecuzione durevole entra in uno stato FAILED, STOPPED o TIMED\_OUT, ma non includono

il payload dell'evento originale che ha attivato l'esecuzione. Per acquisire l'evento scatenante per un'analisi o una rielaborazione successive, configura una coda di lettere morte (DLQ) sulla tua funzione durevole.

Quando un'esecuzione durevole fallisce dopo un'invocazione asincrona, Lambda invia l'evento di attivazione originale al DLQ configurato. Usa EventBridge eventi e DLQ insieme per una strategia completa di gestione dei guasti:

- EventBridge eventi: attiva avvisi immediati, aggiorna le dashboard o avvia flussi di lavoro di correzione quando le esecuzioni falliscono.
- Dead-letter code: preserva il payload originale dell'evento in modo da poter verificare la causa dell'errore, eseguire il debug del problema e, facoltativamente, rielaborare l'evento dopo aver risolto il problema sottostante.

Per informazioni sulla configurazione di un DLQ sulla tua funzione, vedi queues. [Dead-letter](#)

## AWS X-Ray tracciato

Puoi abilitare il X-Ray tracciamento sulle tue funzioni durevoli. Lambda passa l'intestazione X-Ray trace all'esecuzione durevole, consentendoti di tracciare le richieste nel tuo flusso di lavoro.

Per abilitare il X-Ray tracciamento utilizzando la console Lambda, scegli la tua funzione, quindi scegli Strumenti di configurazione, monitoraggio e operazioni e attiva Active tracing in. X-Ray

Per abilitare il X-Ray tracciamento utilizzando: AWS CLI

```
aws lambda update-function-configuration \  
  --function-name my-durable-function \  
  --tracing-config Mode=Active
```

Per abilitare il AWS X-Ray tracciamento utilizzando: AWS SAM

```
Resources:  
  MyDurableFunction:  
    Type: AWS::Serverless::Function  
    Properties:  
      Tracing: Active
```

```
DurableConfig:  
  ExecutionTimeout: 3600
```

Per ulteriori informazioni X-Ray, consulta la Guida [per gli](#) AWS X-Ray sviluppatori.

## Le migliori pratiche per le funzioni durevoli Lambda

Le funzioni Durable utilizzano un modello di esecuzione basato sulla ripetizione che richiede pratiche di programmazione diverse rispetto alle tradizionali funzioni Lambda. Consulta le [best practice](#) nella AWS Durable Execution SDK Developer Guide per indicazioni su come scrivere e testare codice durevole per flussi di lavoro.

I seguenti consigli sono le migliori pratiche per l'implementazione, l'invocazione e il monitoraggio delle funzioni durevoli Lambda.

### Versioni e alias delle funzioni

Richiama funzioni con numeri di versione o alias per aggiungere le esecuzioni a versioni di codice specifiche. Assicurati che le nuove versioni del codice siano in grado di gestire lo stato delle versioni precedenti. Non rinominare i passaggi o modificarne il comportamento in modo da interrompere la riproduzione. Se rinomini un passaggio o ne modifichi il comportamento mentre sono in corso delle esecuzioni, tali esecuzioni possono non riprendere o produrre risultati errati. Ciò accade perché il runtime non è più in grado di far corrispondere lo stato salvato alla definizione della fase prevista.

### Monitoraggio

Abilita la registrazione strutturata con ID di esecuzione e nomi dei passaggi. Imposta CloudWatch allarmi per i tassi di errore e la durata dell'esecuzione. Usa il tracciamento per identificare i colli di bottiglia. Per una guida dettagliata, consulta [Monitoraggio e debug](#).

### Gestione degli errori

Oltre a configurare le strategie di ripetizione dei tentativi per gli errori transitori, configura una coda di lettere morte (DLQ) sulla tua funzione durevole per acquisire eventi da esecuzioni permanentemente fallite. Quando un'esecuzione duratura raggiunge uno stato terminale (FAILED, STOPPED o TIMED\_OUT) dopo una chiamata asincrona, Lambda invia l'evento di attivazione originale al DLQ. In questo modo, è possibile ispezionare, eseguire il debug e, facoltativamente, rielaborare gli eventi non riusciti senza perderli.

Per configurare un DLQ, imposta la `DeadLetterConfig` proprietà della tua funzione su una coda Amazon SQS o su un argomento Amazon SNS ARN. Per ulteriori informazioni, consulta [code](#). [Dead-letter](#)

Segui queste best practice per la gestione degli errori con funzioni durevoli:

- Configura un DLQ per le chiamate asincrone: allega sempre una coda di lettere morte quando richiami funzioni durevoli in modo asincrono. A differenza delle funzioni Lambda standard, il servizio non ritenta automaticamente le esecuzioni durevoli in caso di errore. Un DLQ correttamente configurato acquisisce gli eventi che hanno portato a esecuzioni permanentemente fallite. Assicuratevi che il DLQ disponga delle autorizzazioni corrette e di una capacità sufficiente per ricevere messaggi.
- Utilizza le strategie di ripetizione in fasi: configura strategie di ripetizione esplicite con un backoff appropriato per gli errori transitori. Per indicazioni sulla configurazione dei tentativi, consulta [Ritentativi per funzioni durevoli](#).
- Combina i DLQ con EventBridge le notifiche: utilizza EventBridge le regole per segnalare le modifiche allo stato FAILED, STOPPED e TIMED\_OUT per una visibilità in tempo reale e utilizza un DLQ per preservare il payload dell'evento originale per successive analisi o rielaborazioni.
- Monitora la profondità del DLQ: crea un CloudWatch allarme sulla `ApproximateNumberOfMessagesVisible` metrica per il tuo DLQ per rilevare quando i guasti si stanno accumulando.

## Risorse correlate

- [AWS Guida per gli sviluppatori di Durable Execution SDK](#)
- [Monitoraggio delle funzioni durature di Lambda](#)
- [Tentativi per le funzioni durevoli di Lambda](#)
- [Test delle funzioni durevoli di Lambda](#)

## Miglioramento delle prestazioni di avvio con Lambda SnapStart

Lambda SnapStart è in grado di fornire prestazioni di avvio inferiori al secondo, in genere senza modifiche al codice funzione. SnapStart semplifica la creazione di applicazioni altamente reattive e scalabili senza fornire risorse o implementare complesse ottimizzazioni delle prestazioni.



Il fattore che contribuisce maggiormente alla latenza di avvio (spesso definita tempo di avvio a freddo) è il tempo impiegato da Lambda per inizializzare la funzione, che include il caricamento del codice della funzione, l'avvio del runtime e l'inizializzazione del codice della funzione. Con SnapStart, Lambda inizializza la funzione quando si pubblica una versione della funzione. Lambda acquisisce un'istantanea [Firecracker MicroVM della memoria e dello stato del disco dell'ambiente di esecuzione](#) [inizializzato, crittografa l'istantanea e la memorizza nella cache in modo intelligente per ottimizzare la latenza di recupero](#).

Per garantire la resilienza, Lambda conserva diverse copie di ogni snapshot. Lambda aggiorna automaticamente gli snapshot e le relative copie con gli aggiornamenti di runtime e sicurezza più recenti. Quando richiami la versione della funzione per la prima volta e man mano che le chiamate aumentano, Lambda riprende i nuovi ambienti di esecuzione dallo snapshot memorizzato nella cache invece di inizializzarli da zero, migliorando così la latenza di avvio.

#### Important

Se le applicazioni dipendono dall'univocità dello stato, è necessario valutare il codice della funzione e verificare che sia resiliente alle operazioni di snapshot. Per ulteriori informazioni, consulta [Gestire l'unicità con Lambda SnapStart](#).

Se [the section called “Risoluzione dei problemi”](#) riscontri problemi, consulta.

#### Argomenti

- [Quando usare SnapStart](#)
- [Funzionalità e limitazioni supportate](#)
- [Regioni supportate](#)
- [Considerazioni sulla compatibilità](#)
- [SnapStart prezzi](#)
- [Attivazione e gestione di Lambda SnapStart](#)
- [Gestire l'unicità con Lambda SnapStart](#)
- [Implementare il codice prima o dopo gli snapshot della funzione Lambda](#)
- [Monitoraggio per Lambda SnapStart](#)
- [Modello di sicurezza per Lambda SnapStart](#)
- [Massimizza le prestazioni Lambda SnapStart](#)

- [Risoluzione degli SnapStart errori per le funzioni Lambda](#)

## Quando usare SnapStart

Lambda SnapStart è progettato per risolvere la variabilità della latenza introdotta da un codice di inizializzazione una tantum, come il caricamento delle dipendenze o dei framework dei moduli. Il completamento di queste operazioni a volte può richiedere diversi secondi durante l'invocazione iniziale. SnapStart Usalo per ridurre questa latenza da alcuni secondi a pochi secondi, in scenari ottimali. SnapStart funziona meglio se utilizzato con invocazioni di funzioni su larga scala. Le funzioni richiamate di rado potrebbero non presentare gli stessi miglioramenti delle prestazioni.

SnapStart è particolarmente utile per due tipi principali di applicazioni:

- Latency-sensitive API e flussi utente: le funzioni che fanno parte degli endpoint API critici o dei flussi rivolti agli utenti possono trarre vantaggio dalla latenza ridotta e dai tempi SnapStart di risposta migliorati.
- Latency-sensitive flussi di lavoro di elaborazione dati: Time-bound i flussi di lavoro di elaborazione dati che utilizzano funzioni Lambda possono ottenere un throughput migliore riducendo la latenza di inizializzazione delle funzioni anomale.

La [simultaneità fornita](#) mantiene le funzioni inizializzate e pronte a rispondere entro i 100 millisecondi. Usa la concorrenza predisposta se la tua applicazione ha rigidi requisiti di latenza di avvio a freddo che non possono essere risolti in modo adeguato. SnapStart

## Funzionalità e limitazioni supportate

SnapStart supporta i seguenti runtime gestiti [da Lambda](#) e le corrispondenti immagini di base AWS Lambda nei formati di distribuzione delle immagini ZIP e container:

- Java 11 e versioni successive
- Python 3.12 e versioni successive
- .NET 8 e versioni successive. Se stai utilizzando il framework [Lambda Annotations per .NET](#), esegui l'aggiornamento alla [Amazon.Lambda.Annotations](#) versione 1.6.0 o successiva per garantire la compatibilità con. SnapStart

Per le immagini dei contenitori create su immagini di base AWS Lambda o immagini di base personalizzate, consulta [Implementazione di SnapStart hook per le immagini dei contenitori](#) e convalida il codice della funzione per gestire i requisiti di unicità. SnapStart ???

Altri runtime gestiti (come `nodejs24.x` e `eruby4.0`) e [OS-only tempi di esecuzione](#) non sono supportati. SnapStart non supporta la concorrenza con [provisioning](#), [Amazon Elastic File System \(Amazon EFS\)](#), i [file](#) Amazon S3 o lo storage temporaneo superiore a 512 MB.

#### Note

È possibile utilizzarlo SnapStart solo sulle versioni di funzioni [pubblicate](#) e sugli [alias](#) che rimandano a versioni. Non è possibile utilizzarlo SnapStart sulla versione non pubblicata di una funzione (\$LATEST).

## Regioni supportate

Lambda SnapStart è disponibile in tutte le regioni [commerciali](#) tranne Asia Pacifico (Nuova Zelanda) e Asia Pacifico (Taipei).

## Considerazioni sulla compatibilità

Con SnapStart, Lambda utilizza una singola istantanea come stato iniziale per più ambienti di esecuzione. Se la tua funzione utilizza uno dei seguenti elementi durante la fase di [inizializzazione](#), potrebbe essere necessario apportare alcune modifiche prima di utilizzarla: SnapStart

### Unicità

Se il codice di inizializzazione genera contenuti unici inclusi nello snapshot, il contenuto potrebbe non essere più unico quando viene riutilizzato in più ambienti di esecuzione. Per mantenere l'unicità durante l'utilizzo SnapStart, è necessario generare contenuti univoci dopo l'inizializzazione. Ciò include ID univoci, segreti univoci ed entropia utilizzata per generare pseudocasualità. Per informazioni su come ripristinare l'unicità, consulta [Gestire l'unicità con Lambda SnapStart](#).

### Connessioni di rete

Lo stato delle connessioni che la funzione stabilisce durante la fase di inizializzazione non è garantito quando Lambda riprende la funzione da uno snapshot. Convalida lo stato delle connessioni di rete e ristabiliscile se necessario. Nella maggior parte dei casi, le connessioni

di rete stabilite da un AWS SDK vengono ripristinate automaticamente. Per altre connessioni, consulta le [best practice](#).

## Dati temporanei

Durante la fase di inizializzazione, alcune funzioni scaricano o inizializzano dati effimeri, come credenziali temporanee o timestamp memorizzati nella cache. Aggiorna i dati temporanei nel gestore delle funzioni prima di utilizzarli, anche quando non li usi. SnapStart

## SnapStart prezzi

### Note

Per i runtime gestiti da Java, non sono previsti costi aggiuntivi per SnapStart. L'addebito viene effettuato in base al numero di richieste per le funzioni, al tempo impiegato dal codice per l'esecuzione e alla memoria configurata per la funzione.

Il costo di utilizzo SnapStart include quanto segue:

- Memorizzazione nella cache: per ogni versione della funzione pubblicata con SnapStart abilitata, paghi il costo della memorizzazione nella cache e della manutenzione dell'istantanea. Il prezzo dipende dalla quantità di [memoria](#) allocata alla funzione. L'addebito sarà per un minimo di 3 ore.

L'addebito continua a essere addebitato finché la funzione rimane attiva. [???](#) Utilizza l'azione [ListVersionsByFunction](#) API per identificare le versioni della funzione, quindi utilizzala [DeleteFunction](#) per eliminare le versioni non utilizzate. Per eliminare automaticamente le versioni delle funzioni non utilizzate, consulta il modello [Pulizia della versione Lambda](#) su Serverless Land.

- Ripristino: ogni volta che un'istanza di funzione viene ripristinata da uno snapshot, si paga una tariffa di ripristino. Il prezzo dipende dalla quantità di memoria allocata alla funzione.

Come per tutte le funzioni Lambda, i costi di durata si applicano al codice eseguito nell'handler delle funzioni. Per SnapStart le funzioni, i costi di durata si applicano anche al codice di inizializzazione dichiarato all'esterno del gestore, al tempo impiegato dal runtime per il caricamento e a qualsiasi codice eseguito in un [runtime hook](#). La durata viene calcolata dal momento in cui il codice inizia a funzionare fino a quando ritorna o termina in altro modo, arrotondato al valore di 1 ms più vicino. Lambda conserva copie memorizzate nella cache dello snapshot per garantire la resilienza e applica automaticamente gli aggiornamenti software, come gli upgrade di runtime e le patch di sicurezza.

Gli addebiti si applicano ogni volta che Lambda esegue nuovamente il codice di inizializzazione per applicare gli aggiornamenti software.

Per ulteriori informazioni sui costi di utilizzo SnapStart, consulta [AWS Lambda Prezzi](#).

# Attivazione e gestione di Lambda SnapStart

Per utilizzarlo SnapStart, attiva SnapStart una funzione Lambda nuova o esistente. Quindi, pubblica e richiama una versione della funzione.

## Argomenti

- [Attivazione SnapStart \(console\)](#)
- [Attivazione SnapStart \(AWS CLI\)](#)
- [Attivazione SnapStart \(API\)](#)
- [Lambda SnapStart e stati delle funzioni](#)
- [Aggiornamento di uno snapshot](#)
- [Utilizzo con SnapStart AWS SDK](#)
- [Utilizzo con SnapStart CloudFormation, AWS SAM e AWS CDK](#)
- [Eliminazione di snapshot](#)

## Attivazione SnapStart (console)

### Attivare SnapStart una funzione

1. Aprire la pagina [Functions](#) (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Scegli Configuration (Configurazione), quindi scegli Configurazione generale.
4. Nel pannello General configuration (Configurazione generale), scegli Edit (Modifica).
5. Nella pagina Modifica impostazioni di base SnapStart, scegli Versioni pubblicate.
6. Selezionare Salva.
7. [Pubblica una versione della funzione](#). Lambda inizializza il codice, crea uno snapshot dell'ambiente di esecuzione inizializzato e quindi memorizza nella cache lo snapshot per l'accesso a bassa latenza.
8. [Invoca la versione della funzione](#).

## Attivazione SnapStart (AWS CLI)

SnapStart Attivare una funzione esistente

1. Aggiorna la configurazione della funzione eseguendo il comando [update-function-configuration](#) con l'opzione `--snap-start`.

```
aws lambda update-function-configuration \  
  --function-name my-function \  
  --snap-start ApplyOn=PublishedVersions
```

2. Pubblica una versione della funzione con il comando [publish-version](#).

```
aws lambda publish-version \  
  --function-name my-function
```

3. Verificate che SnapStart sia attivata per la versione della funzione eseguendo il [comando](#) `get-function-configuration` e specificando il numero di versione. Il seguente esempio specifica la versione 1.

```
aws lambda get-function-configuration \  
  --function-name my-function:1
```

Se la risposta indica che [OptimizationStatus](#) è On e [State èActive](#), viene attivata ed SnapStart è disponibile un'istanza per la versione della funzione specificata.

```
"SnapStart": {  
  "ApplyOn": "PublishedVersions",  
  "OptimizationStatus": "On"  
},  
"State": "Active",
```

4. Richiama la versione della funzione eseguendo il comando [invoke](#) e specificando la versione. L'esempio seguente richiama la versione 1.

```
aws lambda invoke \  
  --cli-binary-format raw-in-base64-out \  
  --function-name my-function:1 \  
  --payload '{ "name": "Bob" }' \  
  response.json
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

Da attivare SnapStart quando si crea una nuova funzione

1. Crea una funzione eseguendo il comando [create-function](#) con l'opzione `--snap-start`. Per `--role`, specificare il nome della risorsa Amazon (ARN) del tuo [ruolo di esecuzione](#).

 Note

Se utilizzi una OCI-based funzione che utilizza immagini di base diverse dalle immagini di AWS base di Lambda per Java (versione 11+), Python (versione 3.12+) e .NET (versione 8+), segui le istruzioni su [Implementazione di SnapStart hook per le immagini dei contenitori](#) per rendere l'immagine compatibile con. SnapStart SnapStart

```
aws lambda create-function \  
  --function-name my-function \  
  --runtime "java25" \  
  --zip-file fileb://my-function.zip \  
  --handler my-function.handler \  
  --role arn:aws:iam::111122223333:role/lambda-ex \  
  --snap-start ApplyOn=PublishedVersions
```

2. Crea una versione con il comando [publish-version](#).

```
aws lambda publish-version \  
  --function-name my-function
```

3. Verifica che SnapStart sia attivata per la versione della funzione eseguendo il comando [get-function-configuration e specificando il numero](#) di versione. Il seguente esempio specifica la versione 1.

```
aws lambda get-function-configuration \  
  --function-name my-function:1
```



Se la risposta indica che [OptimizationStatus](#) è 0n e [State èActive](#), viene attivata ed SnapStart è disponibile un'istantanea per la versione della funzione specificata.

```
"SnapStart": {  
  "ApplyOn": "PublishedVersions",  
  "OptimizationStatus": "0n"  
},  
"State": "Active",
```

4. Richiama la versione della funzione eseguendo il comando [invoke](#) e specificando la versione. L'esempio seguente richiama la versione 1.

```
aws lambda invoke \  
  --cli-binary-format raw-in-base64-out \  
  --function-name my-function:1 \  
  --payload '{ "name": "Bob" }' \  
  response.json
```

L'cli-binary-format opzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina [AWS CLI supported global command line options](#) nella Guida per l'utente di AWS Command Line Interface versione 2.

## Attivazione SnapStart (API)

### Attivare SnapStart

1. Esegui una delle seguenti operazioni:
  - Crea una nuova funzione con SnapStart Activated utilizzando l'azione [CreateFunction](#) API con il [SnapStart](#) parametro.
  - SnapStart Attiva una funzione esistente utilizzando l'[UpdateFunctionConfiguration](#) azione con il [SnapStart](#) parametro.
2. Pubblica una versione della funzione con l'[PublishVersion](#) azione. Lambda inizializza il codice, crea uno snapshot dell'ambiente di esecuzione inizializzato e quindi memorizza nella cache lo snapshot per l'accesso a bassa latenza.
3. Verificate che SnapStart sia attivata per la versione della funzione utilizzando l'[GetFunctionConfiguration](#) azione. Specificate un numero di versione per confermare che

SnapStart è attivata per quella versione. Se la risposta indica che [OptimizationStatus](#) è `On` e [State](#) è `Active`, SnapStart viene attivata ed è disponibile un'istanza per la versione della funzione specificata.

```
"SnapStart": {  
  "ApplyOn": "PublishedVersions",  
  "OptimizationStatus": "On"  
},  
"State": "Active",
```

4. Richiama la versione della funzione con l'operazione [Invoke](#).

## Lambda SnapStart e stati delle funzioni

Quando si utilizza SnapStart, possono verificarsi i seguenti stati di funzione.

### Pending (In attesa)

Lambda sta inizializzando il codice e acquisendo uno snapshot dell'ambiente di esecuzione inizializzato. Qualsiasi chiamata o altra azione API che opera sulla versione della funzione ha esito negativo.

### Attivo

La creazione dello snapshot è completa ed è possibile richiamare la funzione. Per utilizzarla SnapStart, è necessario richiamare la versione della funzione pubblicata, non quella non pubblicata (\$LATEST).

### Inattivo

Lo `Inactive` stato può verificarsi quando Lambda rigenera periodicamente le istantanee delle funzioni per applicare gli aggiornamenti software. In questo caso, se la funzione non riesce a inicializzarsi, può entrare in uno stato `Inactive`.

Per le funzioni che utilizzano un runtime Java, Lambda elimina gli snapshot dopo 14 giorni senza alcuna invocazione. Se si richiama la versione della funzione dopo 14 giorni, Lambda restituisce una risposta `SnapStartNotReadyException` e inizia a inizializzare un nuovo snapshot. Attendi che la versione della funzione raggiunga lo stato `Active`, quindi richiamala di nuovo.

### Non riuscito

Lambda ha riscontrato un errore durante l'esecuzione del codice di inizializzazione o la creazione dello snapshot.

## Aggiornamento di uno snapshot

Lambda crea uno snapshot per ogni versione della funzione pubblicata. Per aggiornare uno snapshot, pubblica una nuova versione della funzione.

## Utilizzo con SnapStart AWS SDK

Per effettuare chiamate AWS SDK dalla tua funzione, Lambda genera un set temporaneo di credenziali assumendo il ruolo di esecuzione della tua funzione. Queste credenziali sono disponibili come variabili d'ambiente durante l'invocazione della funzione. Non è necessario fornire le credenziali per l'SDK direttamente nel codice. Per impostazione predefinita, la catena di fornitori di credenziali controlla in sequenza ogni punto in cui è possibile impostare le credenziali e seleziona il primo disponibile, in genere le variabili d'ambiente (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` e `AWS_SESSION_TOKEN`).

### Note

Quando SnapStart è attivato, il runtime Lambda utilizza automaticamente le credenziali del contenitore (`AWS_CONTAINER_CREDENTIALS_FULL_URI` e `AWS_CONTAINER_AUTHORIZATION_TOKEN`) invece delle variabili di ambiente della chiave di accesso. Ciò impedisce la scadenza delle credenziali prima che la funzione venga ripristinata.

## Utilizzo con SnapStart CloudFormation, AWS SAM e AWS CDK

- AWS CloudFormation: dichiara l'[SnapStart](#) entità nel tuo modello.
- AWS Serverless Application Model (AWS SAM): Dichiara la [SnapStart](#) proprietà nel tuo modello.
- Kit di sviluppo per il cloud AWS (AWS CDK): Usa il [SnapStartProperty](#) tipo.

## Eliminazione di snapshot

Lambda elimina gli snapshot quando:

- Si elimina la funzione o la versione della funzione.
- Solo runtime Java: non si richiama la versione della funzione per 14 giorni. Dopo 14 giorni senza una chiamata, la versione della funzione passa allo stato [Inactive](#) (Inattivo). Se si richiama la versione della funzione dopo 14 giorni, Lambda restituisce una risposta

`SnapStartNotReadyException` e inizia a inizializzare un nuovo snapshot. Attendi che la versione della funzione raggiunga lo stato [Active](#) (Attivo), quindi richiamala di nuovo.

Lambda rimuove tutte le risorse associate agli snapshot eliminati in conformità con il Regolamento generale sulla protezione dei dati (GDPR).

## Gestire l'unicità con Lambda SnapStart

Quando le invocazioni aumentano su una SnapStart funzione, Lambda utilizza una singola istantanea inizializzata per riprendere più ambienti di esecuzione. Se il codice di inizializzazione genera contenuti unici inclusi nello snapshot, il contenuto potrebbe non essere più unico quando viene riutilizzato in più ambienti di esecuzione. Per mantenere l'unicità durante l'utilizzo SnapStart, è necessario generare contenuti univoci dopo l'inizializzazione. Ciò include ID univoci, segreti univoci ed entropia utilizzata per generare pseudocasualità.

Di seguito sono riportate le best practice che consentono di mantenere l'unicità nel codice. Per le funzioni Java, Lambda fornisce anche uno strumento di [SnapStart scansione open source per aiutare](#) a verificare la presenza di codice che presuppone l'unicità. Se durante la fase di inizializzazione vengono generati dati univoci, è possibile utilizzare un [hook di runtime](#) per ripristinare l'unicità. Con gli hook di runtime, puoi eseguire codice specifico immediatamente prima che Lambda esegua uno snapshot o subito dopo che Lambda riprende una funzione da uno snapshot.

### Evitare lo stato di salvataggio che dipende dall'unicità durante l'inizializzazione

Durante la [fase di inizializzazione](#) della funzione, evita di memorizzare nella cache dati destinati a essere univoci, ad esempio la generazione di un ID univoco per la registrazione o l'impostazione di seed per funzioni casuali. Ti consigliamo invece di generare dati univoci o impostare seed per le funzioni casuali all'interno dell'handler delle funzioni o di utilizzare un [hook di runtime](#).

I seguenti esempi mostrano come generare un UUID nell'handler delle funzioni.

#### Java

Example: generazione di un ID univoco nel gestore delle funzioni

```
import java.util.UUID;

public class Handler implements RequestHandler<String, String> {
    private static UUID uniqueSandboxId = null;
    @Override
    public String handleRequest(String event, Context context) {
        if (uniqueSandboxId == null)
            uniqueSandboxId = UUID.randomUUID();
        System.out.println("Unique Sandbox Id: " + uniqueSandboxId);
        return "Hello, World!";
    }
}
```

## Python

Example: generazione di un ID univoco nel gestore delle funzioni

```
import json
import random
import time

unique_number = None

def lambda_handler(event, context):
    seed = int(time.time() * 1000)
    random.seed(seed)
    global unique_number
    if not unique_number:
        unique_number = random.randint(1, 10000)

    print("Unique number: ", unique_number)

    return "Hello, World!"
```

## .NET

Example: generazione di un ID univoco nel gestore delle funzioni

```
namespace Example;
public class SnapstartExample
{
    private Guid _myExecutionEnvironmentGuid;
    public SnapstartExample()
    {
        // This GUID is set for non-restore use cases, such as testing or if
        // SnapStart is turned off
        _myExecutionEnvironmentGuid = new Guid();
        // Register the method which will run after each restore. You may need to
        // update Amazon.Lambda.Core to see this
        Amazon.Lambda.Core.SnapshotRestore.RegisterAfterRestore(MyAfterRestore);
    }

    private ValueTask MyAfterRestore()
    {
        // After restoring this snapshot to a new execution environment, update the
        // GUID
    }
}
```

```
        _myExecutionEnvironmentGuid = new Guid();
        return ValueTask.CompletedTask;
    }

    public string Handler()
    {
        return $"Hello World! My Execution Environment GUID is
{_myExecutionEnvironmentGuid}";
    }
}
```

## Utilizza generatori di numeri pseudocasuali crittograficamente sicuri (CSPRNG)

Quando SnapStart è abilitato, Lambda scatta un'istantanea dell'ambiente di esecuzione della funzione, inclusa tutta la memoria dell'applicazione e del sistema. Ciò significa che lo stato interno di ogni generatore di numeri casuali (RNG) viene mantenuto esattamente com'era al momento dell'istantanea, insieme a tutti i byte casuali che l'applicazione o le sue dipendenze hanno già generato e archiviati nei buffer di memoria.

Quando Lambda ripristina un ambiente di esecuzione da un'istantanea, risemina il generatore di numeri casuali del kernel da e con una nuova entropia. `/dev/random` `/dev/urandom` Il software che legge numeri casuali direttamente da questi dispositivi mantiene la casualità con. SnapStart

Se l'applicazione dipende dalla casualità, si consiglia di utilizzare generatori di numeri casuali crittograficamente sicuri (CSPRNG). Oltre all'OpenSSL fornito da AL, i runtime gestiti da Lambda che supportano SnapStart (Java versione 11+, Python versione 3.12+ e .NET versione 8+) includono i seguenti CSPRNG integrati:

- Java: `java.security.SecureRandom`
- Python: `random.SystemRandom`
- .NET: `System.Security.Cryptography.RandomNumberGenerator`

### Note

Per il runtime di [Go Lambda](#) non sono necessarie modifiche se le funzioni utilizzano la libreria standard, che è compatibile con snapstart per impostazione predefinita. `crypto/rand`

Quando impacchettate la funzione come immagine contenitore, l'immagine di base determina la compatibilità dell'unicità con: SnapStart

- Un'immagine base Lambda per un runtime gestito (Java versione 11+, Python versione 3.12+ e .NET versione 8+): compatibile con quanto descritto sopra. SnapStart
- L'immagine di base provided.al2023: compatibile se il runtime del linguaggio di programmazione ottiene entropia da OpenSSL (openssl-snap-safe-libs) o dalla SnapStart-compatible build di OpenSSL (/dev/random openssl-snap-safe-libs) /dev/urandom elencata nell'elenco dei pacchetti di Amazon Linux 2023. <https://docs.aws.amazon.com/linux/al2023/release-notes/all-packages.html>  
Se il tuo runtime si basa su altre fonti di entropia, segui i passaggi descritti nella sezione «La tua immagine di base» di seguito.
- Immagine di base personalizzata: controlla le tue librerie crittografiche e utilizza un [runtime hook successivo](#) al ripristino per eliminare qualsiasi stato del generatore di numeri casuali (RNG) memorizzato nella cache o i byte casuali bufferizzati, assicurandoti che la libreria legga nuova entropia dal sistema dopo il ripristino. Ad esempio, le funzioni Rust utilizzate devono richiamare dopo il ripristino. `rand::rngs::ThreadRng::reseed()` Ti consigliamo di creare utilizzando un'immagine base Lambda quando possibile. In particolare, se l'immagine di base include una propria versione di OpenSSL o un fork di OpenSSL (come BoringSSL o LibreSSL), passa all'immagine di base provided.al2023, che include una SnapStart-compatible build di OpenSSL (openssl-snap-safe-libs), oppure usa libcrypto (-LC). [AWSAWS](#)

### Versioni minime della libreria supportate

AWS Le librerie di crittografia mantengono automaticamente la casualità a SnapStart partire dalle versioni minime specificate nella tabella seguente. Se usi queste librerie con le tue funzioni Lambda, assicurati di utilizzare le seguenti versioni minime o versioni successive:

| Libreria               | Versione minima supportata (x86) | Versione minima supportata (ARM) |
|------------------------|----------------------------------|----------------------------------|
| AWS libcrypto (-LC)AWS | 1.16.0                           | 1.30.0                           |
| AWS libcrypto FIPS     | 2.0.13                           | 2.0.13                           |

Se impacchetti le librerie di crittografia precedenti con le funzioni Lambda come dipendenze transitive tramite le seguenti librerie, assicurati di utilizzare le seguenti versioni minime o versioni successive:



| Libreria                                                  | Versione minima supportata (x86) | Versione minima supportata (ARM) |
|-----------------------------------------------------------|----------------------------------|----------------------------------|
| AWS SDK for Java 2.x                                      | 2.23.20                          | 2,26,12                          |
| AWS Common Runtime per Java                               | 0.29.8                           | 0,29,25                          |
| Fornitore di servizi di crittografia Amazon Corretto      | 2.4.1                            | 2.4.1                            |
| FIPS Fornitore di servizi di crittografia Amazon Corretto | 2.4.1                            | 2.4.1                            |

Gli esempi seguenti mostrano come utilizzare i CSPRNG per garantire sequenze numeriche univoche anche quando la funzione viene ripristinata da uno snapshot.

## Java

### Example— java.security. SecureRandom

```
import java.security.SecureRandom;
public class Handler implements RequestHandler<String, String> {
    private static SecureRandom rng = new SecureRandom();
    @Override
    public String handleRequest(String event, Context context) {
        for (int i = 0; i < 10; i++) {
            System.out.println(rng.next());
        }
        return "Hello, World!";
    }
}
```

## Python

### Example— casuale. SystemRandom

```
import json
import random
```

```

secure_rng = random.SystemRandom()

def lambda_handler(event, context):
    random_numbers = [secure_rng.random() for _ in range(10)]

    for number in random_numbers:
        print(number)

    return "Hello, World!"

```

## .NET

### Example– RandomNumberGenerator

```

using Amazon.Lambda.Core;
using System.Security.Cryptography;
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace DotnetSecureRandom;

public class Function
{
    public string FunctionHandler()
    {
        using (RandomNumberGenerator rng = RandomNumberGenerator.Create())
        {
            byte[] randomUnsignedInteger32Bytes = new byte[4];
            for (int i = 0; i < 10; i++)
            {
                rng.GetBytes(randomUnsignedInteger32Bytes);
                int randomInt32 = BitConverter.ToInt32(randomUnsignedInteger32Bytes,
0);

                Console.WriteLine("{0:G}", randomInt32);
            }
            return "Hello World!";
        }
    }
}

```

## SnapStart strumento di scansione (solo Java)

Lambda fornisce uno strumento di scansione che aiuta a verificare la presenza di codice che presuppone l'unicità. Lo strumento SnapStart di scansione è un [SpotBugs](#) plug-in open source che esegue un'analisi statica rispetto a una serie di regole. Lo strumento di scansione consente di identificare potenziali implementazioni di codice che potrebbero infrangere i presupposti sull'unicità. Per le istruzioni di installazione e un elenco dei controlli eseguiti dallo strumento di scansione, consulta il repository <https://github.com/aws/aws-lambda-snapstart-java-rules> aws-lambda-snapstart-java-rules su. GitHub

Per saperne di più sulla gestione dell'unicità con, consulta [Starting up faster with](#) sul Compute Blog. SnapStart [AWS Lambda SnapStart](#) AWS

# Implementare il codice prima o dopo gli snapshot della funzione Lambda

Gli hook di runtime possono essere utilizzati per implementare il codice prima che Lambda crei uno snapshot o dopo che ha ripristinato una funzione da uno snapshot. Gli hook di runtime sono utili per diversi scopi, ad esempio:

- **Pulizia e inizializzazione:** prima di creare uno snapshot, è possibile utilizzare un hook di runtime per eseguire operazioni di pulizia o rilascio delle risorse. Dopo il ripristino di uno snapshot, è possibile utilizzare un runtime hook per reinizializzare tutte le risorse o lo stato che non sono stati acquisiti nello snapshot.
- **Configurazione dinamica:** è possibile utilizzare gli hook di runtime per aggiornare dinamicamente la configurazione o altri metadati prima della creazione di uno snapshot o dopo il suo ripristino. Ciò può essere utile se la funzione deve adattarsi ai cambiamenti nell'ambiente di runtime.
- **Integrazioni esterne:** è possibile utilizzare gli hook di runtime per l'integrazione con servizi o sistemi esterni, ad esempio l'invio di notifiche o l'aggiornamento dello stato esterno, come parte del processo di checkpoint e ripristino.
- **Ottimizzazione delle prestazioni:** è possibile utilizzare gli hook di runtime per ottimizzare la sequenza di avvio della funzione, ad esempio precaricando le dipendenze. Per ulteriori informazioni, consulta [Ottimizzazione prestazioni](#).

Le pagine seguenti spiegano come implementare gli hook di runtime per il runtime preferito.

## Argomenti

- [Hook SnapStart runtime Lambda per Java](#)
- [Hook di runtime SnapStart per Lambda per Python](#)
- [Hook di runtime SnapStart per Lambda per .NET](#)
- [Implementazione di SnapStart hook per le immagini dei contenitori](#)

## Hook SnapStart runtime Lambda per Java

Gli hook di runtime possono essere utilizzati per implementare il codice prima che Lambda crei uno snapshot o dopo che ha ripristinato una funzione da uno snapshot. Gli hook di runtime sono disponibili come parte del progetto open source Coordinated Restore at Checkpoint (CRAc). CRAc è in fase sviluppo per l'[Open Java Development Kit \(OpenJDK\)](#). Per un esempio di come utilizzare CRAc con un'applicazione di riferimento, consultate il repository [CRAc su GitHub](#). Il CRAc utilizza tre elementi principali:

- **Resource**: un'interfaccia con due metodi, `beforeCheckpoint()` e `afterRestore()`. Utilizza questi metodi per implementare il codice che desideri eseguire prima di uno snapshot e dopo un ripristino.
- **Context <R extends Resource>**: per ricevere notifiche relative ai checkpoint e ai ripristini, è necessario registrare una **Resource** con un **Context**.
- **Core**— Il servizio di coordinamento, che fornisce il valore predefinito globale **Context** tramite il metodo statico. `Core.getGlobalContext()`

Per ulteriori informazioni su **Context** e **Resource**, consulta [Pacchetto org.crac](https://aws.amazon.com/blogs/compute/using-context-to-manage-lambda-function-snapshots/) nella documentazione di CRaC.

Completa le seguenti operazioni per implementare gli hook di runtime con il [pacchetto org.crac](https://aws.amazon.com/blogs/compute/using-context-to-manage-lambda-function-snapshots/). Il runtime Lambda contiene un'implementazione del contesto CRaC personalizzata che chiama gli hook di runtime prima del checkpoint e dopo il ripristino.

### Registrazione ed esecuzione di hook di runtime

L'ordine in cui Lambda esegue gli hook di runtime è determinato dall'ordine di registrazione. L'ordine di registrazione segue l'ordine di importazione, definizione o esecuzione nel codice.

- `beforeCheckpoint()`: eseguito nell'ordine di registrazione inverso
- `afterRestore()`: eseguito nell'ordine di registrazione

Assicurati che tutti gli hook registrati siano stati importati e inclusi correttamente nel codice della tua funzione. Se registri gli hook di runtime in un file o modulo separato, dovrai assicurarti che il modulo venga importato, direttamente o come parte di un pacchetto più grande, nel file handler della funzione. Se il file o il modulo non viene importato nell'handler della funzione, Lambda ignorerà gli hook di runtime.

#### Note

Quando Lambda crea uno snapshot, il codice di inizializzazione può essere eseguito per un massimo di 15 minuti. Il limite di tempo è 130 secondi o il [timeout della funzione configurato](#) (massimo 900 secondi), a seconda di quale dei due valori sia più elevato. I tuoi hook di runtime `beforeCheckpoint()` contano ai fini del limite di tempo del codice di inizializzazione. Quando Lambda ripristina uno snapshot, il runtime deve essere caricato e

gli hook di runtime `afterRestore()` devono essere completati entro il limite di timeout (10 secondi). Altrimenti, otterrai un `SnapStartTimeoutException`.

## Fase 1: aggiornamento della configurazione di build

Aggiungi la dipendenza `org.crac` alla configurazione della build. Nell'esempio seguente viene utilizzato Gradle. Per esempi di altri sistemi di compilazione, consulta la [documentazione di Apache Maven](#).

```
dependencies {  
    compile group: 'com.amazonaws', name: 'aws-lambda-java-core', version: '1.2.1'  
    # All other project dependencies go here:  
    # ...  
    # Then, add the org.crac dependency:  
    implementation group: 'org.crac', name: 'crac', version: '1.4.0'  
}
```

## Fase 2: aggiornamento del gestore Lambda

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Per ulteriori informazioni, consulta [Definire l'handler della funzione Lambda in Java](#).

Il seguente gestore di esempio mostra come eseguire il codice prima del checkpoint (`beforeCheckpoint()`) e dopo il ripristino (`afterRestore()`). Questo gestore registra anche la `Resource` nel `Context` gestito dal runtime.

### Note

Quando Lambda crea uno snapshot, il codice di inizializzazione può essere eseguito per un massimo di 15 minuti. Il limite di tempo è 130 secondi o il [timeout della funzione configurato](#) (massimo 900 secondi), a seconda di quale dei due valori sia più elevato. I tuoi hook di runtime `beforeCheckpoint()` contano ai fini del limite di tempo del codice di inizializzazione. Quando Lambda ripristina uno snapshot, il runtime (JVM) deve essere caricato e gli hook di runtime `afterRestore()` devono essere completati entro il limite di timeout (10 secondi). Altrimenti, otterrai un `SnapStartTimeoutException`.

```

...
import org.crac.Resource;
import org.crac.Core;
...
public class CRaCDemo implements RequestStreamHandler, Resource {
    public CRaCDemo() {
        Core.getGlobalContext().register(this);
    }
    public String handleRequest(String name, Context context) throws IOException {
        System.out.println("Handler execution");
        return "Hello " + name;
    }
    @Override
    public void beforeCheckpoint(org.crac.Context<? extends Resource> context)
        throws Exception {
        System.out.println("Before checkpoint");
    }
    @Override
    public void afterRestore(org.crac.Context<? extends Resource> context)
        throws Exception {
        System.out.println("After restore");
    }
}

```

Context mantiene solo un [WeakReference](#) all'oggetto registrato. Se per una [Resource](#) viene eseguita la rimozione di oggetti inutili (garbage collection), gli hook di runtime non vengono eseguiti. Il codice deve mantenere un forte riferimento alla Resource per garantire l'esecuzione dell'hook di runtime.

Ecco due esempi di modelli da evitare:

Example: oggetto senza un forte riferimento

```
Core.getGlobalContext().register( new MyResource() );
```

Example: oggetti di classi anonime

```

Core.getGlobalContext().register( new Resource() {

    @Override
    public void afterRestore(Context<? extends Resource> context) throws Exception {
        // ...
    }
}

```

```
@Override
public void beforeCheckpoint(Context<? extends Resource> context) throws Exception {
    // ...
}

} );
```

Invece, mantieni un riferimento forte. Nell'esempio seguente, per la risorsa registrata non viene eseguita la rimozione di oggetti inutili (garbage collection) e gli hook di runtime vengono eseguiti in modo coerente.

Example: oggetto con un forte riferimento

```
Resource myResource = new MyResource(); // This reference must be maintained to prevent
the registered resource from being garbage collected
Core.getGlobalContext().register( myResource );
```

## Hook di runtime SnapStart per Lambda per Python

Gli hook di runtime possono essere utilizzati per implementare il codice prima che Lambda crei uno snapshot o dopo che ha ripristinato una funzione da uno snapshot. Gli hook di runtime Python sono disponibili come parte della libreria open source [Snapshot Restore for Python](#), che è inclusa nei runtime gestiti da Python. Questa libreria fornisce due decorator che possono essere utilizzati per definire i tuoi hook di runtime:

- `@register_before_snapshot`: per le funzioni che si desidera eseguire prima che Lambda crei uno snapshot.
- `@register_after_restore`: per le funzioni che si desidera eseguire quando Lambda riprende una funzione da uno snapshot.

In alternativa, è possibile utilizzare i seguenti metodi per registrare gli elementi chiamabili per gli hook di runtime:

- `register_before_snapshot(func, *args, **kwargs)`
- `register_after_restore(func, *args, **kwargs)`

## Registrazione ed esecuzione di hook di runtime

L'ordine in cui Lambda esegue gli hook di runtime è determinato dall'ordine di registrazione:



- Prima dello snapshot: eseguito nell'ordine di registrazione inverso
- Dopo lo snapshot: eseguito nell'ordine di registrazione

L'ordine di registrazione degli hook di runtime dipende da come vengono definiti gli hook. Quando si utilizzano i decorator (`@register_before_snapshot` e `@register_after_restore`), l'ordine di registrazione segue l'ordine di importazione, definizione o esecuzione nel codice. Se hai bisogno di un maggiore controllo sull'ordine di registrazione, usa i metodi `register_before_snapshot()` e `register_after_restore()` anziché i decorator.

Assicurati che tutti gli hook registrati siano stati importati e inclusi correttamente nel codice della tua funzione. Se registri gli hook di runtime in un file o modulo separato, dovrai assicurarti che il modulo venga importato, direttamente o come parte di un pacchetto più grande, nel file handler della funzione. Se il file o il modulo non viene importato nell'handler della funzione, Lambda ignorerà gli hook di runtime.

#### Note

Quando Lambda crea uno snapshot, il codice di inizializzazione può essere eseguito per un massimo di 15 minuti. Il limite di tempo è 130 secondi o il [timeout della funzione configurato](#) (massimo 900 secondi), a seconda di quale dei due valori sia più elevato. I tuoi hook di runtime `@register_before_snapshot` contano ai fini del limite di tempo del codice di inizializzazione. Quando Lambda ripristina uno snapshot, il runtime deve essere caricato e gli hook di runtime `@register_after_restore` devono essere completati entro il limite di timeout (10 secondi). Altrimenti, otterrai un'eccezione `SnapStartTimeoutException`.

## Esempio

Il seguente gestore di esempio mostra come eseguire il codice prima del checkpoint (`@register_before_snapshot`) e dopo il ripristino (`@register_after_restore`).

```
from snapshot_restore_py import register_before_snapshot, register_after_restore

def lambda_handler(event, context):
    # Handler code

@register_before_snapshot
def before_checkpoint():
    # Logic to be executed before taking snapshots
```

```
@register_after_restore
def after_restore():
    # Logic to be executed after restore
```

Per altri esempi, consulta [Ripristino di snapshot per Python](#) nel repository AWS GitHub.

## Hook di runtime SnapStart per Lambda per .NET

Gli hook di runtime possono essere utilizzati per implementare il codice prima che Lambda crei uno snapshot o dopo che ha ripristinato una funzione da uno snapshot. Gli hook di runtime .NET sono disponibili come parte del pacchetto [Amazon.Lambda.Core](#) (versione 2.5.0 o successiva). Questa libreria fornisce due metodi che possono essere utilizzati per definire i tuoi hook di runtime:

- `RegisterBeforeSnapshot()`: codice da eseguire prima della creazione dello snapshot
- `RegisterAfterSnapshot()`: codice da eseguire dopo la ripresa di una funzione da uno snapshot

### Note

Se utilizzi il [framework Lambda Annotations per .NET](#), esegui l'upgrade alla versione 1.6.0 o successiva di [Amazon.Lambda.Annotations](#) per garantire la compatibilità con SnapStart.

## Registrazione ed esecuzione di hook di runtime

Registra i tuoi hook nel codice di inizializzazione. Prendi in considerazione le seguenti linee guida basate sul [modello di esecuzione](#) della tua funzione Lambda:

- Per l'[approccio all'assembly eseguibile](#), registra i tuoi hook prima di avviare il bootstrap Lambda con `RunAsync`.
- Per l'[approccio basato sulle librerie di classi](#), registra i tuoi hook nel costruttore della classe handler.
- Per le [applicazioni ASP.NET Core](#), registra gli hook prima di chiamare il metodo `WebApplications.Run`.

Per registrare gli hook di runtime per SnapStart in .NET, utilizza i seguenti metodi:

```
Amazon.Lambda.Core.SnapshotRestore.RegisterBeforeSnapshot(BeforeCheckpoint);  
Amazon.Lambda.Core.SnapshotRestore.RegisterAfterRestore(AfterCheckpoint);
```

Quando vengono registrati più tipi di hook, l'ordine in cui Lambda esegue gli hook di runtime è determinato dall'ordine di registrazione:

- `RegisterBeforeSnapshot()`: eseguito nell'ordine di registrazione inverso
- `RegisterAfterSnapshot()`: eseguito nell'ordine di registrazione

#### Note

Quando Lambda crea uno snapshot, il codice di inizializzazione può essere eseguito per un massimo di 15 minuti. Il limite di tempo è 130 secondi o il [timeout della funzione configurato](#) (massimo 900 secondi), a seconda di quale dei due valori sia più elevato. I tuoi hook di runtime `RegisterBeforeSnapshot()` contano ai fini del limite di tempo del codice di inizializzazione. Quando Lambda ripristina uno snapshot, il runtime deve essere caricato e gli hook di runtime `RegisterAfterSnapshot()` devono essere completati entro il limite di timeout (10 secondi). Altrimenti, otterrai un'eccezione `SnapStartTimeoutException`.

## Esempio

La seguente funzione di esempio mostra come eseguire il codice prima del checkpoint (`RegisterBeforeSnapshot`) e dopo il ripristino (`RegisterAfterRestore`).

```
public class SampleClass  
{  
    public SampleClass()  
    {  
        Amazon.Lambda.Core.SnapshotRestore.RegisterBeforeSnapshot(BeforeCheckpoint);  
        Amazon.Lambda.Core.SnapshotRestore.RegisterAfterRestore(AfterCheckpoint);  
    }  
  
    private ValueTask BeforeCheckpoint()  
    {  
        // Add logic to be executed before taking the snapshot  
        return ValueTask.CompletedTask;  
    }  
}
```

```
private ValueTask AfterCheckpoint()
{
    // Add logic to be executed after restoring the snapshot
    return ValueTask.CompletedTask;
}

public APIGatewayProxyResponse FunctionHandler(APIGatewayProxyRequest request,
ILambdaContext context)
{
    // Add business logic

    return new APIGatewayProxyResponse
    {
        StatusCode = 200
    };
}
```

## Implementazione di SnapStart hook per le immagini dei contenitori

### Panoramica di

Quando si utilizza SnapStart una [funzione di immagine del](#) contenitore, il runtime deve coordinare il ciclo di vita della funzione e richiamare gli hook before-snapshot e after-restore durante le fasi appropriate del ciclo di vita. Questi hook consentono di eseguire una logica personalizzata (come l'aggiornamento delle credenziali o il reseed di generatori di numeri casuali) in alcuni punti del ciclo di vita dell'istantanea e del ripristino. Se utilizzi un runtime gestito dove SnapStart è supportato o le immagini di base corrispondenti per tali runtime (Java versione 11+, Python versione 3.12+ e .NET versione 8+), Lambda coordina il ciclo di vita per te. Registra i tuoi hook tramite l'API descritta in.

[Implementare il codice prima o dopo gli snapshot della funzione Lambda](#)

Se utilizzi le tue immagini del contenitore di base, i Runtime Interface Client (RIC) o le immagini di base di Lambda per provided.al2023 o Ruby Node.js, segui i passaggi indicati in questa pagina per utilizzarli. SnapStart

### Prerequisiti

Quando Lambda ripristina una funzione da un'istantanea, qualsiasi stato definito durante l'inizializzazione, come generatori di numeri casuali, ID univoci e credenziali memorizzate nella cache, viene condiviso in tutti gli ambienti di esecuzione ripristinati da quell'istantanea. Prima di utilizzarla SnapStart con la funzione Lambda basata su immagini di contenitori, rivedi [Gestire l'unicità](#)

con [Lambda SnapStart](#) e assicurati che i requisiti descritti in Usa generatori di numeri pseudorandom numerici (CSPRNG) crittograficamente sicuri siano soddisfatti. <https://docs.aws.amazon.com/lambda/latest/dg/snapstart-uniqueness.html#snapstart-csprng>

Dopo aver verificato che i requisiti siano soddisfatti, scegli una delle due opzioni seguenti:

1. Opzione 1: se hai bisogno di hook before-snapshot e after-restore per eseguire una logica personalizzata quando viene ripresa l'istantanea della funzione, segui le istruzioni nella sezione. [Implementazione degli hook del ciclo di vita SnapStart](#)
2. Opzione 2: se non hai bisogno di questi hook, abilita questa immagine del contenitore specificando la seguente etichetta all'interno del tuo SnapStart Dockerfile:

```
LABEL com.amazonaws.lambda.feature.snapstart="Allow"
```

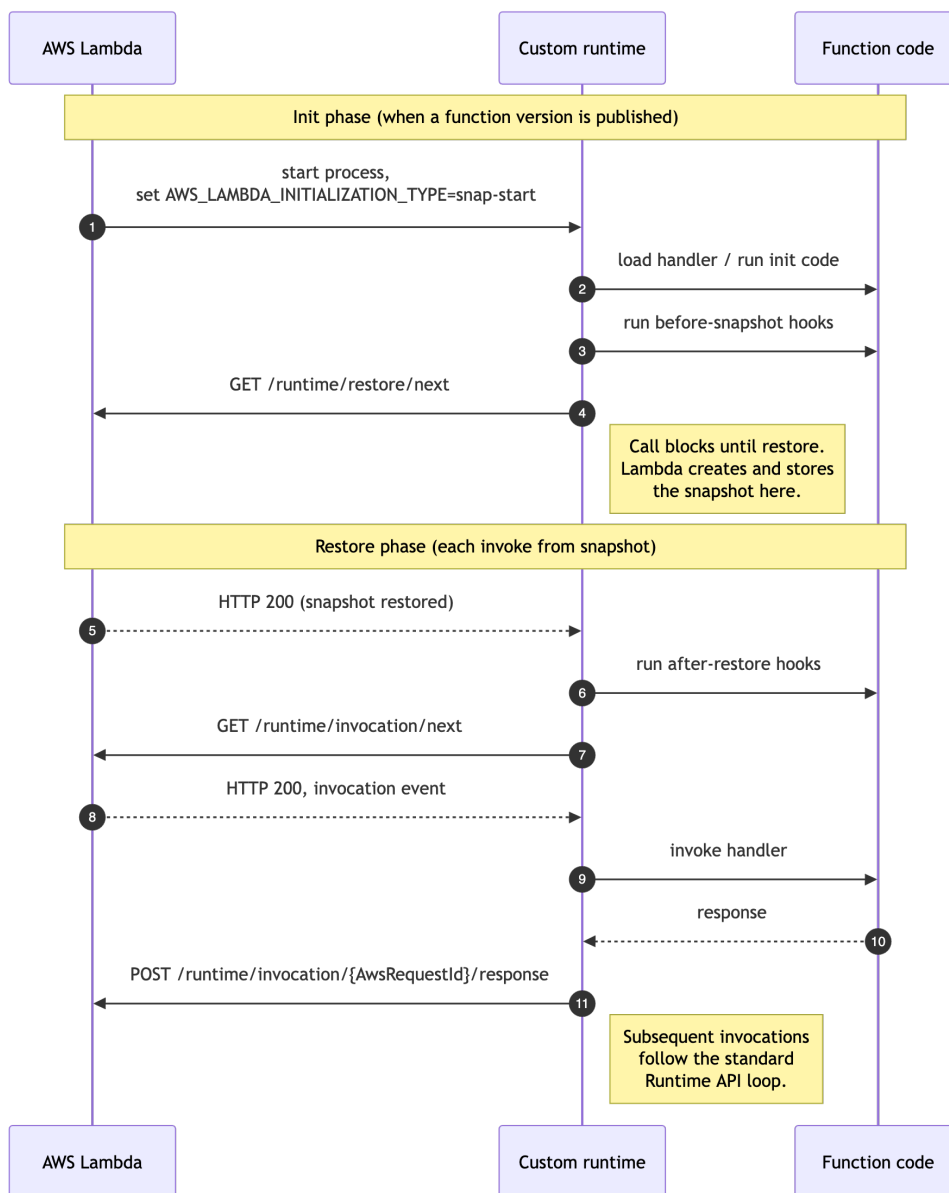
Se l'immagine del contenitore non implementa `/restore/nextAPI` né include l'etichetta, la pubblicazione della versione fallirà.

#### Note

Questi prerequisiti non sono richiesti se utilizzi immagini di base gestite da Lambda per Java (versione 11+), Python (versione 3.12+) e .NET (versione 8+) poiché coordinano già il ciclo di vita e forniscono i requisiti di unicità. SnapStart

## Panoramica del ciclo di vita

Il diagramma seguente mostra l'ordine delle chiamate API Runtime che un runtime SnapStart personalizzato dovrebbe effettuare. Tutte le chiamate fanno parte del contratto Runtime API; mentre le chiamate #3, #4, #5 e #6 sono specifiche per SnapStart.



## Implementazione degli hook del ciclo di vita SnapStart

Per utilizzarle SnapStart con le funzioni relative alle immagini del contenitore, procedi nel seguente modo:

1. Esegui gli hook before-snapshot e attiva il processo di snapshotting: come ultimo passaggio del codice di inizializzazione della funzione, esegui gli hook before-snapshot, se necessario, e attiva il processo di snapshotting. Esegui questi passaggi solo se è abilitato, controllando che il valore della variabile di ambiente sia impostato su. SnapStart `AWS_LAMBDA_INITIALIZATION_TYPE snap-start` Esegui gli hook before-snapshot registrati, quindi chiama `GET /runtime/restore/next` per attivare il processo di snapshotting. Se un hook before-snapshot genera o

restituisce un errore, il runtime invia l'errore all'endpoint. `/runtime/init/error` Guarda gli esempi di pseudo-codice qui sotto:

```
# After all initialization code has finished:

    READ initialization_type FROM environment variable
    "AWS_LAMBDA_INITIALIZATION_TYPE"

    IF initialization_type IS "snap-start" THEN

        TRY
            EXECUTE registered before-snapshot hooks
        ON ERROR
            POST error to /runtime/init/error
                SET header Lambda-Runtime-Function-Error-Type TO
                <Category>.<Reason>
                SET body TO { errorMessage, errorType, stackTrace }
            EXIT process with non-zero code

        # Signal readiness for snapshot
        SEND GET request to /runtime/restore/next
        # The request blocks until Lambda restores the execution environment from
        the snapshot, then returns HTTP 200.

    END IF
```

### Note

Gli hook init phase e before-snapshot condividono un timeout combinato di `max(function_timeout, 130 seconds)`. Se questo limite viene superato, Lambda non risponde alla richiesta. `PublishVersion` Inoltre, ad esempio `/runtime/invoke/next`, una `/runtime/restore/next` chiamata è una chiamata bloccante. Si blocca finché Lambda non ripristina l'ambiente di esecuzione dall'istantanea.

2. Esegui gli hook dopo il ripristino, quindi entra nel ciclo di invoke. Quando `GET /runtime/restore/next` restituisce 200, il runtime deve eseguire tutti gli hook post-ripristino registrati prima di procedere al ciclo di invoke. Se un hook dopo il ripristino fallisce, segnala l'errore a `/runtime/restore/error`. Una volta completati gli hook dopo il ripristino, entrate nel ciclo di invocazione standard chiamando `GET /runtime/invoke/next`. Da questo punto in poi,

il comportamento è identico a una funzione che non utilizza. SnapStart Guarda gli esempi di pseudo-codice qui sotto:

```
# After the snapshot has been restored
# (i.e., GET /runtime/restore/next has returned HTTP 200):

    TRY
        EXECUTE registered after-restore hooks
    ON ERROR
        POST error to /runtime/restore/error
            SET header Lambda-Runtime-Function-Error-Type T0 <Category>.<Reason>
            SET body    T0 { errorMessage, errorType, stackTrace }

# Proceed to the invoke loop
```

## Gestione degli errori

Se un hook fallisce, il runtime deve segnalare l'errore all'endpoint API appropriato e uscire dal processo. La tabella seguente riassume il comportamento per ciascuna fase:

| Fase                      | Endpoint API di errore      | Cosa succede in caso di errore                                                                   |
|---------------------------|-----------------------------|--------------------------------------------------------------------------------------------------|
| Istantanea iniziata/prima | POST /runtime/init/error    | Lambda non risponde alla richiesta. PublishVersion Uscire dal processo.                          |
| After-restore             | POST /runtime/restore/error | Lambda fallisce l'invocazione in volo e distrugge l'ambiente di esecuzione. Uscire dal processo. |

Per entrambi gli endpoint API, imposta l'`Lambda-Runtime-Function-Error-Type` intestazione su un valore nel formato `<Category>.<Reason>` (ad esempio, `Runtime.BeforeSnapshotError` o `Runtime.AfterRestoreError`). Includi un corpo dell'errore con `errorMessage`, `errorType`, e un opzionale. `stackTrace`

Per le specifiche complete degli endpoint API e i codici di risposta, consulta [Errore di inizializzazione](#) e [Errore di ripristino \(applicabile solo per\) SnapStart](#) nel riferimento all'API Runtime.



## Monitoraggio per Lambda SnapStart

È possibile monitorare le funzioni Lambda SnapStart utilizzando Amazon CloudWatch, AWS X-Ray e [l'Accesso ai dati di telemetria in tempo reale per le estensioni tramite l'API Telemetry](#).

### Note

Le [variabili di ambiente](#) `AWS_LAMBDA_LOG_GROUP_NAME` e `AWS_LAMBDA_LOG_STREAM_NAME` non sono disponibili nelle funzioni Lambda SnapStart.

## Informazioni sul comportamento di registrazione e fatturazione con SnapStart

Esistono alcune differenze con il formato del [flusso di log di CloudWatch](#) per le funzioni SnapStart:

- **Log di inizializzazione:** quando viene creato un nuovo ambiente di esecuzione, il REPORT non include il campo `Init Duration`. Questo perché Lambda inizializza le funzioni SnapStart quando si crea una versione anziché durante l'invocazione della funzione. Per le funzioni SnapStart, il campo `Init Duration` è nel record `INIT_REPORT`. Questo record mostra i dettagli della durata di [Fase di init](#), inclusa la durata di eventuali [hook di runtime](#) `beforeCheckpoint`.
- **Log di invocazione:** quando viene creato un nuovo ambiente di esecuzione, il REPORT include i campi `Restore Duration` e `Billed Restore Duration`:
  - **Restore Duration:** il tempo impiegato da Lambda per ripristinare uno snapshot, caricare il runtime (JVM) ed eseguire qualsiasi hook di runtime. Il processo di ripristino degli snapshot può includere il tempo dedicato ad attività esterne alla MicroVM. Questo tempo non è riportato in `Restore Duration`.
  - **Billed Restore Duration:** il tempo impiegato da Lambda per ripristinare uno snapshot, caricare il runtime (JVM) ed eseguire qualsiasi hook di runtime.

### Note

Come per tutte le funzioni Lambda, i costi di durata si applicano al codice eseguito nell'handler delle funzioni. Per le funzioni SnapStart, i costi di durata si applicano anche al codice di inizializzazione dichiarato all'esterno dell'handler, al tempo necessario per il caricamento del runtime e a qualsiasi codice eseguito in un [hook di runtime](#).

La durata dell'avviamento a freddo è la somma di `Restore Duration` e `Duration`.

L'esempio seguente è una query Lambda Insights che restituisce i percentili di latenza per le funzioni SnapStart. Per ulteriori informazioni sulle query Lambda Insights, consulta [Esempio di flusso di lavoro utilizzando query per risolvere i problemi di una funzione](#).

```
filter @type = "REPORT"
  | parse @log /\d+:\aws\lambda\(?<function>.*)/
  | parse @message /Restore Duration: (?<restoreDuration>.*?) ms/
  | stats
count(*) as invocations,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 50) as p50,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 90) as p90,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 99) as p99,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 99.9) as p99.9
group by function, (ispresent(@initDuration) or ispresent(restoreDuration)) as
coldstart
  | sort by coldstart desc
```

## Tracciamento attivo X-Ray per SnapStart

È possibile utilizzare [X-Ray](#) per tracciare le richieste alle funzioni Lambda SnapStart. I sottosegmenti X-Ray per le funzioni SnapStart presentano alcune differenze:

- Non esiste un sottosegmento `Initialization` per le funzioni SnapStart.
- Il sottosegmento mostra il tempo impiegato da Lambda per ripristinare uno snapshot, caricare il runtime (JVM) ed eseguire qualsiasi hook di runtime. Il processo di ripristino degli snapshot può includere il tempo dedicato ad attività esterne alla MicroVM. Questa volta è riportato nel segmento secondario `Restore`. Non ti viene addebitato il tempo trascorso fuori dalla microVM per il ripristino di una snapshot.

## Eventi dell'API di telemetria per SnapStart

Lambda invia i seguenti eventi SnapStart all'[API di telemetria](#):

- [Platform.RestoreStart](#): mostra l'ora in cui è iniziata la [fase Restore](#).
- [piattaforma. Restore RuntimeDone](#): indica se la fase `Restore` è riuscita correttamente. Lambda genera questo messaggio quando il runtime invia una richiesta API di runtime `restore/next`. Ci sono tre stati possibili: riuscito, errore e timeout.

- [Rapporto Platform.restore](#): mostra quanto è durata la fase Restore e quanti millisecondi sono stati fatturati durante questa fase.

## Parametri di Gateway Amazon API e della funzione URL

Se crei un'API Web [utilizzando Gateway API](#), puoi utilizzare il parametro [IntegrationLatency](#) per misurare la latenza end-to-end (il tempo che intercorre tra il momento in cui Gateway API inoltra una richiesta al backend e il momento in cui riceve una risposta dal backend).

Se utilizzi [l'URL di una funzione Lambda](#), puoi utilizzare il parametro [URLRequestLatency](#) per misurare la latenza end-to-end (il tempo che intercorre tra il momento in cui la funzione URL riceve una richiesta e il momento in cui restituisce una risposta).

## Modello di sicurezza per Lambda SnapStart

AWS Lambda SnapStart supporta la crittografia a riposo. Lambda crittografa le istantanee con un. AWS KMS key Per impostazione predefinita, Lambda utilizza un. Chiave gestita da AWS Se questo comportamento predefinito si adatta al flusso di lavoro, non è necessario impostare altro.

Altrimenti, puoi utilizzare l' `--kms-key-arn` opzione nel comando [create-function](#) o [update-function-configuration](#) per fornire una chiave gestita dal cliente. AWS KMS È possibile eseguire questa operazione per controllare la rotazione della chiave KMS o per soddisfare i requisiti dell'organizzazione per la gestione delle chiavi KMS. Le chiavi gestite dal cliente sono soggette a costi AWS KMS standard. Per ulteriori informazioni, consultare [Prezzi di AWS Key Management Service](#).

Se la tua funzione è anche una funzione durevole, la chiave gestita dal cliente che crittografa le SnapStart istantanee è indipendente dalla chiave gestita dal cliente che crittografa i dati di esecuzione duraturi. Per ulteriori informazioni, consulta [Crittografia AWS Lambda dati di esecuzione durevoli](#).

Quando si elimina una SnapStart funzione o una versione della funzione, tutte le Invoke richieste a tale funzione o versione della funzione hanno esito negativo. Lambda rimuove tutte le risorse associate agli snapshot eliminati in conformità con il Regolamento generale sulla protezione dei dati (GDPR).

# Massimizza le prestazioni Lambda SnapStart

## Argomenti

- [Ottimizzazione prestazioni](#)
- [Best practice per la rete](#)

## Ottimizzazione prestazioni

Per massimizzare i vantaggi di SnapStart, considera i seguenti consigli di ottimizzazione del codice per il tuo runtime.

### Note

SnapStart funziona meglio se utilizzato con invocazioni di funzioni su larga scala. Le funzioni richiamate di rado potrebbero non presentare gli stessi miglioramenti delle prestazioni.

## Java

Per massimizzare i vantaggi di SnapStart, si consiglia di precaricare le dipendenze e inizializzare le risorse che contribuiscono alla latenza di avvio nel codice di inizializzazione anziché nel gestore delle funzioni. Ciò sposta la latenza associata al caricamento intensivo delle classi fuori dal percorso di richiamo, ottimizzando le prestazioni di avvio con SnapStart.

Se non riesci a precaricare le dipendenze o le risorse durante l'inizializzazione, ti consigliamo di precargarle con invocazioni fittizie. A tale scopo, aggiorna il codice del gestore della funzione, come mostrato nell'esempio seguente tratto dalla funzione <https://github.com/awslabs/aws-serverless-java-container/tree/main/samples/spring/pet-store> pet store nel repository Labs. AWS GitHub

```
private static SpringLambdaContainerHandler<AwsProxyRequest, AwsProxyResponse> handler;
static {
    try {
        handler =
            SpringLambdaContainerHandler.getAwsProxyHandler(PetStoreSpringAppConfig.class);

        // Use the onStartup method of the handler to register the custom filter
        handler.onStartup(servletContext -> {
            FilterRegistration.Dynamic registration =
                servletContext.addFilter("CognitoIdentityFilter", CognitoIdentityFilter.class);
```

```

        registration.addMappingForUrlPatterns(EnumSet.of(DispatcherType.REQUEST),
false, "/*");
    });

    // Send a fake Amazon API Gateway request to the handler to load classes
ahead of time
    ApiGatewayRequestIdentity identity = new ApiGatewayRequestIdentity();
    identity.setApiKey("foo");
    identity.setAccountId("foo");
    identity.setAccessKey("foo");

    AwsProxyRequestContext reqCtx = new AwsProxyRequestContext();
    reqCtx.setPath("/pets");
    reqCtx.setStage("default");
    reqCtx.setAuthorizer(null);
    reqCtx.setIdentity(identity);

    AwsProxyRequest req = new AwsProxyRequest();
    req.setHttpMethod("GET");
    req.setPath("/pets");
    req.setBody("");
    req.setRequestContext(reqCtx);

    Context ctx = new TestContext();
    handler.proxy(req, ctx);

} catch (ContainerInitializationException e) {
    // if we fail here. We re-throw the exception to force another cold start
    e.printStackTrace();
    throw new RuntimeException("Could not initialize Spring framework", e);
}
}

```

## Python

Per massimizzare i vantaggi di SnapStart, concentratevi sull'organizzazione efficiente del codice e sulla gestione delle risorse all'interno delle funzioni Python. Come linea guida generale, esegui attività di calcolo complesse durante la [fase di inizializzazione](#). Questo approccio elimina dal percorso di invocazione le operazioni che richiedono molto tempo, migliorando le prestazioni complessive delle funzioni. Per implementare questa strategia in modo efficace, consigliamo le seguenti best practice:

- Importa le dipendenze al di fuori dell'handler della funzione.

- Crea istanze boto3 al di fuori dell'handler.
- Inizializza le risorse o le configurazioni statiche prima che l'handler venga richiamato.
- Prendi in considerazione l'utilizzo di un [hook di runtime](#) prima dello snapshot per attività che richiedono molte risorse come il download di file esterni, il precaricamento di framework come Django o il caricamento di modelli di machine learning.

### Example— Ottimizza la funzione Python per SnapStart

```
# Import all dependencies outside of Lambda handler
from snapshot_restore_py import register_before_snapshot
import boto3
import pandas
import pydantic

# Create S3 and SSM clients outside of Lambda handler
s3_client = boto3.client("s3")

# Register the function to be called before snapshot
@register_before_snapshot
def download_llm_models():
    # Download an object from S3 and save to tmp
    # This files will persist in this snapshot
    with open('/tmp/FILE_NAME', 'wb') as f:
        s3_client.download_fileobj('amzn-s3-demo-bucket', 'OBJECT_NAME', f)
    ...

def lambda_handler(event, context):
    ...
```

## .NET

Per ridurre i tempi di compilazione just-in-time (JIT) e di caricamento degli assembly, considera la possibilità di richiamare l'handler della funzione da un [hook di runtime](#) `RegisterBeforeCheckpoint`. Grazie al funzionamento della compilazione su più livelli .NET, otterrete risultati ottimali richiamando il gestore più volte, come mostrato nell'esempio seguente.

 Important

Assicurati che l'invocazione della funzione fittizia non produca effetti collaterali indesiderati, come l'avvio di transazioni commerciali.

## Example

```
public class Function
{
    public Function()
    {
        Amazon.Lambda.Core.SnapshotRestore.RegisterBeforeSnapshot(FunctionWarmup);
    }

    // Warmup method that calls the function handler before snapshot to warm up
    the .NET code and runtime.
    // This speeds up future cold starts after restoring from a snapshot.

    private async ValueTask FunctionWarmup()
    {
        var request = new APIGatewayProxyRequest
        {
            Path = "/healthcheck",
            HttpMethod = "GET"
        };

        for (var i = 0; i < 10; i++)
        {
            await FunctionHandler(request, null);
        }
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
request, ILambdaContext context)
    {
        //
        // Process HTTP request
        //

        var response = new APIGatewayProxyResponse
        {
            StatusCode = 200
        }
    }
}
```



```
};  
  
    return await Task.FromResult(response);  
}  
}
```

## Best practice per la rete

Lo stato delle connessioni che la funzione stabilisce durante la fase di inizializzazione non è garantito quando Lambda riprende la funzione da uno snapshot. Nella maggior parte dei casi, le connessioni di rete stabilite da un AWS SDK vengono ripristinate automaticamente. Per altre connessioni, consigliamo le seguenti best practice.

### Re-establish connessioni di rete

Ristabilisci sempre le connessioni di rete quando la funzione riprende da uno snapshot. Si consiglia di ristabilire le connessioni di rete nel gestore delle funzioni. In alternativa, puoi usare un [hook di runtime](#) dopo il ripristino.

### Non utilizzare hostname come identificatore univoco dell'ambiente di esecuzione

Si consiglia di non utilizzare `hostname` per identificare l'ambiente di esecuzione come nodo o container univoco nelle applicazioni. Con SnapStart, una singola istantanea viene utilizzata come stato iniziale per più ambienti di esecuzione. Tutti gli ambienti di esecuzione restituiscono lo stesso valore `hostname` per `InetAddress.getLocalHost()` (Java), `socket.gethostname()` (Python) e `Dns.GetHostName()` (.NET). Per le applicazioni che richiedono un'identità dell'ambiente di esecuzione o un valore `hostname` univoco, si consiglia di generare un ID univoco nel gestore della funzione. Oppure, utilizza un [hook di runtime](#) dopo il ripristino per generare un ID univoco, quindi utilizza l'ID univoco come identificatore per l'ambiente di esecuzione.

### Evitare di collegare connessioni a porte di origine fisse

Si consiglia di evitare di associare le connessioni di rete a porte di origine fisse. Le connessioni vengono ristabilite quando una funzione riprende da uno snapshot e le connessioni di rete legate a una porta di origine fissa potrebbero non riuscire.

### Evitare di usare la cache DNS Java

Le funzioni Lambda memorizzano già nella cache le risposte DNS. Se si utilizza un'altra cache DNS con SnapStart, è possibile che si verifichino dei timeout di connessione quando la funzione riprende da un'istantanea.

La classe `java.util.logging.Logger` può abilitare indirettamente la cache DNS JVM. Per sovrascrivere le impostazioni predefinite, imposta [networkaddress.cache.ttl](#) su 0 prima dell'inizializzazione di `logger`. Esempio:

```
public class MyHandler {
    // first set TTL property
    static{
        java.security.Security.setProperty("networkaddress.cache.ttl" , "0");
    }
    // then instantiate logger
    var logger = org.apache.logging.log4j.LogManager.getLogger(MyHandler.class);
}
```

Per evitare errori `UnknownHostException` nel runtime di Java 11, si consiglia di impostare `networkaddress.cache.negative.ttl` su 0. Nei runtime di Java 17 e successivi, questa operazione non è necessaria. È possibile impostare questa proprietà per una funzione Lambda con la variabile di ambiente `AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0`.

La disabilitazione della cache DNS JVM non disabilita la cache DNS gestita da Lambda.

# Risoluzione degli SnapStart errori per le funzioni Lambda

Questa pagina risolve i problemi più comuni che si verificano quando si utilizza Lambda SnapStart, inclusi errori di creazione di istantanee, errori di timeout ed errori interni del servizio.

## SnapStartNotReadyException

Errore: si è verificato un errore (SnapStartNotReadyException) durante la chiamata all'operazione Invoke20150331: Lambda sta inizializzando la funzione. Sarà pronta per essere richiamata una volta che lo stato della funzione diventerà ATTIVO.

### Cause comuni

Questo errore si verifica quando si prova a richiamare una versione della funzione che si trova nello [stato Inactive](#). La versione della funzione diventa Inactive quando non viene richiamata per 14 giorni o quando Lambda riavvia periodicamente l'ambiente di esecuzione

### Risoluzione

Attendi che la versione della funzione raggiunga lo stato Active, quindi richiamala di nuovo.

## SnapStartTimeoutException

Problema: viene visualizzato un messaggio **SnapStartTimeoutException** quando si tenta di richiamare la versione di una funzione. SnapStart

### Cause comuni

Durante la fase di [ripristino](#), Lambda ripristina il runtime Java ed esegue qualsiasi [hook di runtime](#) successivo al ripristino. Se un hook di runtime post-ripristino viene eseguito per più di 10 secondi, la fase Restore scade e viene visualizzato un errore quando si prova a richiamare la funzione. I problemi relativi alla connessione di rete e alle credenziali possono anche causare un timeout per la fase Restore.

### Risoluzione

Controlla i CloudWatch log della funzione per individuare eventuali errori di timeout verificatisi durante la fase di [ripristino](#). Assicurati che tutti gli hook post-ripristino vengano completati in meno di 10 secondi.

## Example CloudWatch registro

```
{ "cause": "Lambda couldn't restore the snapshot within the timeout limit. (Service: Lambda, Status Code: 408, Request ID: 11a222c3-410f-427c-ab22-931d6bcbf4f2)", "error": "Lambda.SnapStartTimeoutException" }
```

## Errore interno del servizio 500

Errore: Lambda non è riuscito a creare un nuovo snapshot perché è stato raggiunto il limite di creazione simultanea di snapshot.

### Cause comuni

Un errore 500 è un errore interno al servizio Lambda stesso e non un problema relativo alla funzione o al codice. Questi errori sono spesso intermittenti.

### Risoluzione

Prova a pubblicare nuovamente la versione della funzione.

## 401 - Autorizzazione negata

Errore: token di sessione o chiave di intestazione errati

### Cause comuni

Questo errore si verifica quando si utilizzano il [AWS Systems Manager Parameter Store e AWS Secrets Manager l'estensione](#) con Lambda SnapStart. Quando la funzione utilizza SnapStart, ottiene le credenziali da un endpoint del contenitore e Lambda non imposta la variabile di ambiente. `AWS_SESSION_TOKEN` Se la funzione imposta `X-Aws-Parameters-Secrets-Token` intestazione da quella variabile di ambiente, l'intestazione è vuota e l'estensione rifiuta la richiesta.

### Risoluzione

Per correggere questo errore, utilizzate la catena di provider di credenziali AWS SDK per recuperare il token di sessione invece di leggere direttamente la variabile di ambiente. `AWS_SESSION_TOKEN` Imposta `X-Aws-Parameters-Secrets-Token` intestazione sul token restituito dalla catena di provider. La catena di provider di credenziali restituisce il token di sessione corretto in tutte le modalità di inizializzazione, incluse. SnapStart Per il codice di esempio, consulta [Uso dei segreti in Secrets Manager nelle funzioni Lambda](#).

## UnknownHostException (Java)

Errore: impossibile eseguire la richiesta HTTP: il certificato per `abc.us-east-1.amazonaws.com` non corrisponde a nessuno dei nomi alternativi del soggetto.

### Cause comuni

Le funzioni Lambda memorizzano già nella cache le risposte DNS. Se si utilizza un'altra cache DNS con SnapStart, è possibile che si verifichino dei timeout di connessione quando la funzione riprende da un'istantanea.

### Risoluzione

Per evitare errori `UnknownHostException` nel runtime di Java 11, si consiglia di impostare `networkaddress.cache.negative.ttl` su 0. Nei runtime di Java 17 e successivi, questa operazione non è necessaria. È possibile impostare questa proprietà per una funzione Lambda con la variabile di ambiente `AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0`.

## Errori di creazione snapshot

Errore: AWS Lambda impossibile richiamare la funzione. SnapStart Se l'errore persiste, controlla i CloudWatch log della tua funzione per eventuali errori di inizializzazione.

### Risoluzione

Controlla i CloudWatch log Amazon della tua funzione per verificare i timeout degli hook in fase di esecuzione prima del checkpoint. Puoi anche provare a pubblicare una nuova versione della funzione, che a volte può risolvere il problema.

## Frequenza di creazione degli snapshot

Problema: quando si pubblica una nuova versione della funzione, la funzione rimane nello [stato Pending](#) per un lungo periodo.

### Cause comuni

Quando Lambda crea uno snapshot, il codice di inizializzazione può essere eseguito per un massimo di 15 minuti. Il limite di tempo è 130 secondi o il [timeout della funzione configurato](#) (massimo 900 secondi), a seconda di quale dei due valori sia più elevato.

Se la funzione è [collegata a un VPC](#), Lambda potrebbe anche dover creare interfacce di rete prima che la funzione diventi `Active`. Se provi a richiamare la versione della funzione mentre la funzione

è Pending, potresti ottenere un `ResourceConflictException` 409. Se la funzione viene richiamata utilizzando un endpoint Gateway Amazon API, potresti ricevere un errore 500 in API Gateway.

## Risoluzione

Attendi almeno 15 minuti per l'inizializzazione della versione della funzione prima di richiamarla.

# Isolamento degli inquilini

Usa la modalità di isolamento dei tenant quando hai bisogno di un'elaborazione isolata delle richieste per singoli utenti finali o tenant che richiamano una funzione Lambda. Questa funzionalità semplifica la creazione di applicazioni multi-tenant che elaborano codice o dati specifici del tenant, come le piattaforme SaaS per l'automazione del flusso di lavoro o l'esecuzione di codice, eliminando la necessità di gestire le risorse funzionali specifiche del tenant e richiedere la logica di routing.

Multi-tenant le applicazioni hanno severi requisiti di isolamento durante l'esecuzione del codice o l'elaborazione di dati per singoli tenant o utenti finali. Con la modalità di isolamento dei tenant, Lambda utilizza un identificatore tenant specificato dal cliente per indirizzare le richieste agli ambienti di esecuzione delle funzioni sottostanti, assicurando che gli ambienti di esecuzione di una funzione vengano utilizzati solo per fornire chiamate dall'utente finale o dal tenant specificato. Gli ambienti di esecuzione delle funzioni di Lambda utilizzano la virtualizzazione Firecracker per fornire l'isolamento del carico di lavoro. <https://firecracker-microvm.github.io/>

Quando una funzione che utilizza la modalità di isolamento del tenant riceve un richiamo con un identificatore tenant, Lambda tenta innanzitutto di individuare un ambiente di esecuzione disponibile associato a tale identificatore tenant. Se non esistono ambienti di esecuzione, Lambda crea e assegna un nuovo ambiente di esecuzione a quel tenant. Man mano che le chiamate di funzione con l'identificatore del tenant specificato aumentano, Lambda individua o crea nuovi ambienti di esecuzione, se necessario.

Per informazioni su come utilizzare l'isolamento dei tenant, consulta. [the section called “Abilitare l'isolamento dei tenant”](#) Per informazioni sull'accesso al contesto dei tenant nel codice, consulta. [the section called “Accesso tramite codice”](#) Per monitorare le funzioni isolate dai tenant, vedere. [the section called “Monitoraggio”](#) Per assistenza nella risoluzione dei problemi, vedere. [the section called “risoluzione dei problemi”](#)

## Argomenti

- [Quando utilizzare la modalità di isolamento dei tenant](#)

- [Funzionalità e limitazioni supportate](#)
- [Supportata Regioni AWS](#)
- [Considerazioni](#)
- [Prezzi](#)
- [Modalità di isolamento](#)
- [Abilitazione dell'isolamento dei tenant per le funzioni Lambda](#)
- [Richiamo di funzioni Lambda con isolamento del tenant](#)
- [Accesso all'identificatore del tenant nel codice della funzione Lambda](#)
- [Monitoraggio delle funzioni Lambda con isolamento dei tenant](#)
- [Risoluzione dei problemi di isolamento dei tenant per le funzioni Lambda](#)

## Quando utilizzare la modalità di isolamento dei tenant

Utilizza la modalità di isolamento dei tenant quando devi servire più utenti finali o tenant utilizzando una singola funzione Lambda, assicurando al contempo che gli ambienti di esecuzione utilizzati per elaborare le chiamate per i singoli tenant rimangano isolati l'uno dall'altro. Questo rigoroso isolamento degli ambienti di esecuzione è necessario per le applicazioni multi-tenant che:

- Esecuzione del codice fornito dall'utente finale: il mantenimento di ambienti di esecuzione separati per i singoli tenant può limitare l'impatto dell'esecuzione di codice fornito dall'utente che potrebbe essere errato o dannoso.
- Elabora dati specifici del tenant: il mantenimento di ambienti di esecuzione separati per i singoli tenant può impedire l'esposizione di dati sensibili specifici del tenant ad altri tenant.

Più richieste di invocazione provenienti dallo stesso tenant possono riutilizzare lo stesso ambiente di esecuzione delle funzioni, riducendo i cold start e migliorando i tempi di risposta per le applicazioni sensibili alla latenza.

## Funzionalità e limitazioni supportate

La modalità di isolamento dei tenant non è supportata dalle funzioni che utilizzano [gli URL delle funzioni, la concorrenza predisposta o. ??? SnapStart](#). Puoi inviare richieste a una funzione isolata dal tenant utilizzando invocazioni sincrone, [invocazioni asincrone o utilizzando Amazon API Gateway come trigger di eventi. ???](#)

## Supportata Regioni AWS

La modalità di isolamento dei tenant è supportata in tutte le regioni commerciali ad eccezione dell'Asia Pacifico (Nuova Zelanda). <https://docs.aws.amazon.com/general/latest/gr/gloss-chap.html#region>

## Considerazioni

Quando utilizzi l'isolamento dei tenant con le tue funzioni Lambda, tieni presente quanto segue:

- Configurazione immutabile: l'isolamento del tenant è una proprietà della funzione immutabile. Può essere abilitato solo quando si crea una funzione.
- Parametro tenant-id obbligatorio: le funzioni che utilizzano la modalità di isolamento dei tenant devono essere richiamate con un parametro. `tenant-id` L'omissione di questo parametro causa l'esito negativo delle chiamate alle funzioni.
- Il ruolo di esecuzione si applica a tutti i tenant: le chiamate da tutti i tenant utilizzano le autorizzazioni definite nel ruolo di esecuzione della funzione Lambda. ???
- Concorrenza: non ci sono modifiche al comportamento di concorrenza o di scalabilità della funzione quando si utilizza l'isolamento dei [tenant](#). ??? Lambda impone un limite di 2.500 ambienti di esecuzione isolati dai tenant (attivi o inattivi) per ogni 1.000 esecuzioni simultanee configurate per la funzione Lambda. ???

## Prezzi

Ti viene addebitato un costo quando Lambda crea un nuovo ambiente di esecuzione isolato dal tenant. Il prezzo dipende dalla quantità di [memoria](#) allocata alla funzione e dall'architettura della [CPU utilizzata](#). Per ulteriori informazioni, consultare [Prezzi di AWS Lambda](#).

## Modalità di isolamento

La tabella seguente illustra le differenze tra le funzioni Lambda con e senza isolamento del tenant.

| Funzionalità       | Con isolamento degli inquilini | Senza isolamento degli inquilini |
|--------------------|--------------------------------|----------------------------------|
| Tipo di isolamento | Tenant-level isolamento        | Function-level isolamento        |



| Funzionalità             | Con isolamento degli inquilini                                             | Senza isolamento degli inquilini                                                                             |
|--------------------------|----------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| Riutilizzo dell'ambiente | Gli ambienti di esecuzione non vengono mai riutilizzati tra tenant diversi | Gli ambienti di esecuzione potrebbero essere riutilizzati per più invocazioni della stessa funzione          |
| Isolamento dei dati      | I dati di altri tenant non sono accessibili                                | I dati provenienti da precedenti richiami della stessa versione della funzione potrebbero essere accessibili |
| Partenze a freddo        | Più avviamenti a freddo grazie agli ambienti specifici del locatario       | Riduzione delle partenze a freddo grazie al riutilizzo dell'ambiente                                         |
| Prezzi                   | Costi aggiuntivi oltre ai prezzi Lambda standard                           | Prezzi Lambda standard                                                                                       |

# Abilitazione dell'isolamento dei tenant per le funzioni Lambda

Per attivare la modalità di isolamento del tenant, crea una nuova funzione Lambda. Non è possibile abilitare l'isolamento dei tenant sulle funzioni esistenti.

## Argomenti

- [Abilitazione dell'isolamento dei tenant \(console\)](#)
- [Abilitazione dell'isolamento dei tenant \(\)AWS CLI](#)
- [Abilitazione dell'isolamento dei tenant \(API\)](#)
- [Abilitare l'isolamento dei tenant \(\)CloudFormation](#)

## Abilitazione dell'isolamento dei tenant (console)

Per creare una funzione Lambda utilizzando la console

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli Crea funzione.
3. Scegli Crea da zero.
4. Nel riquadro Informazioni base, per Nome funzione inserisci **image-analysis**.
5. Per Runtime, scegli uno dei runtime [Lambda supportati](#).
6. In configurazioni aggiuntive, modalità di isolamento Tenant, seleziona Abilita.
7. Controlla le impostazioni e scegli Crea funzione.

## Abilitazione dell'isolamento dei tenant ()AWS CLI

Crea una funzione con l'isolamento dei tenant

Quando crei una nuova funzione utilizzando la CLI, aggiungi l'`--tenancy-config '{"TenantIsolationMode": "PER_TENANT"}'` opzione alla tua richiesta [create-function](#).

Esempio:

```
aws lambda create-function \
  --function-name image-analysis \
  --runtime nodejs24.x \
  --zip-file fileb://image-analysis-function.zip \
  --handler image-analysis-function.handler \
```

```
--role arn:aws:iam:123456789012:role/execution-role \
--tenancy-config '{"TenantIsolationMode": "PER_TENANT"}'
```

## Abilitazione dell'isolamento dei tenant (API)

Per abilitare l'isolamento dei tenant utilizzando l'API Lambda

1. Crea una nuova funzione con l'isolamento dei tenant abilitato utilizzando l'azione [CreateFunction](#) API con il parametro. `TenancyConfig`
2. Verifica che l'isolamento del tenant sia abilitato per la funzione utilizzando l'azione [GetFunctionConfiguration](#). Se la risposta indica che `TenantIsolationMode` è `PER_TENANT`, l'isolamento del tenant è abilitato per la funzione:

```
"TenancyConfig": {
  "TenantIsolationMode": "PER_TENANT"
}
```

Richiama la versione della funzione con l'operazione [Invoke](#). Per ulteriori informazioni, consulta [Richiamo di funzioni Lambda con isolamento del tenant](#).

## Abilitare l'isolamento dei tenant (CloudFormation)

Il CloudFormation modello seguente crea una nuova funzione Lambda con l'isolamento dei tenant abilitato:

```
MyLambdaFunction:
  Type: AWS::Lambda::Function
  Properties:
    FunctionName: my-sample-python-lambda
    Runtime: python3.14
    Role: !GetAtt LambdaExecutionRole.Arn
    Handler: index.lambda_handler
    TenancyConfig:
      TenantIsolationMode: PER_TENANT
    Code:
      ZipFile: |
        import json

        def lambda_handler(event, context):
          return {
```

```
        'statusCode': 200,  
        'body': json.dumps(f'Hello from Lambda! Tenant-ID:  
{context.tenant_id}')  
    }  
    Timeout: 10  
    MemorySize: 128
```

## Richiamo di funzioni Lambda con isolamento del tenant

Quando si richiama una funzione con l'isolamento del tenant abilitato, è necessario fornire un parametro. `tenant-id` Questo parametro garantisce che la chiamata della funzione venga elaborata in un ambiente di esecuzione dedicato a quel tenant specifico.

### Richiamo di funzioni con tenant isolation ( )AWS CLI

Invocazione sincrona

[Per le chiamate sincrone, aggiungi il parametro al tuo comando Invoke: `--tenant-id`](#)

```
aws lambda invoke \  
  --function-name image-analysis \  
  --tenant-id blue \  
  response.json
```

Invocazione asincrona

Per le chiamate asincrone, includi entrambi i parametri e: `--tenant-id --invocation-type Event`

```
aws lambda invoke \  
  --function-name image-analysis \  
  --tenant-id blue \  
  --invocation-type Event \  
  response.json
```

### Richiamo di funzioni con isolamento dei tenant (API)

Quando utilizzi direttamente l'azione [Invoke](#) API, includi l'identificatore del tenant utilizzando il `X-Amzn-Tenant-Id` parametro nella richiesta.

Esempio di richiesta API

```
POST /2015-03-31/functions/image-analysis/invocations HTTP/1.1  
Host: lambda.us-east-1.amazonaws.com  
Content-Type: application/json  
Authorization: AWS4-HMAC-SHA256 Credential=...  
X-Amzn-Tenant-Id: blue  
  
{
```

```
"key1": "value1",  
"key2": "value2"  
}
```

## Richiamo di funzioni con isolamento del tenant (API Gateway)

Quando si utilizza API Gateway REST APIs per attivare funzioni Lambda isolate dal tenant, è necessario configurare API Gateway per mappare le proprietà della richiesta del client all'`X-Amz-Tenant-Id` intestazione prevista da Lambda. API Gateway utilizza l'azione [Invoke](#) API di Lambda, che richiede il passaggio dell'ID tenant tramite l'`X-Amz-Tenant-Id` intestazione HTTP. Puoi configurare API Gateway per iniettare questa intestazione HTTP nella richiesta di chiamata Lambda con un valore ottenuto dalle proprietà della richiesta del client come intestazioni HTTP, parametri di query o parametri di percorso. È necessario mappare la proprietà della richiesta del client prima di poter sovrascrivere l'intestazione. `X-Amz-Tenant-Id`

### Note

Non è possibile utilizzare HTTP APIs per richiamare funzioni Lambda isolate dal tenant perché non è possibile sovrascrivere l'intestazione. `X-Amz-Tenant-Id`

## Utilizzo delle intestazioni di richiesta

Configura l'integrazione con API Gateway per mappare un'intestazione personalizzata dalla richiesta del client all'`X-Amz-Tenant-Id` intestazione. L'esempio seguente mostra una richiesta client con un'`x-tenant-id` intestazione:

```
POST /api/process HTTP/1.1  
Host: your-api-id.execute-api.us-east-1.amazonaws.com  
Content-Type: application/json  
x-tenant-id: blue  
  
{  
  "data": "sample payload"  
}
```

Nella configurazione del metodo API Gateway, è necessario:

1. Abilitare il parametro di intestazione della richiesta del client (ad esempio, `method.request.header.x-tenant-id`)

## 2. Mappare l'intestazione del client all'intestazione di integrazione Lambda utilizzando `integration.request.header.X-Amz-Tenant-Id`

### Utilizzo dei parametri di interrogazione

Allo stesso modo, puoi mappare i parametri della query all'`X-Amz-Tenant-Id` intestazione:

```
GET /api/process?tenant-id=blue&data=sample HTTP/1.1
Host: your-api-id.execute-api.us-east-1.amazonaws.com
```

Configura il metodo per abilitare il parametro di query e mapparlo all'intestazione di integrazione.

### Richiamo di funzioni con isolamento dei tenant (SDK)

Quando si utilizza AWS SDKs per richiamare funzioni isolate dal tenant, includi l'identificatore del tenant nella richiesta di invocazione.

#### Python

```
import boto3
import json

lambda_client = boto3.client('lambda')

response = lambda_client.invoke(
    FunctionName='image-analysis',
    TenantId='blue',
    Payload=json.dumps({
        'key1': 'value1',
        'key2': 'value2'
    })
)

result = json.loads(response['Payload'].read())
```

#### Node.js

```
const AWS = require('aws-sdk');
const lambda = new AWS.Lambda();

const params = {
```

```

    FunctionName: 'image-analysis',
    TenantId: 'blue',
    Payload: JSON.stringify({
      key1: 'value1',
      key2: 'value2'
    })
  });

lambda.invoke(params, (err, data) => {
  if (err) {
    console.error(err);
  } else {
    const result = JSON.parse(data.Payload);
    console.log(result);
  }
});

```

## Java

```

import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.InvokeRequest;
import software.amazon.awssdk.services.lambda.model.InvokeResponse;
import software.amazon.awssdk.core.SdkBytes;

public class TenantIsolationExample {

    public static void main(String[] args) {
        LambdaClient lambdaClient = LambdaClient.create();

        String payload = "{\"key1\": \"value1\", \"key2\": \"value2\"}";

        InvokeRequest request = InvokeRequest.builder()
            .functionName("image-analysis")
            .tenantId("blue")
            .payload(SdkBytes.fromUtf8String(payload))
            .build();

        InvokeResponse response = lambdaClient.invoke(request);
    }
}

```



## Accesso all'identificatore del tenant nel codice della funzione Lambda

Quando la funzione Lambda ha l'isolamento del tenant abilitato, l'identificatore tenant utilizzato per richiamare la funzione viene reso disponibile all'interno dell'oggetto context passato al gestore della funzione. È possibile utilizzare questo identificatore per implementare funzionalità di logica, monitoraggio e debug specifiche del tenant.

### Argomenti

- [Accesso all'identificatore del tenant](#)
- [Modelli di utilizzo comuni](#)
- [Monitoraggio e debug](#)

### Accesso all'identificatore del tenant

L'identificatore del tenant è disponibile tramite la `tenantId` proprietà dell'oggetto context. [Si noti che questa proprietà è disponibile durante la fase di invocazione, non durante la fase di inizializzazione.](#)

### Python

```
def lambda_handler(event, context):
    tenant_id = context.tenant_id
    print(f"Processing request for tenant: {tenant_id}")

    # Implement tenant-specific logic
    if tenant_id == "blue":
        return process_blue_tenant(event)
    elif tenant_id == "green":
        return process_green_tenant(event)
    else:
        return process_default_tenant(event)
```

### Node.js

```
exports.handler = async (event, context) => {
    const tenantId = context.tenantId;
    console.log(`Processing request for tenant: ${tenantId}`);

    // Implement tenant-specific logic
    switch (tenantId) {
        case 'blue':
```

```

        return processBlueTenant(event);
    case 'green':
        return processGreenTenant(event);
    default:
        return processDefaultTenant(event);
    }
};

```

## Java

```

public class TenantHandler implements RequestHandler<Map<String, Object>, String> {

    @Override
    public String handleRequest(Map<String, Object> event, Context context) {
        String tenantId = context.getTenantId();
        System.out.println("Processing request for tenant: " + tenantId);

        // Implement tenant-specific logic
        switch (tenantId) {
            case "blue":
                return processBlueTenant(event);
            case "green":
                return processGreenTenant(event);
            default:
                return processDefaultTenant(event);
        }
    }
}

```

## Modelli di utilizzo comuni

Ecco alcuni modi comuni per utilizzare l'identificatore tenant nel codice della funzione:

Configurazione specifica per il tenant

Usa l'ID tenant per caricare configurazioni o impostazioni specifiche del tenant:

```

def lambda_handler(event, context):
    tenant_id = context.tenant_id

    # Load tenant-specific configuration
    config = load_tenant_config(tenant_id)

```

```
database_url = config['database_url']
api_key = config['api_key']

# Process with tenant-specific settings
return process_request(event, database_url, api_key)
```

Accesso ai dati specifico del tenant

Utilizza l'ID del tenant per garantire l'isolamento dei dati e il controllo degli accessi:

```
import boto3

def lambda_handler(event, context):
    tenant_id = context.tenant_id

    # Ensure data access is scoped to the tenant
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('user_data')

    user_id = event.get('userId')

    response = table.get_item(
        Key={
            'tenant_id': tenant_id,
            'user_id': user_id
        }
    )

    return process_results(response.get('Item'), tenant_id)
```

## Monitoraggio e debug

L'identificatore del tenant viene incluso automaticamente nei log Lambda quando è [abilitata la registrazione JSON, semplificando](#) il monitoraggio e il debug dei problemi specifici del tenant. Puoi anche utilizzare l'ID tenant per metriche e tracciamento personalizzati.

### Example Metriche personalizzate con ID tenant

L'esempio seguente mostra come utilizzare l'ID tenant per creare CloudWatch metriche specifiche per il tenant per monitorare i modelli di utilizzo e le prestazioni per tenant:

```
import boto3
```

```
def lambda_handler(event, context):
    tenant_id = context.tenant_id
    cloudwatch = boto3.client('cloudwatch')

    # Record tenant-specific metrics
    cloudwatch.put_metric_data(
        Namespace='MyApp/TenantMetrics',
        MetricData=[
            {
                'MetricName': 'RequestCount',
                'Dimensions': [
                    {
                        'Name': 'TenantId',
                        'Value': tenant_id
                    }
                ],
                'Value': 1,
                'Unit': 'Count'
            }
        ]
    )

    return process_request(event, tenant_id)
```

## Monitoraggio delle funzioni Lambda con isolamento dei tenant

[Puoi monitorare le funzioni Lambda isolate dai tenant utilizzando CloudWatch AWS X-Ray Amazon e accedendo ai dati di telemetria in tempo reale per le estensioni utilizzando l'API di telemetria.](#)

### Comprendere la registrazione per la modalità isolata dei tenant

Per le funzioni che utilizzano l'isolamento dei tenant, Lambda include automaticamente l'identificatore del tenant [nei registri delle funzioni quando è abilitata la registrazione JSON, semplificando](#) il monitoraggio e il debug di problemi specifici del tenant. Lambda crea un [flusso di CloudWatch log](#) separato per ogni ambiente di esecuzione. Puoi utilizzare CloudWatch Logs Insights per trovare i flussi di log che appartengono a un particolare tenant filtrandoli per identificatore del tenant:

```
fields @logStream, @message
| filter tenantId=='BlueTenant' or record.tenantId=='BlueTenant'
| stats count() as logCount by @logStream
| sort @timestamp desc
```

È inoltre possibile utilizzare questo parametro per recuperare i log specifici del tenant in tutti i flussi di log:

```
fields @message
| filter tenantId=='BlueTenant' or record.tenantId=='BlueTenant'
| limit 1000
```

La `tenantId` proprietà è inclusa per gli eventi delle piattaforme (come `platform.start` e `platform.report`) e i log personalizzati stampati nel codice della funzione, come illustrato di seguito:

```
{
  "time": "2025-10-13T19:48:06.990Z",
  "type": "platform.start",
  "record": {
    "requestId": "a0f40320-b43c-44b3-91bf-d5b5240a1bed",
    "functionArn": "arn:aws:lambda:us-east-1:xxxxxx:function:multitenant-
function-1",
    "version": "$LATEST",
    "tenantId": "BlueTenant"
  }
}
```

```
{
  "timestamp": "2025-10-13T19:48:06.992Z",
  "level": "INFO",
  "requestId": "a0f40320-b43c-44b3-91bf-d5b5240a1bed",
  "tenantId": "BlueTenant",
  "message": "custom log line1"
}
{
  "timestamp": "2025-10-13T19:48:07.022Z",
  "level": "WARN",
  "requestId": "a0f40320-b43c-44b3-91bf-d5b5240a1bed",
  "tenantId": "BlueTenant",
  "message": "custom log line2"
}
```

# Risoluzione dei problemi di isolamento dei tenant per le funzioni Lambda

Questa pagina affronta i problemi più comuni che si verificano quando si utilizza l'isolamento dei tenant per. AWS Lambda

## InvalidParameterValueException

Errore: configurazione dell'ID tenant non specificata o passata a una funzione quando l'isolamento del tenant non è abilitato.

### Cause comuni

Questo errore si verifica quando si richiama una funzione isolata dal tenant senza un ID tenant o si richiama una funzione con un ID tenant. non-tenant-isolated

### Risoluzione

Aggiungi un ID tenant se la funzione ha l'isolamento del tenant abilitato o rimuovi l'ID tenant se la funzione non ha l'isolamento del tenant abilitato.

## TooManyRequestsException

Errore: tariffa superata

### Cause comuni

Oltre alla limitazione della velocità basata sul numero [massimo di esecuzioni simultanee](#) e sulla [velocità di scalabilità delle funzioni](#), Lambda limita il numero massimo di ambienti di esecuzione con riconoscimento del tenant (attivi o inattivi) che possono esistere contemporaneamente a 2.500 per ogni 1.000 esecuzioni simultanee della funzione.

### Risoluzione

Per risolvere questo problema, puoi ridurre la frequenza con cui vengono effettuate le richieste di invocazione con identificatori di tenant univoci, [implementare nuovi tentativi con backoff e jitter](#) o [richiedere un aumento del limite di concomitanza delle funzioni](#).

## Istanze Lambda gestite

Con Lambda Managed Instances, puoi eseguire funzioni Lambda sulle istanze Amazon EC2 di ultima generazione, tra cui Graviton4, istanze ottimizzate per la rete e altre opzioni di calcolo specializzate,

senza dover gestire i cicli di vita delle istanze, l'applicazione di patch in fase di esecuzione del sistema operativo e del linguaggio, il routing, il bilanciamento del carico o le politiche di scalabilità. Con le istanze gestite Lambda, puoi beneficiare dei vantaggi dei prezzi di EC2, inclusi i piani di risparmio EC2 e le istanze riservate.

Per un elenco dei tipi di istanze supportati, vai alla pagina [AWS Lambda Prezzi](#) e seleziona il tuo. Regione AWS

## Funzionalità chiave

Lambda Managed Instances offre le seguenti funzionalità:

- Scegli le istanze adatte: seleziona le istanze [appropriate in](#) base ai requisiti di prestazioni e costi, incluso l'accesso alle CPU più recenti come Graviton4, rapporti memoria-CPU configurabili e reti a larghezza di banda elevata.
- Provisioning automatico: esegue automaticamente il provisioning delle istanze idonee e attiva gli ambienti di esecuzione delle funzioni. AWS
- Scalabilità dinamica: le istanze vengono scalate dinamicamente in base ai modelli di traffico delle funzioni.
- Esperienza completamente gestita: AWS gestisce la gestione, il ridimensionamento, l'applicazione di patch e il routing dell'infrastruttura, con le stesse integrazioni complete di sorgenti di eventi che conosci.

## Quando usare le istanze gestite Lambda

Prendi in considerazione le istanze gestite Lambda per i seguenti casi d'uso:

- Carichi di lavoro prevedibili ad alto volume: ideali per carichi di lavoro stazionari senza picchi di traffico imprevisti. Per impostazione predefinita, le istanze gestite Lambda sono scalabili per gestire il raddoppio del traffico entro cinque minuti.
- Performance-critical applicazioni: accesso alle CPU più recenti, rapporti memoria-CPU variabili e throughput di rete elevato
- Requisiti normativi: esigenze di governance granulari con controllo sul VPC e sul posizionamento delle istanze
- Varietà di applicazioni: Event-driven applicazioni, media/data elaborazione, applicazioni web e carichi di lavoro legacy che migrano verso il serverless



## Come funziona

Lambda Managed Instances utilizza i fornitori di capacità come base per l'esecuzione delle funzioni:

1. Crea un fornitore di capacità: definisci dove vengono eseguite le tue funzioni specificando la configurazione del VPC e, facoltativamente, i requisiti delle istanze e la configurazione di scalabilità
2. Crea la tua funzione: crea le funzioni Lambda come al solito e collegale a un fornitore di capacità
3. Pubblica una versione della funzione: una volta pubblicate, le versioni delle funzioni diventano attive sulle istanze del fornitore di capacità

Quando pubblichi una versione della funzione con un fornitore di capacità, Lambda avvia le istanze gestite nel tuo account. Avvia tre istanze per impostazione predefinita per la resilienza AZ e avvia tre ambienti di esecuzione prima di contrassegnare la versione della funzione come ACTIVE. Se colleghi una funzione a un fornitore di capacità esistente che sta già eseguendo altre funzioni, Lambda potrebbe non generare nuove istanze se le istanze disponibili hanno già la capacità di adattarsi agli ambienti di esecuzione della nuova funzione.

## Modello di concorrenza

Le istanze gestite Lambda supportano chiamate multiple simultanee, in cui un ambiente di esecuzione può gestire più invocazioni contemporaneamente. Questo è diverso dal tipo di calcolo Lambda (predefinito), che fornisce un modello di concorrenza unico in cui un ambiente di esecuzione può eseguire un massimo di un'invocazione alla volta. Multi-concurrency consente un migliore utilizzo delle istanze EC2 sottostanti ed è particolarmente utile per applicazioni come servizi Web o lavori batch. IO-heavy Questa modifica nel modello di esecuzione significa che la sicurezza dei thread, la gestione dello stato e l'isolamento del contesto devono essere gestiti in modo diverso a seconda del runtime.

## Locazione e isolamento

Il tipo di elaborazione Lambda (predefinito) è multi-tenant e utilizza la tecnologia Firecracker MicroVM per fornire isolamento tra gli ambienti di esecuzione in esecuzione su flotte Lambda condivise. Le istanze gestite Lambda vengono eseguite nel tuo account e forniscono l'hardware e le opzioni di prezzo di EC2 più recenti. Le istanze gestite utilizzano contenitori in esecuzione su istanze EC2 Nitro per fornire isolamento anziché Firecracker. I fornitori di capacità fungono da limite di sicurezza per le funzioni Lambda. Le funzioni vengono eseguite in contenitori all'interno di istanze.

## Comprensione delle istanze gestite

Le funzioni Lambda Managed Instances vengono eseguite sulle [istanze gestite](#) EC2 presenti nel tuo account. Queste istanze sono completamente gestite da Lambda, il che significa che hai autorizzazioni limitate rispetto alle istanze EC2 standard. Puoi identificare le istanze gestite Lambda nel tuo account tramite:

- La presenza del `Operator` campo nell'output `EC2 DescribeInstances`
- Il `aws:lambda:capacity-provider` tag sull'istanza

Non è possibile eseguire operazioni EC2 standard direttamente su queste istanze, ad esempio terminarle manualmente. Per distruggere le istanze gestite, elimina il fornitore di capacità associato. Lambda termina quindi le istanze come parte del processo di eliminazione del fornitore di capacità.

Per impostazione predefinita, le istanze gestite sono nascoste dalle visualizzazioni della console EC2 e dalle operazioni dell'elenco delle API. È possibile regolare la visibilità utilizzando l'impostazione di visibilità delle risorse [gestite](#). Le istanze gestite rimangono completamente operative e fatturabili nel tuo AWS account.

## Prezzi

Le istanze gestite Lambda utilizzano un EC2-based prezzo con una commissione di gestione del 15% in aggiunta al costo delle istanze EC2. Questo modello di prezzo supporta i piani di risparmio EC2, le istanze riservate e qualsiasi altro sconto applicato all'utilizzo di EC2. Per ulteriori dettagli, consulta la pagina dei prezzi: <https://aws.amazon.com/lambda/pricing/>

Per confrontare i costi di esecuzione delle tue funzioni su Lambda default, Lambda Managed Instances e Amazon EC2 autogestito, utilizza il calcolatore dei prezzi di [Lambda Managed Instances](#).

Importante: gli sconti sui prezzi di EC2 si applicano solo al calcolo EC2 sottostante, non alla commissione di gestione.

## In che modo Lambda Managed Instances differisce dal tipo di elaborazione Lambda (predefinito)

Le istanze gestite da Lambda cambiano il modo in cui Lambda elabora le richieste rispetto a Lambda (impostazione predefinita).

Differenze principali:

|                                   | Lambda (impostazione predefinita)                                                                                                                               | Istanze Lambda gestite                                                                                                                                                                      |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Modello di concorrenza            | Modello di concorrenza unico in cui un ambiente di esecuzione può supportare un massimo di una chiamata alla volta                                              | Multi-concurrent invocazioni in cui un ambiente di esecuzione è in grado di gestire più chiamate contemporaneamente, aumentando la produttività soprattutto per le applicazioni IO-heavy    |
| Locazione e isolamento            | Multi-tenant, utilizzando la tecnologia Firecracker MicroVM per fornire l'isolamento tra gli ambienti di esecuzione in esecuzione su flotte Lambda condivise    | Esegui nel tuo account, utilizzando EC2 Nitro per fornire l'isolamento. I fornitori di capacità fungono da limite di sicurezza, con funzioni eseguite in contenitori all'interno di istanze |
| Modello tariffario                | Per-request prezzi relativi alla durata                                                                                                                         | Instance-based prezzi con modelli di prezzo EC2, incluse On-Demand le istanze riservate, e opzioni di risparmio come Compute Savings Plans                                                  |
| Comportamento del dimensionamento | Scalabile quando non esiste un ambiente di esecuzione libero per gestire una chiamata in entrata (avvio a freddo). Scala a zero senza traffico                  | Scalabilità asincrona in base al solo utilizzo delle risorse della CPU, senza avviamenti a freddo. Si adatta agli ambienti di esecuzione minimi configurati senza traffico                  |
| Ideale per                        | Funziona con traffico intenso in grado di gestire tempi di avvio a freddo o applicazioni senza carico sostenuto che traggono vantaggio dalla scalabilità a zero | Il traffico prevedibile ad alto volume funziona quando si desidera la flessibilità, i piani tariffari e le opzioni hardware di EC2                                                          |

## Fasi successive

- Scopri i concetti [fondamentali per le istanze gestite Lambda](#)
- [Inizia con Lambda Managed Instances](#)
- Scopri i fornitori di [capacità per le istanze gestite Lambda](#)
- Scopri la [scalabilità per le istanze gestite Lambda](#)
- Rivedi i runtime supportati [???](#)
- Consulta le guide specifiche per i runtime per [Java e Python](#) [Node.js](#) [???](#)
- Configura la connettività [VPC per i tuoi fornitori di capacità](#)
- Comprendi [la sicurezza e le autorizzazioni per le istanze gestite Lambda](#)
- Rivedi [le quote e i limiti](#)
- [Risolvi i problemi relativi alle istanze gestite Lambda](#)

## Guida introduttiva alle istanze gestite Lambda

### Creazione di una funzione Lambda Managed Instance (console)

Puoi utilizzare la console Lambda per creare una funzione Managed Instance che viene eseguita su istanze Amazon EC2 gestite da un fornitore di capacità.

Importante: prima di creare una funzione di istanza gestita, devi prima creare un fornitore di capacità. Queste funzioni richiedono che un fornitore di capacità definisca l'infrastruttura Amazon EC2 che esegue le tue funzioni.

### Per creare una funzione Lambda Managed Instance (console)

1. Apri la console Lambda.
2. Scegli i fornitori di capacità dal riquadro di navigazione a sinistra.
3. Scegli Crea fornitore di capacità.
4. Nella sezione delle impostazioni del fornitore di capacità, inserisci un nome per il tuo fornitore di capacità.
5. Seleziona VPC e le autorizzazioni per il tuo fornitore di capacità. Puoi utilizzarne uno esistente o crearne uno nuovo. Per informazioni sulla creazione del ruolo operatore richiesto, consulta Ruolo operatore [Lambda per le istanze gestite Lambda](#).
6. Espandere Advanced settings (Impostazioni avanzate).

7. Definisci i requisiti dell'istanza scegliendo l'architettura del processore e i tipi di istanza.
8. In Auto scaling, specifica il numero massimo di vCPU EC2 per il tuo fornitore di capacità. Puoi anche scegliere la modalità di ridimensionamento manuale delle istanze per impostare il tuo valore di scalabilità per un controllo preciso.
9. Scegli Crea fornitore di capacità per crearne uno nuovo.
10. Quindi, scegli Crea funzione.
11. Scegli Crea da zero.
12. Nel riquadro Informazioni di base, fornisci un nome di funzione.
13. Per Runtime, scegli uno dei Runtime supportati.
14. Scegli l'architettura per la tua funzione (uguale a quella selezionata per il fornitore di capacità). Per impostazione predefinita, x86\_64.
15. In Autorizzazioni, assicurati di disporre dell'autorizzazione per il ruolo di esecuzione scelto. Altrimenti, puoi creare un nuovo ruolo.
16. In Configurazioni aggiuntive, seleziona il tipo di calcolo come Lambda Managed Instances.
17. L'ARN del fornitore di capacità creato nei passaggi precedenti deve essere preselezionato.
18. Scegliete la dimensione della memoria e il rapporto tra memoria dell'ambiente di esecuzione (GiB) per vCPU.
19. Scegli Crea funzione.

La funzione Lambda Managed Instance viene creata e fornisce capacità al provider di capacità specificato. La creazione della funzione richiede in genere diversi minuti. Una volta completata, puoi modificare il codice della funzione ed eseguire il primo test.

## Creazione di una funzione Lambda Managed Instance (AWS CLI)

### Prerequisiti

Prima di iniziare, assicurati di disporre di:

- AWS CLI: installa e configura la CLI AWS . Per ulteriori informazioni, vedere [Installazione o aggiornamento della versione più recente della CLI AWS](#) .
- Autorizzazioni IAM: l'utente o il ruolo IAM deve disporre delle autorizzazioni per creare funzioni Lambda, fornitori di capacità e passare ruoli IAM. Tieni presente che ti serviranno anche `iam:CreateServiceLinkedRole` se è la prima volta che crei un fornitore di capacità nell'account o se il Service Linked Role (SLR) è stato eliminato.

## Fase 1: Creare i ruoli IAM richiesti

Le istanze gestite Lambda richiedono due ruoli IAM: un ruolo di esecuzione per la funzione e un ruolo di operatore per il fornitore di capacità. Il ruolo di operatore consente a Lambda di avviare, terminare e monitorare le istanze Amazon EC2 per tuo conto. Il ruolo di esecuzione della funzione concede alla funzione le autorizzazioni per accedere ad altri servizi e risorse. AWS

Per creare il ruolo di esecuzione Lambda

1. Crea un documento sulla politica di fiducia che consenta a Lambda di assumere il ruolo:

```
cat > lambda-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

2. Crea il ruolo di esecuzione:

```
aws iam create-role \
  --role-name MyLambdaExecutionRole \
  --assume-role-policy-document file://lambda-trust-policy.json
```

3. Allega la politica di esecuzione di base:

```
aws iam attach-role-policy \
  --role-name MyLambdaExecutionRole \
  --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

Per creare il ruolo di operatore del fornitore di capacità

1. Crea un documento sulla politica di fiducia che consenta a Lambda di assumere il ruolo di operatore:

```
cat > operator-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

2. Crea il ruolo di operatore:

```
aws iam create-role \
  --role-name MyCapacityProviderOperatorRole \
  --assume-role-policy-document file://operator-trust-policy.json
```

3. Allega la politica di autorizzazione EC2 richiesta:

```
aws iam attach-role-policy \
  --role-name MyCapacityProviderOperatorRole \
  --policy-arn arn:aws:iam::aws:policy/AWSLambdaManagedEC2ResourceOperator
```

## Fase 2: Configurare le risorse VPC

Le istanze gestite Lambda vengono eseguite nel tuo VPC e richiedono una sottorete e un gruppo di sicurezza.

Per creare risorse VPC

1. Crea un VPC:

```
VPC_ID=$(aws ec2 create-vpc \
  --cidr-block 10.0.0.0/16 \
```

```
--query 'Vpc.VpcId' \  
--output text)
```

## 2. Crea una sottorete:

```
SUBNET_ID=$(aws ec2 create-subnet \  
--vpc-id $VPC_ID \  
--cidr-block 10.0.1.0/24 \  
--query 'Subnet.SubnetId' \  
--output text)
```

## 3. Crea un gruppo di sicurezza:

```
SECURITY_GROUP_ID=$(aws ec2 create-security-group \  
--group-name my-capacity-provider-sg \  
--description "Security group for Lambda Managed Instances" \  
--vpc-id $VPC_ID \  
--query 'GroupId' \  
--output text)
```

Nota: le funzioni delle tue istanze gestite Lambda richiedono la configurazione del VPC per accedere alle risorse esterne al VPC e per trasmettere i dati di telemetria a Logs e CloudWatch X-Ray. Per i dettagli sulla configurazione, vedi [Networking for Lambda Managed Instances](#). ???

## Fase 3: creazione di un provider di capacità

Un fornitore di capacità gestisce le istanze EC2 che eseguono le funzioni Lambda.

## Come creare un provider di capacità

```
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)  
  
aws lambda create-capacity-provider \  
--capacity-provider-name my-capacity-provider \  
--vpc-config SubnetIds=[$SUBNET_ID],SecurityGroupIds=[$SECURITY_GROUP_ID] \  
--permissions-config CapacityProviderOperatorRoleArn=arn:aws:iam::${ACCOUNT_ID}:role/  
MyCapacityProviderOperatorRole \  
--instance-requirements Architectures=[x86_64] \  
--capacity-provider-scaling-config MaxVCpuCount=30
```

Questo comando crea un fornitore di capacità con la seguente configurazione:



- Configurazione VPC: specifica la sottorete e il gruppo di sicurezza per le istanze EC2
- Autorizzazioni: definisce il ruolo IAM utilizzato da Lambda per gestire le istanze EC2
- Requisiti di istanza: specifica l'architettura x86\_64
- Configurazione di scalabilità: imposta un massimo di 30 vCPU per il fornitore di capacità

## Fase 4: Creare una funzione Lambda con codice in linea

Per creare una funzione con codice in linea

1. Innanzitutto, crea una semplice funzione Python e impacchettala in linea:

```
# Create a temporary directory for the function code
mkdir -p /tmp/my-lambda-function
cd /tmp/my-lambda-function

# Create a simple Python handler
cat > lambda_function.py << 'EOF'
import json

def lambda_handler(event, context):
    return {
        'statusCode': 200,
        'body': json.dumps({
            'message': 'Hello from Lambda Managed Instances!',
            'event': event
        })
    }
EOF

# Create a ZIP file
zip function.zip lambda_function.py
```

2. Crea la funzione Lambda usando il file ZIP in linea:

```
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
REGION=$(aws configure get region)

aws lambda create-function \
  --function-name my-managed-instance-function \
  --package-type Zip \
  --runtime python3.13 \
```

```
--handler lambda_function.lambda_handler \  
--zip-file fileb:///tmp/my-lambda-function/function.zip \  
--role arn:aws:iam::${ACCOUNT_ID}:role/MyLambdaExecutionRole \  
--architectures x86_64 \  
--memory-size 2048 \  
--ephemeral-storage Size=512 \  
--capacity-provider-config  
LambdaManagedInstancesCapacityProviderConfig={CapacityProviderArn=arn:aws:lambda:  
${REGION}:${ACCOUNT_ID}:capacity-provider:my-capacity-provider}
```

La funzione viene creata con:

- Runtime — Python 3.13
- Handler — La funzione in `lambda_handler lambda_function.py`
- Memoria: 2048 MB
- Memoria temporanea: 512 MB
- Fornitore di capacità: collegamenti al fornitore di capacità creato

## Fase 5: Pubblicare una versione della funzione

Per eseguire la funzione su Lambda Managed Instances, devi pubblicare una versione.

Per pubblicare una versione della funzione

```
aws lambda publish-version \  
--function-name my-managed-instance-function
```

Questo comando pubblica la versione 1 della funzione e la distribuisce al fornitore di capacità.

## Passaggio 6: richiama la tua funzione

Dopo la pubblicazione, puoi richiamare la tua funzione.

Per richiamare la tua funzione

```
aws lambda invoke \  
--function-name my-managed-instance-function:1 \  
--payload '{"name": "World"}' \  
response.json  
  
# View the response
```

```
cat response.json
```

La funzione viene eseguita sulle istanze EC2 gestite dal fornitore di capacità e restituisce una risposta.

## Eliminazione

Per evitare di incorrere in addebiti, elimina le risorse che hai creato:

### 1. Elimina la funzione:

```
aws lambda delete-function --function-name my-managed-instance-function
```

### 2. Eliminare il fornitore di capacità:

```
aws lambda delete-capacity-provider --capacity-provider-name my-capacity-provider
```

### 3. Elimina le risorse VPC:

```
aws ec2 delete-security-group --group-id $SECURITY_GROUP_ID
aws ec2 delete-subnet --subnet-id $SUBNET_ID
aws ec2 delete-vpc --vpc-id $VPC_ID
```

### 4. Elimina i ruoli IAM:

```
aws iam detach-role-policy \
  --role-name MyLambdaExecutionRole \
  --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
aws iam detach-role-policy \
  --role-name MyCapacityProviderOperatorRole \
  --policy-arn arn:aws:iam::aws:policy/AWSLambdaManagedEC2ResourceOperator

aws iam delete-role --role-name MyLambdaExecutionRole
aws iam delete-role --role-name MyCapacityProviderOperatorRole
```

## Concetti principali

Lambda Managed Instances introduce diversi concetti fondamentali che differiscono dalle funzioni Lambda tradizionali. La comprensione di questi concetti è essenziale per implementare e gestire in modo efficace le funzioni sull'infrastruttura EC2.

I fornitori di capacità costituiscono la base delle istanze gestite Lambda. Un fornitore di capacità definisce l'infrastruttura di calcolo in cui vengono eseguite le funzioni, inclusa la configurazione VPC, i requisiti delle istanze e le politiche di scalabilità. I fornitori di capacità fungono anche da limite di sicurezza per le funzioni, il che significa che tutte le funzioni assegnate allo stesso fornitore di capacità devono essere affidabili reciprocamente.

Il comportamento di scalabilità è notevolmente diverso dalle funzioni Lambda tradizionali. Invece di scalare su richiesta quando arrivano le chiamate, le istanze gestite scalano in modo asincrono in base all'utilizzo delle risorse della CPU. Questo approccio elimina le partenze a freddo ma richiede una pianificazione della crescita del traffico. Se il traffico raddoppia più che in 5 minuti, è possibile che si verifichino rallentamenti poiché Lambda aumenta la capacità per soddisfare la domanda.

La sicurezza e le autorizzazioni richiedono un'attenta valutazione. Sono necessarie le autorizzazioni relative ai ruoli di operatore per consentire a Lambda di gestire le risorse EC2 nei fornitori di capacità. Inoltre, gli utenti devono disporre dell'`lambda:PassCapacityProviderAuthorization` per assegnare funzioni ai fornitori di capacità, fungendo da gateway di sicurezza per controllare quali funzioni possono essere eseguite su un'infrastruttura specifica.

Multi-concurrent l'esecuzione è una caratteristica chiave delle istanze gestite. Ogni ambiente di esecuzione è in grado di gestire più invocazioni contemporaneamente, massimizzando l'utilizzo delle risorse per le applicazioni. IO-heavy Questo è diverso dal Lambda tradizionale, in cui ogni ambiente elabora una richiesta alla volta. Questo modello di esecuzione richiede attenzione alla sicurezza dei thread, alla gestione dello stato e all'isolamento del contesto a seconda del runtime.

Per informazioni dettagliate sulla pubblicazione delle versioni delle funzioni sul tuo capacity provider, consulta [the section called “Pubblicazione della versione”](#).

Le sezioni seguenti forniscono informazioni dettagliate su ciascun concetto fondamentale.

## fornitori di capacità

Un fornitore di capacità è la base per l'esecuzione di Lambda Managed Instances. Funge da limite di sicurezza per le tue funzioni e definisce le risorse di calcolo che Lambda fornisce e gestisce per tuo conto.

Quando crei un fornitore di capacità, specifichi:

- Configurazione VPC: le sottoreti e i gruppi di sicurezza in cui vengono eseguite le istanze
- Autorizzazioni: ruoli IAM per Lambda per la gestione delle risorse EC2
- Requisiti di istanza (opzionale) - Preferenze relative all'architettura e [al tipo di istanza](#)

- Configurazione del ridimensionamento (opzionale): in che modo Lambda scala le tue istanze

## Comprendere i fornitori di capacità come limite di sicurezza

I fornitori di capacità fungono da limite di sicurezza per le funzioni Lambda all'interno del VPC, sostituendo l'isolamento. Firecracker-based Le funzioni vengono eseguite in contenitori all'interno delle istanze, ma i contenitori non forniscono un forte isolamento di sicurezza tra le funzioni, a differenza di Firecracker MicroVMS.

### Concetti chiave di sicurezza:

- Capacity Provider: il limite di sicurezza che definisce i livelli di affidabilità per le funzioni Lambda
- Isolamento dei container: i container NON sono un fornitore di sicurezza: non affidatevi a loro per la sicurezza tra carichi di lavoro non attendibili
- Separazione della fiducia: separa i carichi di lavoro che non sono reciprocamente affidabili utilizzando fornitori di capacità diversi

## Creazione di un provider di capacità

È possibile creare un fornitore di capacità utilizzando la AWS CLI, la console di AWS gestione o gli SDK. AWS

### Utilizzo AWS della CLI:

```
aws lambda create-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --vpc-config
SubnetIds=subnet-12345,subnet-67890,subnet-11111,SecurityGroupIds=sg-12345 \
  --permissions-config CapacityProviderOperatorRoleArn=arn:aws:iam::123456789012:role/
MyOperatorRole \
  --instance-requirements Architectures=x86_64 \
  --capacity-provider-scaling-config ScalingMode=Auto
```

## Parametri obbligatori

### CapacityProviderName

- Un nome univoco per il tuo fornitore di capacità
- Deve essere univoco all'interno del tuo AWS account

## VpcConfig

- **SubnetIds**(obbligatorio): almeno una sottorete, massimo 16. Usa le sottoreti in più zone di disponibilità per la resilienza
- **SecurityGroupIds**(opzionale): gruppi di sicurezza per le tue istanze. Il valore predefinito è il gruppo di sicurezza predefinito di VPC se non specificato

## PermissionsConfig

- **CapacityProviderOperatorRoleArn**(obbligatorio): ruolo IAM che consente a Lambda di gestire le risorse EC2 nel tuo capacity provider

## Parametri facoltativi

### InstanceRequirements

Specifica l'architettura e i tipi di [istanza](#) per il tuo fornitore di capacità:

- **Architetture**: scegli `x86_64` o `arm64`. L'impostazione predefinita è `x86_64`
- **AllowedInstanceTypes**: Specifica i tipi di istanza consentiti. Ad esempio: `m5.8xlarge`
- **ExcludedInstanceTypes**: specifica i tipi di istanza esclusi utilizzando caratteri jolly. È possibile specificare solo uno tra `AllowedInstanceTypes` e `ExcludedInstanceTypes`

Per impostazione predefinita, Lambda sceglie i tipi di istanza ottimali per il carico di lavoro. Ti consigliamo di lasciare che Lambda Managed Instances scelga i tipi di istanza per te, poiché limitare il numero di possibili tipi di istanza potrebbe comportare una minore disponibilità.

## CapacityProviderScalingConfig

Configura il modo in cui Lambda ridimensiona le tue istanze:

- **ScalingMode**: impostato su `Auto` per il ridimensionamento automatico o `Manual` per il controllo manuale. L'impostazione predefinita è `Auto`
- **MaxVCpuCount**: numero massimo di vCPU per il fornitore di capacità. L'impostazione predefinita è 400.
- **ScalingPolicies**: definisce le politiche di scalabilità del tracciamento degli obiettivi per l'utilizzo della CPU e della memoria

## KmsKeyArn

Specifica una AWS KMS chiave per la crittografia EBS. Il valore predefinito è la chiave AWS gestita se non specificata.

## Tag

Aggiungi tag per organizzare e gestire i tuoi fornitori di capacità.

## PropagateTags

Configura la propagazione dei tag per applicare automaticamente i tag alle risorse gestite (istanze Amazon EC2, volumi Amazon EBS ed ENI) lanciate dal fornitore di capacità. Usa la propagazione dei tag per applicare l'allocazione dei costi, gli SCP e i requisiti di conformità. Per ulteriori informazioni, consulta [Propagazione dei tag](#). ???

## Gestione dei fornitori di capacità

### Aggiornare un provider di capacità

È possibile aggiornare determinate proprietà di un fornitore di capacità utilizzando l'`UpdateCapacityProviderAPI`.

```
aws lambda update-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --capacity-provider-scaling-config ScalingMode=Auto
```

### Eliminare un provider di capacità

Puoi eliminare un fornitore di capacità quando non è più necessario utilizzando l'`DeleteCapacityProviderAPI`.

```
aws lambda delete-capacity-provider \
  --capacity-provider-name my-capacity-provider
```

Nota: non è possibile eliminare un fornitore di capacità a cui sono associate versioni di funzioni.

### Visualizzazione dei dettagli del fornitore di capacità

Recupera informazioni su un fornitore di capacità utilizzando l'`GetCapacityProviderAPI`.

```
aws lambda get-capacity-provider \  
  --capacity-provider-name my-capacity-provider
```

## Stati del fornitore di capacità

Un fornitore di capacità può trovarsi in uno dei seguenti stati:

- In sospeso: il fornitore di capacità è in fase di creazione
- Attivo: il fornitore di capacità è pronto per l'uso
- Fallito: la creazione del fornitore di capacità non è riuscita
- Eliminazione: il fornitore di capacità è in fase di eliminazione

## Quote

- Capacità massima dei fornitori per account: 1.000
- Numero massimo di versioni di funzioni per fornitore di capacità: 100 (non può essere aumentato)

## Best practice

1. Separate per livello di affidabilità: create diversi fornitori di capacità per carichi di lavoro con requisiti di sicurezza diversi
2. Usa nomi descrittivi: nomina i fornitori di capacità per indicare chiaramente l'uso previsto e il livello di affidabilità (ad esempio, `production-trusted`). `dev-sandbox`
3. Utilizza più zone di disponibilità: specifica le sottoreti su più AZ per un'elevata disponibilità
4. Lascia che Lambda scelga i tipi di istanza: a meno che tu non abbia requisiti hardware specifici, consenti a Lambda di selezionare i tipi di istanza ottimali per la disponibilità
5. Monitora l'utilizzo: AWS CloudTrail da utilizzare per monitorare le assegnazioni dei fornitori di capacità e i modelli di accesso

## Fasi successive

- Scopri come [scalare le istanze gestite Lambda](#)
- Scopri la [sicurezza e le autorizzazioni per le istanze gestite Lambda](#)
- Configura la connettività [VPC per i tuoi fornitori di capacità](#)



- Consulta le guide specifiche per il runtime per [Java e Python](#) [Node.js](#) [???](#)

## Propagazione di tag

Con la propagazione dei tag, puoi specificare un set di tag nella configurazione del tuo provider di capacità e Lambda applica automaticamente tali tag a tutte le risorse gestite che crea, incluse le istanze Amazon EC2, i volumi Amazon EBS e ENI. Ciò garantisce un'etichettatura coerente per l'allocazione dei costi, le politiche di controllo del servizio (SCP) e i requisiti di conformità senza richiedere interventi manuali o automazione personalizzata.

### Configurazione della propagazione dei tag

Specificate l'`PropagateTags` impostazione durante la creazione o l'aggiornamento di un provider di capacità utilizzando le API `CreateCapacityProvider` o `UpdateCapacityProvider`.

`PropagateTags` parametri:

- Modalità (obbligatoria): la modalità di propagazione dei tag:
  - `Explicit`— Propaga i tag specificati in `ExplicitTags` alle risorse gestite.
  - `None`— Disabilita la propagazione dei tag. Nessun tag personalizzato viene applicato alle risorse gestite ad eccezione dei tag di sistema applicati per impostazione predefinita.
- `ExplicitTags`(richiesto quando `Mode` è `Explicit`): una mappa di coppie chiave-valore da applicare alle risorse gestite. È possibile specificare fino a 40 tag.

#### Important

La propagazione dei tag si applica solo alle nuove risorse gestite fornite dopo l'applicazione della configurazione. Le risorse esistenti non vengono etichettate retroattivamente.

## Esempi

Crea un provider di capacità con propagazione dei tag (AWS CLI):

```
aws lambda create-capacity-provider \  
  --capacity-provider-name my-capacity-provider \  
  --vpc-config SubnetIds=subnet-12345,subnet-67890,SecurityGroupIds=sg-12345 \  
  --propagate-tags Explicit \  
  --explicit-tags Key=Value
```

```
--permissions-config CapacityProviderOperatorRoleArn=arn:aws:iam::123456789012:role/MyOperatorRole \  
--propagate-tags '{"Mode": "Explicit", "ExplicitTags": {"CostCenter": "12345",  
"Environment": "Production"}}'
```

Aggiorna un provider di capacità esistente per abilitare la propagazione dei tag:

```
aws lambda update-capacity-provider \  
--capacity-provider-name my-capacity-provider \  
--propagate-tags '{"Mode": "Explicit", "ExplicitTags": {"CostCenter": "12345",  
"Environment": "Production"}}'
```

Disabilita la propagazione dei tag:

```
aws lambda update-capacity-provider \  
--capacity-provider-name my-capacity-provider \  
--propagate-tags '{"Mode": "None"}'
```

### Comportamento di propagazione dei tag

- Le modifiche alla `PropagateTags` configurazione influiscono solo sulle risorse fornite dopo l'aggiornamento. Le risorse avviate in precedenza mantengono i tag originali.
- I tag di propagazione dei tag sono separati dal `Tags` parametro del provider di capacità stesso. I tag del fornitore di capacità identificano la risorsa del fornitore di capacità; i tag propagati vengono applicati alle risorse gestite avviate dal provider di capacità.
- I tag propagati vengono conteggiati ai fini dei limiti dei tag di AWS risorsa sulle risorse di destinazione.

## Scalabilità delle istanze gestite Lambda

Lambda Managed Instances non è scalabile quando arrivano le chiamate e non supporta gli avviamenti a freddo. Invece, scala in modo asincrono utilizzando segnali di consumo di risorse. Le istanze gestite attualmente sono scalabili in base all'utilizzo delle risorse della CPU e alla saturazione di più concorrenti.

Differenze principali:

- Lambda (impostazione predefinita): scala quando non esiste un ambiente di esecuzione libero per gestire una chiamata in arrivo (avvio a freddo)

- Istanze gestite Lambda: scalabilità asincrona in base all'utilizzo delle risorse della CPU e alla saturazione multi-concorrenza degli ambienti di esecuzione

Se il traffico raddoppia più che in 5 minuti, potresti riscontrare dei rallentamenti man mano che Lambda aumenta le istanze e gli ambienti di esecuzione per soddisfare la domanda.

Il ciclo di vita della scalabilità

Lambda Managed Instances utilizza un'architettura distribuita per gestire la scalabilità:

Componenti:

- Istanze gestite: eseguite nel tuo account nelle sottoreti fornite
- Router e scaler: componenti Lambda condivisi che instradano le chiamate e gestiscono il ridimensionamento
- Lambda Agent: viene eseguito su ogni istanza gestita per gestire il ciclo di vita dell'ambiente di esecuzione e monitorare il consumo di risorse

Come funziona:

1. Quando pubblichi una versione della funzione con un fornitore di capacità, Lambda avvia le istanze gestite nel tuo account. Ne avvia tre per impostazione predefinita per la resilienza AZ e avvia tre ambienti di esecuzione prima di contrassegnare la versione della funzione come ATTIVA.
2. Ogni istanza gestita può eseguire ambienti di esecuzione per più funzioni mappate allo stesso fornitore di capacità.
3. Man mano che il traffico entra nell'applicazione, gli ambienti di esecuzione consumano risorse. L'agente Lambda invia una notifica allo Scaler, che decide se scalare nuovi ambienti di esecuzione o istanze gestite.
4. Se il router tenta di inviare una chiamata a un ambiente di esecuzione con un elevato consumo di risorse, il Lambda Agent su quell'istanza lo avvisa di riprovare su un'altra.
5. Man mano che il traffico diminuisce, l'agente Lambda invia una notifica a Scaler, che decide di ridimensionare gli ambienti di esecuzione e scalare le istanze gestite.

Regolazione del comportamento di ridimensionamento

È possibile personalizzare il comportamento di scalabilità delle istanze gestite tramite cinque controlli:

## Controlli a livello di funzione

### 1. Memoria funzionale e vCPU

Scegliete la dimensione della memoria e l'allocazione della vCPU per la vostra funzione. La dimensione minima della funzione supportata è 2 GB e 1 vCPU.

Considerazioni:

- Scegliete un'impostazione di memoria e vCPU che supporti le esecuzioni simultanee della vostra funzione
- Non è possibile configurare una funzione con meno di 1 vCPU perché le funzioni in esecuzione su istanze gestite dovrebbero supportare carichi di lavoro simultanei multipli
- Non è possibile scegliere meno di 2 GB perché corrisponde al rapporto di 2 a 1 tra memoria e vCPU delle istanze c, che hanno il rapporto più basso
- Per le applicazioni Python, potrebbe essere necessario scegliere un rapporto più elevato tra memoria e vCPU, ad esempio 4 a 1 o 8 a 1, a causa del modo in cui Python gestisce la multi-concorrenza
- Se stai eseguendo CPU-intensive operazioni o esegui pochi I/O, dovresti scegliere più di una vCPU

### 2. Simultaneità massima

Imposta la concorrenza massima per ambiente di esecuzione.

Comportamento predefinito: Lambda sceglie impostazioni predefinite ragionevoli che bilanciano il consumo di risorse e la velocità effettiva che funzionano per un'ampia varietà di applicazioni.

Linee guida per la regolazione:

- Aumentare la concorrenza: se le chiamate di funzione utilizzano pochissima CPU, è possibile aumentare la concorrenza massima fino a un massimo di 64 per vCPU
- Riduzione della concorrenza: se l'applicazione consuma una grande quantità di memoria e pochissima CPU, è possibile ridurre la concorrenza massima

Importante: poiché le istanze gestite Lambda sono pensate per applicazioni con più concorrenti, gli ambienti di esecuzione con una concorrenza molto bassa potrebbero subire limitazioni durante la scalabilità. Quando le chiamate arrivano in un ambiente di esecuzione che ha raggiunto il limite

di concorrenza, Lambda le indirizza altrove e ridimensiona i nuovi ambienti di esecuzione per gestire il carico. Per identificare quale vincolo di risorse sta causando le limitazioni, monitora le metriche del motivo dell'accelerazione (,, e) descritte in. [ConcurrencyThrottles](#) [CPULimits](#) [MemoryThrottles](#) [DiskThrottles](#) [Tipi di parametri per le funzioni Lambda](#)

### 3. Ambienti di esecuzione per funzione

Imposta il numero minimo e massimo di ambienti di esecuzione per la tua funzione.

Comportamento predefinito: il minimo predefinito è di 3 ambienti di esecuzione nelle zone di disponibilità, senza un massimo predefinito. È possibile sovrascrivere entrambi i valori dopo la creazione della funzione.

Linee guida per la regolazione:

- Stabilisci il valore minimo: fornisci capacità per il traffico di base e riduci le accelerazioni in caso di raffiche improvvise. I valori inferiori a 3 riducono la ridondanza nella zona di disponibilità.
- Imposta il valore massimo: limita il numero di ambienti di esecuzione per controllare lo scale-out e prevenire i rumorosi problemi adiacenti quando più funzioni condividono un Capacity Provider.
- Disattiva la funzione: imposta il valore minimo e massimo su 0 per disattivare una funzione senza eliminarla.

Esempio:

```
aws lambda put-function-scaling-config \  
  --function-name my-lmi-function \  
  --qualifier '$LATEST.PUBLISHED' \  
  --function-scaling-config MinExecutionEnvironments=5,MaxExecutionEnvironments=20 \  
  --region us-east-1
```

Note importanti:

- Ambito del qualificatore: queste configurazioni si applicano a livello di funzione per ogni ARN qualificato. Quando è attivata `$LATEST.PUBLISHED`, la configurazione si propaga alle versioni future. `$LATEST.PUBLISHED` Se impostate su una versione specifica, le nuove versioni pubblicate tornano ai valori predefiniti.
- Configurazione abbinata: è necessario impostare contemporaneamente i valori minimo e massimo. Qualsiasi impostazione non specificata viene ripristinata al valore predefinito. Valori validi

per entrambi `MinExecutionEnvironments` e `MaxExecutionEnvironments` vanno da 0 a 15000. Un valore minimo di 0 è valido solo quando anche il valore massimo è 0.

- Implicazione in termini di costi: la disattivazione della funzione ha effetto a livello di versione della funzione. Lambda termina un'istanza EC2 sottostante quando non dispone di ambienti di esecuzione attivi e i costi dell'istanza continuano fino al completamento della terminazione (in genere entro pochi minuti).

Controlli a livello di fornitore di capacità

#### 4. Utilizzo delle risorse di destinazione

Scegli il tuo obiettivo per il consumo di utilizzo della CPU.

Comportamento predefinito: Lambda mantiene un margine di manovra sufficiente per raddoppiare il traffico in 5 minuti senza rallentamenti.

Opzioni di ottimizzazione:

- Se il carico di lavoro è molto costante o se l'applicazione non è sensibile alle limitazioni, è possibile impostare l'obiettivo su un livello elevato per ottenere un utilizzo maggiore e costi inferiori
- Se si desidera mantenere un margine di manovra per le ondate di traffico, è possibile impostare obiettivi di risorse a un livello basso, il che richiede una maggiore capacità

#### 5. Selezione del tipo di istanza

Imposta i tipi di istanza consentiti o esclusi.

Comportamento predefinito: Lambda sceglie i tipi di istanza migliori per il carico di lavoro. Si consiglia di consentire a Lambda Managed Instances di scegliere i tipi di istanza, poiché limitare il numero di possibili tipi di istanza potrebbe comportare una minore disponibilità.

Configurazione personalizzata:

- Requisiti hardware specifici: imposta i tipi di istanza consentiti su un elenco di istanze compatibili. Ad esempio, se hai un'applicazione che richiede un'elevata larghezza di banda di rete, puoi selezionare diversi tipi di istanze n
- Ottimizzazione dei costi: per gli ambienti di test o sviluppo, potresti scegliere tipi di istanza più piccoli, come i tipi di istanza `m7a.large`

## Dimensionamento pianificato

Usa [Amazon EventBridge Scheduler](#) per modificare gli ambienti di esecuzione minima e massima della tua funzione in base a una pianificazione ricorrente o una tantum. Ciò è utile per modelli di traffico prevedibili, ad esempio la scalabilità verso l'alto prima delle ore di punta e la riduzione durante le ore non di punta.

Configurazione dello scheduler:

- Crea un ruolo di esecuzione EventBridge dello Scheduler o utilizza un ruolo esistente che conceda il permesso di chiamare la funzione di `lambda:PutFunctionScalingConfig` destinazione.
- Crea una pianificazione utilizzando un'espressione cron o rate, indirizzando l'`PutFunctionScalingConfigAPI` come destinazione universale. Specifica i nuovi `MaxExecutionEnvironments` valori `MinExecutionEnvironments` e nel payload di input.

Esempio 1: scalabilità per gestire i picchi di traffico pianificati

Crea due pianificazioni da scalare prima delle ore di punta e ridimensionarle in seguito. Ogni pianificazione è indirizzata all'`PutFunctionScalingConfigAPI` con `MaxExecutionEnvironments` valori `MinExecutionEnvironments` e dati aggiornati.

Scalabilità alle 8:00 UTC (min=100, max=1000):

```
aws scheduler create-schedule \  
  --name "ScaleUpLambdaManagedInstances" \  
  --schedule-expression "cron(0 8 * * ? *)" \  
  --flexible-time-window '{"Mode": "OFF"}' \  
  --target '{  
    "Arn": "arn:aws:scheduler::aws-sdk:lambda:PutFunctionScalingConfig",  
    "RoleArn": "arn:aws:iam::<account-id>:role/eventbridge-scheduler-role",  
    "Input": "{\"FunctionName\": \"my-lmi-function\", \"Qualifier\":  
\"$LATEST.PUBLISHED\", \"FunctionScalingConfig\": {\"MinExecutionEnvironments\": 100,  
\"MaxExecutionEnvironments\": 1000}}"  
  }'
```

Ridurre la scala alle 18:00 UTC (min=5, max=20):

```
aws scheduler create-schedule \  
  --name "ScaleDownLambdaManagedInstances" \  
  --schedule-expression "cron(0 18 * * ? *)" \  
  --flexible-time-window '{"Mode": "OFF"}' \  

```

```
--target '{
  "Arn": "arn:aws:scheduler::aws-sdk:lambda:PutFunctionScalingConfig",
  "RoleArn": "arn:aws:iam::<account-id>:role/eventbridge-scheduler-role",
  "Input": "{\"FunctionName\": \"my-lmi-function\", \"Qualifier\":
\\$LATEST.PUBLISHED\", \"FunctionScalingConfig\": {\"MinExecutionEnvironments\": 5,
\\MaxExecutionEnvironments\": 20}}\"
}'
```

## Esempio 2: disattiva durante le ore non di punta e riattiva

L'impostazione di entrambi `MinExecutionEnvironments` e `MaxExecutionEnvironments` su 0 disattiva la versione della funzione senza eliminarla. Una funzione disattivata non viene ridimensionata automaticamente in base al traffico. È necessario riattivarla esplicitamente impostando valori diversi da zero tramite un'altra azione pianificata.

Disattiva alle 22:00 UTC (min=0, max=0):

```
aws scheduler create-schedule \
  --name "DeactivateLambdaManagedInstances" \
  --schedule-expression "cron(0 22 * * ? *)" \
  --flexible-time-window '{"Mode": "OFF"}' \
  --target '{
    "Arn": "arn:aws:scheduler::aws-sdk:lambda:PutFunctionScalingConfig",
    "RoleArn": "arn:aws:iam::<account-id>:role/eventbridge-scheduler-role",
    "Input": "{\"FunctionName\": \"my-lmi-function\", \"Qualifier\":
\\$LATEST.PUBLISHED\", \"FunctionScalingConfig\": {\"MinExecutionEnvironments\": 0,
\\MaxExecutionEnvironments\": 0}}\"
  }'
```

Riattiva alle 7:00 UTC (min=10, max=20):

```
aws scheduler create-schedule \
  --name "ReactivateLambdaManagedInstances" \
  --schedule-expression "cron(0 7 * * ? *)" \
  --flexible-time-window '{"Mode": "OFF"}' \
  --target '{
    "Arn": "arn:aws:scheduler::aws-sdk:lambda:PutFunctionScalingConfig",
    "RoleArn": "arn:aws:iam::<account-id>:role/eventbridge-scheduler-role",
    "Input": "{\"FunctionName\": \"my-lmi-function\", \"Qualifier\":
\\$LATEST.PUBLISHED\", \"FunctionScalingConfig\": {\"MinExecutionEnvironments\": 10,
\\MaxExecutionEnvironments\": 20}}\"
  }'
```



## Linee guida per la regolazione:

- Per carichi di lavoro con picchi prevedibili, crea più pianificazioni in base al tuo modello di traffico: una per aumentare la funzionalità prima delle ore di punta e un'altra per ridurla dopo le ore di punta. Ogni pianificazione segue lo stesso schema con valori e aggiornamenti `MinExecutionEnvironments`. `MaxExecutionEnvironments`
- Il ridimensionamento pianificato modifica il livello minimo e massimo previsto per gli ambienti di esecuzione, ma il ridimensionamento effettivo tra valori minimi e massimi risponde comunque all'utilizzo della CPU e alla saturazione della concorrenza.
- Se il traffico raddoppia di più entro 5 minuti da uno scale-up pianificato, è possibile che si verifichino comunque delle limitazioni man mano che viene fornita la capacità.
- Quando scalate a zero per disattivare una funzione, ricordate che la riattivazione richiede una chiamata esplicita con valori diversi da zero. `PutFunctionScalingConfig`

## Fasi successive

- Scopri i fornitori di [capacità per le istanze gestite Lambda](#)
- Consulta le guide specifiche per il runtime per la gestione della concorrenza multipla
- Configura la connettività [VPC per i tuoi fornitori di capacità](#)
- Monitora le metriche di scalabilità per ottimizzare il comportamento di scalabilità

## Sicurezza e autorizzazioni

Le istanze gestite Lambda utilizzano i fornitori di capacità come limiti di fiducia. Le funzioni vengono eseguite in contenitori all'interno di queste istanze, ma i contenitori non forniscono l'isolamento di sicurezza tra i carichi di lavoro. Tutte le funzioni assegnate allo stesso fornitore di capacità devono essere considerate attendibili reciprocamente.

## Concetti chiave di sicurezza

- Capacity Provider: il limite di sicurezza che definisce i livelli di affidabilità per le funzioni Lambda
- Isolamento dei container: i container non rappresentano un limite di sicurezza: non affidatevi ad essi per la sicurezza tra carichi di lavoro non attendibili
- Separazione della fiducia: separa i carichi di lavoro che non sono reciprocamente affidabili utilizzando fornitori di capacità diversi

## Autorizzazioni richieste

### PassCapacityProvider Azione

Gli utenti devono `lambda:PassCapacityProvider` disporre dell'autorizzazione per assegnare funzioni ai fornitori di capacità. Questa autorizzazione funge da barriera di sicurezza, garantendo che solo gli utenti autorizzati possano collocare funzioni in fornitori di capacità specifici.

Gli amministratori degli account controllano quali funzioni possono utilizzare fornitori di capacità specifici tramite l'azione `lambda:PassCapacityProvider IAM`. Questa azione è necessaria quando:

- Creazione di funzioni che utilizzano Lambda Managed Instances
- Aggiornamento delle configurazioni delle funzioni per utilizzare un fornitore di capacità
- Implementazione delle funzioni utilizzando l'infrastruttura come codice

### Esempio di politica IAM

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:PassCapacityProvider",
      "Resource": "arn:aws:lambda:*:*:capacity-provider:trusted-workloads-*"
    }
  ]
}
```

### Service-Linked Ruolo

AWS Lambda utilizza il ruolo `AWSServiceRoleForLambda` collegato ai servizi per gestire le risorse `ec2` di Lambda Managed Instances nei tuoi capacity provider.

### Best practice

1. Separato per livello di affidabilità: crea diversi fornitori di capacità per carichi di lavoro con requisiti di sicurezza diversi
2. Usa nomi descrittivi: nomina i fornitori di capacità per indicare chiaramente l'uso previsto e il livello di affidabilità (ad esempio,,) `production-trusted dev-sandbox`

3. Applica il privilegio minimo: concedi `PassCapacityProvider` le autorizzazioni solo ai fornitori di capacità necessari
4. Monitora l'utilizzo: utilizzato AWS CloudTrail per monitorare le assegnazioni dei fornitori di capacità e i modelli di accesso

### Fasi successive

- Scopri i fornitori di [capacità per le istanze gestite Lambda](#)
- Scopri la [scalabilità per le istanze gestite Lambda](#)
- Configura la connettività [VPC per i tuoi fornitori di capacità](#)
- Consulta le guide specifiche per il runtime per [Java e Python](#) [Node.js](#) ???

## Ruolo dell'operatore Lambda per le istanze gestite Lambda

Quando utilizzi Lambda Managed Instances, Lambda necessita delle autorizzazioni per gestire la capacità di elaborazione nel tuo account. Il ruolo operatore fornisce queste autorizzazioni tramite policy IAM che consentono a Lambda di gestire le istanze EC2 nel provider di capacità.

Lambda assume il ruolo di operatore durante l'esecuzione di queste operazioni di gestione, in modo simile a come Lambda assume un ruolo di esecuzione durante l'esecuzione della funzione.

### Creazione di un ruolo di operatore

Puoi creare un ruolo operatore nella console IAM o con la AWS CLI. Il ruolo deve includere:

- Politica delle autorizzazioni: concede le autorizzazioni per gestire i provider di capacità e le risorse associate
- Politica di fiducia: consente al servizio Lambda (`lambda.amazonaws.com`) di assumere il ruolo

### Policy delle autorizzazioni

Il ruolo di operatore richiede le autorizzazioni per gestire i provider di capacità e le risorse di elaborazione sottostanti. Come minimo, il ruolo richiede le autorizzazioni previste nella politica [AWSLambdaManagedEC2ResourceOperator](#) gestita, attualmente:

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
{
  "Effect": "Allow",
  "Action": [
    "ec2:RunInstances",
    "ec2:CreateTags",
    "ec2:AttachNetworkInterface"
  ],
  "Resource": [
    "arn:aws:ec2:*:*:instance/*",
    "arn:aws:ec2:*:*:network-interface/*",
    "arn:aws:ec2:*:*:volume/*"
  ],
  "Condition": {
    "StringEquals": {
      "ec2:ManagedResourceOperator": "scaler.lambda.amazonaws.com"
    }
  }
},
{
  "Effect": "Allow",
  "Action": [
    "ec2:DescribeAvailabilityZones",
    "ec2:DescribeCapacityReservations",
    "ec2:DescribeInstances",
    "ec2:DescribeInstanceStatus",
    "ec2:DescribeInstanceTypeOfferings",
    "ec2:DescribeInstanceTypes",
    "ec2:DescribeSecurityGroups",
    "ec2:DescribeSubnets",
    "ec2:DescribeVpcEncryptionControls"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "ec2:RunInstances",
    "ec2:CreateNetworkInterface"
  ],
  "Resource": [
    "arn:aws:ec2:*:*:subnet/*",
    "arn:aws:ec2:*:*:security-group/*"
  ]
},
```

```
{
  "Effect": "Allow",
  "Action": [
    "ec2:RunInstances"
  ],
  "Resource": [
    "arn:aws:ec2:*:*:image/*"
  ],
  "Condition": {
    "StringEquals": {
      "ec2:Owner": "amazon"
    }
  }
},
{
  "Effect": "Allow",
  "Action": [
    "logs:CreateLogGroup"
  ],
  "Resource": "arn:aws:logs:*:*:log-group:*"
},
{
  "Effect": "Allow",
  "Action": [
    "logs:CreateLogStream",
    "logs:PutLogEvents"
  ],
  "Resource": "arn:aws:logs:*:*:log-group:*:log-stream:*"
}
]
```

## Policy di attendibilità

La policy di fiducia consente a Lambda di assumere il ruolo di operatore:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
```

```
    "Action": "sts:AssumeRole"
  }
]
}
```

## Service-Linked Ruolo per le istanze gestite da Lambda

Per gestire in modo responsabile il ciclo di vita delle istanze gestite Lambda, Lambda richiede un accesso persistente per terminare le istanze gestite nel tuo account. Lambda utilizza un ruolo collegato al servizio (SLR) AWS Identity and Access Management (IAM) per eseguire queste operazioni.

Creazione automatica: il ruolo collegato al servizio viene creato automaticamente la prima volta che si crea un provider di capacità. L'utente che crea il primo fornitore di capacità deve disporre dell'`iam:CreateServiceLinkedRole` autorizzazione per il `lambda.amazonaws.com` principale.

Autorizzazioni: il ruolo collegato al servizio concede a Lambda le seguenti autorizzazioni sulle istanze gestite:

- `ec2:TerminateInstances`— Per terminare le istanze alla fine del loro ciclo di vita
- `ec2:DescribeInstances`— Per enumerare le istanze gestite

Eliminazione: puoi eliminare questo ruolo collegato al servizio solo dopo aver eliminato tutti i provider di capacità di Lambda Managed Instances nel tuo account.

Per ulteriori informazioni sui ruoli collegati al servizio, consulta [the section called “Uso di ruoli collegati ai servizi”](#).

## Comprendere l'ambiente di esecuzione delle istanze gestite Lambda

Le istanze gestite Lambda forniscono un modello di distribuzione alternativo che esegue il codice funzionale su istanze Amazon EC2 di proprietà del cliente mentre Lambda gestisce gli aspetti operativi. L'ambiente di esecuzione per le istanze gestite presenta diverse importanti differenze rispetto alle funzioni Lambda (predefinite), in particolare nel modo in cui gestisce le chiamate simultanee e gestisce i cicli di vita dei container.

Nota: per informazioni sull'ambiente di esecuzione Lambda (predefinito), consulta [Understanding the Lifecycle dell'ambiente di esecuzione Lambda](#).

## Ciclo di vita dell'ambiente di esecuzione

Il ciclo di vita di un ambiente di esecuzione delle funzioni Lambda Managed Instances è diverso da Lambda (predefinito) in diversi modi chiave:

### Fase di init

Durante la fase di avvio, Lambda esegue i seguenti passaggi:

- Inizializza e registra tutte le estensioni
- Avvia il punto di ingresso del runtime. Il runtime genera il numero configurato di runtime worker (l'implementazione dipende dal runtime)
- Esegui il codice di inizializzazione della funzione (codice esterno al gestore)
- Attendi che almeno un runtime worker segnali la disponibilità chiamando `/runtime/invocation/next`

La fase Init è considerata completa quando le estensioni sono state inizializzate e almeno un runtime worker è stato chiamato. `/runtime/invocation/next` La funzione è quindi pronta per elaborare le chiamate.

#### Note

Per le funzioni Lambda Managed Instances, l'inizializzazione può richiedere fino a 15 minuti. Il timeout è il massimo di 130 secondi o il timeout della funzione configurata (fino a 900 secondi).

### Invoca fase

La fase Invoke per le funzioni Lambda Managed Instances presenta diverse caratteristiche uniche:

**Funzionamento continuo.** A differenza di Lambda (impostazione predefinita), l'ambiente di esecuzione rimane continuamente attivo, elaborando le chiamate man mano che arrivano senza bloccarsi tra una chiamata e l'altra.

**Elaborazione parallela.** È possibile eseguire più invocazioni contemporaneamente all'interno dello stesso ambiente di esecuzione, ciascuna gestita da un runtime worker diverso.

Timeout indipendenti. Il timeout configurato della funzione si applica a ogni singola chiamata. Quando una chiamata scade, Lambda contrassegna quella specifica invocazione come fallita ma non interrompe altre chiamate in esecuzione né termina l'ambiente di esecuzione.

Gestione della contropressione. Se tutti i runtime worker sono impegnati nell'elaborazione delle chiamate, le nuove richieste di richiamo vengono rifiutate fino a quando un worker non diventa disponibile.

## Gestione e ripristino degli errori

La gestione degli errori negli ambienti di esecuzione delle funzioni Lambda Managed Instances è diversa da Lambda (impostazione predefinita):

Errori del runtime worker. Se un processo di runtime worker si blocca, l'ambiente di esecuzione continua a funzionare con i lavoratori sani rimanenti.

L'estensione si blocca. Se un processo di estensione si blocca durante l'inizializzazione o l'operazione, l'intero ambiente di esecuzione viene contrassegnato come non integro e viene terminato. Lambda crea un nuovo ambiente di esecuzione per sostituirlo.

No reset/repair A differenza di Lambda (impostazione predefinita), le istanze gestite non tentano di reimpostare e reinizializzare l'ambiente di esecuzione dopo gli errori. Invece, i contenitori non integri vengono terminati e sostituiti con contenitori nuovi.

## Richiama i timeout

Quando una singola chiamata scade, Lambda restituisce al chiamante un `Task timed out after <timeout> seconds` errore con lo stato di errore di funzione. Tuttavia, Lambda Managed Instances non termina forzatamente il codice, ma continua a funzionare nell'ambiente di esecuzione. In qualità di sviluppatore di funzioni, sei responsabile del rilevamento e della gestione del timeout.

L'oggetto `context` espone il tempo rimanente per l'invocazione. Un valore zero o negativo indica che l'invocazione è scaduta. Le altre chiamate simultanee nell'ambiente di esecuzione continuano a essere elaborate normalmente.

## Comportamento di ripetizione

Quando scade il timeout di una chiamata:

- Chiamate sincrone: il chiamante riceve l'errore di timeout ed è responsabile del nuovo tentativo.



- Chiamate asincrone: Lambda riprova in base alla politica di ripetizione della funzione (impostazione predefinita: 2 tentativi). Una volta esauriti tutti i tentativi, l'evento viene inviato alla coda di lettere morte configurata o alla destinazione in caso di errore, se presente.
- Mappature dell'origine degli eventi: il comportamento dei tentativi dipende dalla configurazione dell'origine dell'evento (ad esempio, dimensione del batch, bisezione in base all'errore, numero massimo di tentativi di ripetizione). Il batch può essere ritentato o inviato a una destinazione in caso di errore in base alle politiche relative ai nuovi tentativi.

## Cosa succede se non gestisci il timeout

Se il codice non controlla il tempo rimanente e interrompe l'esecuzione:

- L'invocazione è già contrassegnata come fallita. Lambda ha già restituito un errore di timeout al chiamante: qualsiasi lavoro completato dal codice dopo il timeout viene effettivamente perso dal punto di vista del chiamante.
- Le risorse rimangono consumate. Il codice continua a occupare uno slot di runtime worker, riducendo la concorrenza disponibile per le nuove chiamate su quell'istanza.
- Comportamento non deterministico. Il codice non si interrompe quando scatta il timeout, ma continua a funzionare in background. Ciò significa che gli effetti indesiderati possono ancora verificarsi dopo che Lambda ha già comunicato al chiamante che l'invocazione è fallita.

Ad esempio, il gestore scrive un record su DynamoDB, quindi si attiva il timeout e Lambda restituisce un errore di timeout al chiamante, ma il codice è ancora in esecuzione e procede con l'invio di una notifica SNS. Il chiamante riprova l'invocazione, che scrive nuovamente il record e invia nuovamente la notifica. Ora hai dati e notifiche duplicati e non c'è un modo semplice per stabilire quali provenissero dalla chiamata «fallita» ancora in esecuzione in background.

## Gestione dei timeout nel codice

Usa l'oggetto `context` per controllare il tempo rimanente e interrompere l'elaborazione prima del timeout. Configura un buffer in base alla durata prevista della prossima unità di lavoro: ad esempio, se l'elaborazione di ogni elemento richiede circa 500 ms, imposta il buffer su almeno 500 ms più il margine.

Per esempi di gestione dei timeout specifici per lingua, consulta la sezione relativa al contesto della richiesta di ciascuna pagina di runtime:

- [Node.js contesto della richiesta](#)

- [contesto della richiesta Python](#)
- [Contesto della richiesta Java](#)
- [Contesto della richiesta.NET](#)

## LATEST.PUBLISHED versione \$ nelle istanze gestite Lambda

Le funzioni Lambda Managed Instances supportano lo stesso flusso di lavoro di controllo delle versioni numerato di Lambda (impostazione predefinita). Se preferisci non mantenere le versioni numerate, Lambda Managed Instances introduce un nuovo tipo di versione: `$LATEST.PUBLISHED`. Con questo tipo di versione, puoi creare o ripubblicare una versione pubblicata con codice o configurazione aggiornati. Non è necessario gestire le versioni numerate.

Differenza fondamentale rispetto a `$LATEST`: quando si richiama una funzione Lambda Managed Instances utilizzando un ARN non qualificato, Lambda richiama implicitamente la versione anziché la versione `$LATEST` non pubblicata. `$LATEST.PUBLISHED`

Il seguente comando `$LATEST.PUBLISHED CLI` crea o ripubblica la versione. AWS

```
aws lambda publish-version --function-name my-function --publish-to LATEST_PUBLISHED
```

Verrà visualizzato l'output seguente:

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function:
$LATEST.PUBLISHED",
  "Version": "$LATEST.PUBLISHED",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "function.handler",
  "Runtime": "nodejs24.x",
  ...
}
```

### Note

Se si utilizza AWS CloudFormation la console Lambda per creare una funzione Lambda Managed Instances, Lambda crea automaticamente la versione. `$LATEST.PUBLISHED`

## Runtime di esecuzione delle istanze gestite Lambda

Lambda elabora le richieste in modo diverso quando utilizza istanze gestite Lambda. Invece di gestire le richieste in sequenza in ogni ambiente di esecuzione, le istanze gestite Lambda elaborano più richieste contemporaneamente all'interno di ciascun ambiente di esecuzione. Questa modifica nel modello di esecuzione significa che le funzioni che utilizzano le istanze gestite Lambda devono considerare la sicurezza dei thread, la gestione dello stato e l'isolamento del contesto, problemi che non sorgono nel modello a concorrenza singola Lambda (predefinito). Inoltre, l'implementazione multi-concorrenza varia da un runtime all'altro.

### Lingue supportate

Le istanze gestite Lambda possono essere utilizzate con i seguenti linguaggi di programmazione e runtime:

- Java: Java 21 e versioni successive.
- Python: Python 3.13 e versioni successive.
- Node.js: Node.js 22 e versioni successive.
- .NET: .NET 8 e versioni successive.
- Rust: supportato utilizzando il runtime solo per il sistema operativo `provided.al2023` e versioni successive.

### Considerazioni specifiche sulla lingua

Ogni linguaggio di programmazione implementa la multiconcorrenza in modo diverso. È necessario comprendere come viene implementata la concorrenza multipla nel linguaggio di programmazione prescelto per applicare le migliori pratiche di concorrenza appropriate.

#### Java

Utilizza un unico processo con thread del sistema operativo per la concorrenza. Più thread eseguono il metodo handler contemporaneamente, richiedendo una gestione thread-safe delle risorse statali e condivise.

#### Python

Utilizza più processi Python in cui ogni richiesta simultanea viene eseguita in un processo separato. Ciò protegge dalla maggior parte dei problemi di concorrenza, sebbene sia necessaria attenzione per le risorse condivise come la `directory. /tmp`

## Node.js

Utilizza [thread di lavoro con esecuzione](#) asincrona. Le richieste concorrenti sono distribuite tra i thread di lavoro e ogni thread di lavoro può anche gestire le richieste simultanee in modo asincrono, il che richiede una gestione sicura delle risorse statali e condivise.

## .NET

Utilizza .NET Tasks con l'elaborazione asincrona di più richieste simultanee. Richiede una gestione sicura delle risorse statali e condivise.

## Rust

[Utilizza un unico processo con attività asincrone basate su Tokio](#). Il gestore deve essere `+`. Clone Send

## Fasi successive

Per informazioni dettagliate su ogni runtime, consultate i seguenti argomenti:

- [Runtime Java per istanze gestite Lambda](#)
- [Runtime Node.js per istanze gestite Lambda](#)
- [Runtime Python per istanze gestite Lambda](#)
- [.NET runtime per istanze gestite Lambda](#)
- [Supporto Rust per istanze gestite Lambda](#)

## Runtime Java per istanze gestite Lambda

Per i runtime Java, le istanze gestite Lambda utilizzano i thread del sistema operativo per la concorrenza. Lambda carica l'oggetto handler una volta per ambiente di esecuzione durante l'inizializzazione e quindi crea più thread. Questi thread vengono eseguiti in parallelo e richiedono una gestione thread-safe dello stato e delle risorse condivise. Ogni thread condivide lo stesso oggetto gestore e tutti i campi statici.

## Configurazione della concorrenza

Il numero massimo di richieste simultanee che Lambda invia a ciascun ambiente di esecuzione è controllato dall'`PerExecutionEnvironmentMaxConcurrency` impostazione nella configurazione della funzione. Questa è un'impostazione opzionale e il valore predefinito varia a seconda del

runtime. Per i runtime Java, l'impostazione predefinita è 32 richieste simultanee per vCPU, oppure è possibile configurare un valore personalizzato. Questo valore determina anche il numero di thread utilizzati dal runtime Java. Lambda regola automaticamente il numero di richieste simultanee fino al valore massimo configurato in base alla capacità di ciascun ambiente di esecuzione di assorbire tali richieste.

### Creazione di funzioni per la multi-concorrenza

È necessario applicare le stesse pratiche di sicurezza dei thread quando si utilizzano le istanze gestite Lambda come si farebbe in qualsiasi altro ambiente multithread. Poiché l'oggetto handler è condiviso tra tutti i thread di lavoro in fase di esecuzione, qualsiasi stato mutabile deve essere thread-safe. Ciò include raccolte, connessioni al database e qualsiasi oggetto statico modificato durante l'elaborazione delle richieste.

AWS I client SDK sono thread-safe e non richiedono una gestione speciale.

### Esempio: pool di connessioni al database

Il codice seguente utilizza un oggetto di connessione al database statico condiviso tra i thread. A seconda della libreria di connessione utilizzata, potrebbe non essere thread-safe.

```
public class DBQueryHandler implements RequestHandler<Object, String> {
    // Single connection shared across all threads - NOT SAFE
    private static Connection connection;

    public DBQueryHandler() {
        this.connection = DriverManager.getConnection(jdbcUrl, username, password);
    }

    @Override
    public String handleRequest(Object input, Context context) {
        PreparedStatement stmt = connection.prepareStatement(query);
        ResultSet rs = stmt.executeQuery();
        // Multiple threads using same connection causes issues
        return result.toString();
    }
}
```

Un approccio thread-safe consiste nell'utilizzare un pool di connessioni. Nell'esempio seguente, il gestore della funzione recupera una connessione dal pool. La connessione viene utilizzata solo nel contesto di una singola richiesta.

```

public class DBQueryHandler implements RequestHandler<Object, String> {

    private static HikariDataSource dataSource;

    public DBQueryHandler() {
        HikariConfig config = new HikariConfig();
        config.setJdbcUrl("jdbc:mysql://localhost:3306/your_database");
        dataSource = new HikariDataSource(config); // Create pool once per Lambda
container
    }

    @Override
    public String handleRequest(Object input, Context context) {
        String query = "SELECT column_name FROM your_table LIMIT 10";
        StringBuilder result = new StringBuilder("Data:\n");

        // try-with-resources automatically calls close() on the connection,
        // which returns it to the HikariCP pool (does NOT close the physical DB
connection)
        try (Connection connection = dataSource.getConnection();
            PreparedStatement stmt = connection.prepareStatement(query);
            ResultSet rs = stmt.executeQuery()) {

            while (rs.next()) {
                result.append(rs.getString("column_name")).append("\n");
            }

        } catch (Exception e) {
            context.getLogger().log("Error: " + e.getMessage());
            return "Error";
        }

        return result.toString();
    }
}

```

## Esempio: collezioni

Le raccolte Java standard non sono thread-safe:

```

public class Handler implements RequestHandler<Object, String> {
    private static List<String> items = new ArrayList<>();
    private static Map<String, Object> cache = new HashMap<>();

```

```
@Override
public String handleRequest(Object input, Context context) {
    items.add("list item"); // Not thread-safe
    cache.put("key", input); // Not thread-safe
    return "Success";
}
}
```

Usa invece le raccolte thread-safe:

```
public class Handler implements RequestHandler<Object, String> {
    private static final List<String> items =
        Collections.synchronizedList(new ArrayList<>());
    private static final ConcurrentHashMap<String, Object> cache =
        new ConcurrentHashMap<>();

    @Override
    public String handleRequest(Object input, Context context) {
        items.add("list item"); // Thread-safe
        cache.put("key", input); // Thread-safe
        return "Success";
    }
}
```

## Directory /tmp condivisa

La /tmp directory è condivisa tra tutte le richieste simultanee nell'ambiente di esecuzione. Le scritture simultanee sullo stesso file possono causare il danneggiamento dei dati, ad esempio se un altro processo sovrascrive il file. Per risolvere questo problema, implementa il blocco dei file condivisi o utilizza nomi di file univoci per thread o per richiesta per evitare conflitti. Ricordati di pulire i file non necessari per evitare di esaurire lo spazio disponibile.

## Registrazione dei log

L'interlacciamento dei log (voci di log provenienti da richieste diverse che vengono interlacciate nei log) è normale nei sistemi con più concorrenti.

Le funzioni che utilizzano Lambda Managed Instances utilizzano sempre il formato di registro JSON strutturato introdotto con controlli di registrazione avanzati. [???](#) Questo formato include il requestId, che consente di correlare le voci di registro a una singola richiesta. Quando si utilizza

L'LambdaLoggeroggetto, requestId viene automaticamente incluso in ogni voce di registro. context.getLogger() Per ulteriori informazioni, vedere [the section called “Utilizzo dei controlli di registrazione avanzati di Lambda con Java”](#).

## Contesto della richiesta

L'contextoggetto è associato al thread della richiesta. L'utilizzo context.getAwsRequestId() fornisce un accesso thread-safe all'ID della richiesta corrente.

context.getXrayTraceId() Usare per accedere all'ID di traccia X-Ray . Ciò fornisce un accesso thread-safe all'ID di traccia per la richiesta corrente. Lambda non supporta la variabile di \_X\_AMZN\_TRACE\_ID ambiente con Lambda Managed Instances. L'ID di X-Ray traccia viene propagato automaticamente quando si utilizza l'SDK. AWS

## Utilizzato

com.amazonaws.services.lambda.runtime.Context.getRemainingTimeInMillis() per rilevare i timeout. Per ulteriori informazioni, consulta [the section called “Gestione e ripristino degli errori”](#).

Se si utilizzano thread virtuali nel programma o si creano thread durante l'inizializzazione, è necessario passare a questi thread qualsiasi contesto di richiesta richiesto.

## Esempio: gestione del timeout

Controlla il tempo rimanente prima di ogni unità di lavoro e interrompi l'elaborazione prima che scada il timeout. Configura in BUFFER\_MS base alla durata prevista della prossima parte di lavoro.

```
private static final int BUFFER_MS = 2000; // Configure based on your next chunk of
work

public Map<String, Object> handleRequest(Map<String, Object> event, Context context) {
    for (Object item : (List<Object>) event.get("items")) {
        if (context.getRemainingTimeInMillis() < BUFFER_MS)
            return Map.of("statusCode", 206, "body", "Timeout approaching, stopping
early");
        processItem(item);
    }
    return Map.of("statusCode", 200, "body", "Done");
}
```

## Esempio: propagazione delle scadenze alle chiamate a valle



Quando effettui chiamate ai servizi downstream, propaga il tempo rimanente sotto forma di timeout per evitare il blocco delle chiamate di rete che potrebbero durare più a lungo della chiamata. Utilizzate un override per richiesta su un client condiviso anziché creare un nuovo client per chiamata:

```
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import java.time.Duration;

private static final S3Client s3 = S3Client.create();

public String handleRequest(Map<String, Object> event, Context context) {
    Duration timeout = Duration.ofMillis(Math.max(1000,
        context.getRemainingTimeInMillis() - 500));

    GetObjectRequest req = GetObjectRequest.builder()
        .bucket("my-bucket").key("my-key")
        .overrideConfiguration(cfg -> cfg.apiCallTimeout(timeout))
        .build();
    s3.getObject(req);
    return "Done";
}
```

## Inizializzazione e spegnimento

L'inizializzazione della funzione avviene una volta per ambiente di esecuzione. Gli oggetti creati durante l'inizializzazione vengono condivisi tra i thread.

Per le funzioni Lambda con estensioni, l'ambiente di esecuzione emette un segnale SIGTERM durante l'arresto. Questo segnale viene utilizzato dalle estensioni per attivare attività di pulizia, come lo svuotamento dei buffer. È possibile iscriversi agli eventi SIGTERM per attivare attività di pulizia delle funzioni, come la chiusura delle connessioni al database. Per ulteriori informazioni sul ciclo di vita dell'ambiente di esecuzione, consulta [the section called “Ambiente di esecuzione”](#).

## Versioni di dipendenza

Lambda Managed Instances richiede le seguenti versioni minime del pacchetto:

- AWS SDK per Java 2.0: versione 2.34.0 o successiva
- AWS X-Ray SDK per Java: versione 2.20.0 o successiva
- AWS Distro for OpenTelemetry - Instrumentation for Java: versione 2.20.0 o successiva
- Powertools for AWS Lambda (Java): versione 2.8.0 o successiva

## Powertools per AWS Lambda (Java)

Powertools for AWS Lambda (Java) è compatibile con Lambda Managed Instances e fornisce utilità per la registrazione, la traccia, le metriche e altro ancora. Per ulteriori informazioni, consulta Powertools for Lambda (Java). [AWS](#)

### Fasi successive

- Esamina il [Node.js runtime per le istanze gestite Lambda](#)
- Esamina il runtime di [Python per le istanze gestite Lambda](#)
- Esamina il runtime [.NET per le istanze gestite Lambda](#)
- Scopri come [scalare le istanze gestite Lambda](#)

## Node.js runtime per le istanze gestite Lambda

Per i Node.js runtime, Lambda Managed Instances utilizza thread di lavoro con esecuzione `await` basata su `async` per gestire le richieste simultanee. L'inizializzazione della funzione avviene una volta per thread di lavoro. Le chiamate simultanee vengono gestite su due dimensioni: i thread di lavoro forniscono il parallelismo tra le vCPU e l'esecuzione asincrona fornisce la concorrenza all'interno di ciascun thread. Ogni richiesta simultanea gestita dallo stesso thread di lavoro condivide lo stesso oggetto gestore e lo stesso stato globale, richiedendo una gestione sicura in caso di più richieste simultanee.

### Simultaneità massima

Il numero massimo di richieste simultanee che Lambda invia a ciascun ambiente di esecuzione è controllato dall'impostazione nella configurazione della `PerExecutionEnvironmentMaxConcurrency` funzione. Questa è un'impostazione opzionale e il valore predefinito varia a seconda del runtime. Per i Node.js runtime, l'impostazione predefinita è 64 richieste simultanee per vCPU, oppure è possibile configurare un valore personalizzato. Lambda regola automaticamente il numero di richieste simultanee fino al numero massimo configurato in base alla capacità di ciascun ambiente di esecuzione di assorbire tali richieste.

Infatti Node.js, il numero di richieste simultanee che ogni ambiente di esecuzione può elaborare è determinato dal numero di thread di lavoro e dalla capacità di ciascun thread di lavoro di elaborare le richieste simultanee in modo asincrono. Il numero predefinito di thread di lavoro è determinato dal numero di vCPU disponibili oppure è possibile configurare il numero di thread di lavoro impostando la variabile di ambiente. `AWS_LAMBDA_NODEJS_WORKER_COUNT` Si consiglia di utilizzare gestori di

funzioni asincroni poiché ciò consente l'elaborazione di più richieste per thread di lavoro. Se il gestore di funzioni è sincrono, ogni thread di lavoro può elaborare una sola richiesta alla volta.

### Creazione di funzioni per la multi-concorrenza

Con un gestore di funzioni asincrone, ogni runtime worker elabora più richieste contemporaneamente. Gli oggetti globali sono condivisi tra più richieste simultanee. Per gli oggetti mutabili, evita di utilizzare lo stato o l'uso globali. `AsyncLocalStorage`

AWS I client SDK sono asincroni e non richiedono una gestione speciale.

### Esempio: stato globale

Il codice seguente utilizza un oggetto globale che è mutato all'interno del gestore della funzione. Questo non è asincrono.

```
let state = {
  currentUser: null,
  requestData: null
};

export const handler = async (event, context) => {
  state.currentUser = event.userId;
  state.requestData = event.data;

  await processData(state.requestData);

  // state.currentUser might now belong to a different request
  return { user: state.currentUser };
};
```

L'inizializzazione dell'`state` oggetto all'interno del gestore di funzioni evita lo stato globale condiviso.

```
export const handler = async (event, context) => {
  let state = {
    currentUser: event.userId,
    requestData: event.data
  };

  await processData(state.requestData);

  return { user: state.currentUser };
};
```

```
};
```

## Esempio: connessioni al database

Il codice seguente utilizza un oggetto client condiviso condiviso tra più invocazioni. A seconda della libreria di connessione utilizzata, questa potrebbe non essere sicura per la concorrenza.

```
const { Client } = require('pg');

// Single connection created at init time
const client = new Client({
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD
});

// Connect once during cold start
client.connect();

exports.handler = async (event) => {
  // Multiple parallel invocations share this single connection = BAD
  // With multi-concurrent Lambda, queries will collide
  const result = await client.query('SELECT * FROM users WHERE id = $1',
    [event.userId]);

  return {
    statusCode: 200,
    body: JSON.stringify(result.rows[0])
  };
};
```

Un approccio sicuro per la concorrenza consiste nell'utilizzare un pool di connessioni. Il pool utilizza una connessione separata per ogni interrogazione simultanea del database.

```
const { Pool } = require('pg');

// Connection pool created at init time
const pool = new Pool({
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
```

```
max: 20, // Max connections in pool
idleTimeoutMillis: 30000,
connectionTimeoutMillis: 2000
});

exports.handler = async (event) => {
  // Pool gives each parallel invocation its own connection
  const result = await pool.query('SELECT * FROM users WHERE id = $1', [event.userId]);

  return {
    statusCode: 200,
    body: JSON.stringify(result.rows[0])
  };
};
```

## Node.js 22 gestori basati su callback

Quando si utilizza Node.js 22, non è possibile utilizzare un gestore di funzioni basato su callback con Lambda Managed Instances. Callback-based i gestori sono supportati solo per le funzioni Lambda (predefinite). Per i runtime Node.js 24 e successivi, i gestori di funzioni basati su callback sono obsoleti sia per le istanze Lambda (impostazione predefinita) che per le istanze gestite Lambda.

Utilizza invece un gestore di funzioni quando usi Lambda Managed Instances. `async` Per ulteriori informazioni, consulta [Define Lambda function handler in. Node.js](#)

## Directory /tmp condivisa

La /tmp directory è condivisa tra tutte le richieste simultanee nell'ambiente di esecuzione. Le scritture simultanee sullo stesso file possono causare il danneggiamento dei dati, ad esempio se un altro processo sovrascrive il file. Per risolvere questo problema, implementa il blocco dei file condivisi o utilizza nomi di file univoci per richiesta per evitare conflitti. Ricordati di pulire i file non necessari per evitare di esaurire lo spazio disponibile.

## Registrazione dei log

L'interlacciamento dei log (voci di log provenienti da richieste diverse che vengono interlacciate nei log) è normale nei sistemi con più concorrenti. Le funzioni che utilizzano Lambda Managed Instances utilizzano sempre il formato di registro JSON strutturato introdotto con controlli di registrazione avanzati. [???](#) Questo formato include il `requestId`, che consente di correlare le voci di registro a una singola richiesta. Quando si utilizza il console logger, `requestId` viene automaticamente incluso in ogni voce di registro. Per ulteriori informazioni, vedere [the section called “Utilizzo dei controlli di registrazione avanzati Lambda con Node.js”](#).

Le librerie di registrazione di terze parti più diffuse, come [Winston](#), in genere includono il supporto per l'utilizzo della console per l'output dei log.

### Contesto della richiesta

L'utilizzo `context.awsRequestId` fornisce un accesso asincrono sicuro all'ID della richiesta corrente.

Utilizzare `context.xRayTraceId` per accedere all'ID di traccia. X-Ray Ciò fornisce un accesso sicuro in caso di concorrenza all'ID di traccia per la richiesta corrente. Lambda non supporta la variabile di `_X_AMZN_TRACE_ID` ambiente con Lambda Managed Instances. L'ID di X-Ray traccia viene propagato automaticamente quando si utilizza l'SDK. AWS

Utilizzato `context.getRemainingTimeInMillis()` per rilevare i timeout. Per ulteriori informazioni, consulta [the section called “Gestione e ripristino degli errori”](#).

### Esempio: gestione dei timeout

Controlla il tempo rimanente prima di ogni unità di lavoro e interrompi l'elaborazione prima che scada il timeout. Configura in `BUFFER_MS` base alla durata prevista della prossima parte di lavoro.

```
const BUFFER_MS = 2000; // Configure based on your next chunk of work

exports.handler = async (event, context) => {
  for (const item of event.items) {
    if (context.getRemainingTimeInMillis() < BUFFER_MS)
      return { statusCode: 206, body: "Timeout approaching, stopping early" };
    await processItem(item);
  }
  return { statusCode: 200, body: "Done" };
};
```

### Esempio: propagazione delle scadenze alle chiamate a valle

Quando effettui chiamate ai servizi downstream, propaga il tempo rimanente sotto forma di timeout per evitare il blocco delle chiamate di rete che sopravviverebbero alla chiamata:

```
const { S3Client, GetObjectCommand } = require("@aws-sdk/client-s3");
const client = new S3Client({});

exports.handler = async (event, context) => {
  const timeout = Math.max(1000, context.getRemainingTimeInMillis() - 500);
```

```
const response = await client.send(  
  new GetObjectCommand({ Bucket: "my-bucket", Key: "my-key" }),  
  { abortSignal: AbortSignal.timeout(timeout) }  
);  
return { statusCode: 200, body: "Done" };  
};
```

## Inizializzazione e arresto

L'inizializzazione della funzione avviene una volta per thread di lavoro. Potresti visualizzare voci di registro ripetute se la tua funzione emette log durante l'inizializzazione.

Per le funzioni Lambda con estensioni, l'ambiente di esecuzione emette un segnale SIGTERM durante l'arresto. Questo segnale viene utilizzato dalle estensioni per attivare attività di pulizia, come lo svuotamento dei buffer. Le funzioni Lambda (predefinite) con estensioni possono anche sottoscrivere il segnale SIGTERM utilizzando `process.on()`. Questo non è supportato per le funzioni che utilizzano Lambda Managed Instances poiché `process.on()` non può essere utilizzato con thread di lavoro. Per ulteriori informazioni sul ciclo di vita dell'ambiente di esecuzione, consulta [the section called “Ambiente di esecuzione”](#).

## Versioni di dipendenza

Lambda Managed Instances richiede le seguenti versioni minime del pacchetto:

- AWS SDK per JavaScript v3: versione 3.933.0 o successiva
- AWS X-Ray SDK per: versione 3.12.0 o successiva Node.js
- AWS Distro for OpenTelemetry - Instrumentation per: versione 0.8.0 o successiva JavaScript
- Powertools for AWS Lambda (): versione 2.29.0 o successiva TypeScript

## Powertools per AWS Lambda () TypeScript

Powertools for AWS Lambda (TypeScript) è compatibile con Lambda Managed Instances e fornisce utilità per la registrazione, la traccia, le metriche e altro ancora. Per ulteriori informazioni, vedere Powertools for Lambda (). [AWS TypeScript](#)

## Fasi successive

- Esamina il runtime [Java per le istanze gestite Lambda](#)
- Esamina il runtime di [Python per le istanze gestite Lambda](#)

- Esamina il runtime [.NET per le istanze gestite Lambda](#)
- Scopri come [scalare le istanze gestite Lambda](#)

## Runtime Python per istanze gestite Lambda

Il runtime Lambda utilizza più processi Python per gestire richieste simultanee. Ogni richiesta simultanea viene eseguita in un processo separato con il proprio spazio di memoria e inizializzazione. Ogni processo gestisce una richiesta alla volta, in modo sincrono. I processi non condividono direttamente la memoria, quindi le variabili globali, le cache a livello di modulo e gli oggetti singleton sono isolati tra le richieste simultanee.

### Configurazione della concorrenza

Il numero massimo di richieste simultanee che Lambda invia a ciascun ambiente di esecuzione è controllato dall'`PerExecutionEnvironmentMaxConcurrency` impostazione nella configurazione della funzione. Questa è un'impostazione opzionale e il valore predefinito varia a seconda del runtime. Per i runtime di Python, l'impostazione predefinita è 16 richieste simultanee per vCPU, oppure è possibile configurare un valore personalizzato. Questo valore determina anche il numero di processi utilizzati dal runtime Python. Lambda regola automaticamente il numero di richieste simultanee fino al valore massimo configurato in base alla capacità di ciascun ambiente di esecuzione di assorbire tali richieste.

#### Important

L'uso della concorrenza basata sui processi significa che ogni processo di runtime worker esegue la propria inizializzazione. L'utilizzo totale della memoria è uguale alla memoria per processo moltiplicata per il numero di processi concorrenti. Se si caricano librerie o set di dati di grandi dimensioni e la concorrenza è elevata, si dispone di un ingombro di memoria elevato. In base al carico di lavoro, potrebbe essere necessario regolare il CPU-to-memory rapporto o utilizzare un'impostazione di concorrenza inferiore per evitare di superare la memoria disponibile. Puoi utilizzare la `MemoryUtilization` metrica in per tenere traccia del consumo CloudWatch di memoria.

### Creazione di funzioni per la concorrenza multipla

Grazie al modello multi-concorrenza basato sui processi, le funzioni di Lambda Managed Instances che utilizzano i runtime Python non accedono contemporaneamente alle risorse in memoria da



più richiami. Non è necessario applicare pratiche di codifica per la sicurezza della concorrenza in memoria.

## Directory /tmp condivisa

La /tmp directory è condivisa tra tutte le richieste simultanee nell'ambiente di esecuzione. Le scritture simultanee sullo stesso file possono causare il danneggiamento dei dati, ad esempio se un altro processo sovrascrive il file. Per risolvere questo problema, implementa il blocco dei file condivisi o utilizza nomi di file univoci per processo o per richiesta per evitare conflitti. Ricordati di pulire i file non necessari per evitare di esaurire lo spazio disponibile.

## Registrazione dei log

L'interlacciamento dei log (voci di log provenienti da richieste diverse che vengono interlacciate nei log) è normale nei sistemi con più concorrenti.

Le funzioni che utilizzano Lambda Managed Instances utilizzano sempre il formato di registro JSON strutturato introdotto con controlli di registrazione avanzati. [???](#) Questo formato include il `requestId`, che consente di correlare le voci di registro a una singola richiesta. Quando si utilizza il `logging` modulo della libreria standard Python in Lambda, `requestId` viene automaticamente incluso in ogni voce di registro. Per ulteriori informazioni, consulta [Utilizzo dei controlli di registrazione avanzati Lambda con Python.](#)

## Contesto della richiesta

`context.aws_request_id` Da utilizzare per accedere all'ID della richiesta corrente.

Con Python runtimes, puoi utilizzare la variabile di `_X_AMZN_TRACE_ID` ambiente per accedere all'ID di X-Ray traccia con Lambda Managed Instances. L'ID di X-Ray traccia viene propagato automaticamente quando si utilizza l'SDK. AWS

Utilizzato `context.get_remaining_time_in_millis()` per rilevare i timeout. Per ulteriori informazioni, consulta [the section called “Gestione e ripristino degli errori”](#).

## Esempio: gestione dei timeout

Controlla il tempo rimanente prima di ogni unità di lavoro e interrompi l'elaborazione prima che scada il timeout. Configura in `BUFFER_MS` base alla durata prevista della prossima parte di lavoro.

```
BUFFER_MS = 2000 # Configure based on your next chunk of work
```

```
def handler(event, context):
    for item in event["items"]:
        if context.get_remaining_time_in_millis() < BUFFER_MS:
            return {"statusCode": 206, "body": "Timeout approaching, stopping early"}
        process_item(item)
    return {"statusCode": 200, "body": "Done"}
```

## Esempio: propagazione delle scadenze alle chiamate a valle

Quando effettui chiamate ai servizi downstream, propaga il tempo rimanente sotto forma di timeout per evitare il blocco delle chiamate di rete che potrebbero durare più a lungo della chiamata. L'SDK boto3 non supporta i timeout per richiesta su un client esistente, quindi è necessario creare un client con il timeout desiderato. Per le funzioni ad alto rendimento, valuta se un timeout fisso configurato al momento dell'inizializzazione è più appropriato rispetto alla creazione del client per richiesta.

```
import boto3
from botocore.config import Config

def handler(event, context):
    remaining = context.get_remaining_time_in_millis() / 1000
    timeout = max(1, remaining - 0.5)
    s3 = boto3.client("s3", config=Config(read_timeout=timeout,
    connect_timeout=timeout))
    response = s3.get_object(Bucket="my-bucket", Key="my-key")
    return {"statusCode": 200, "body": "Done"}
```

## Inizializzazione e arresto

L'inizializzazione della funzione avviene una volta per processo. È possibile che vengano visualizzate voci di registro ripetute se la funzione emette registri durante l'inizializzazione.

Per le funzioni Lambda con estensioni, l'ambiente di esecuzione emette un segnale SIGTERM durante l'arresto. Questo segnale viene utilizzato dalle estensioni per attivare attività di pulizia, come lo svuotamento dei buffer. È possibile iscriversi agli eventi SIGTERM per attivare attività di pulizia delle funzioni, come la chiusura delle connessioni al database. Per ulteriori informazioni sul ciclo di vita dell'ambiente di esecuzione, consulta [the section called “Ambiente di esecuzione”](#).

## Versioni di dipendenza

Lambda Managed Instances richiede le seguenti versioni minime del pacchetto:

- Powertools for AWS Lambda (Python): versione 3.23.0 o successiva

## Powertools per AWS Lambda (Python)

Powertools for AWS Lambda (Python) è compatibile con Lambda Managed Instances e fornisce utilità per la registrazione, la traccia, le metriche e altro ancora. Per ulteriori informazioni, consulta Powertools for Lambda (Python). [AWS](#)

## Fasi successive

- Esamina il runtime [Java per le istanze gestite Lambda](#)
- Esamina il [Node.js runtime per Lambda Managed Instances](#)
- Esamina il runtime [.NET per le istanze gestite Lambda](#)
- Scopri come [scalare le istanze gestite Lambda](#)

## .NET runtime per istanze gestite Lambda

Per i runtime.NET, le istanze gestite Lambda utilizzano un singolo processo.NET per ambiente di esecuzione. Più richieste simultanee vengono elaborate utilizzando .NET Tasks.

## Configurazione della concorrenza

Il numero massimo di richieste simultanee che Lambda invia a ciascun ambiente di esecuzione è controllato dall'impostazione nella configurazione `PerExecutionEnvironmentMaxConcurrency` della funzione. Questa è un'impostazione opzionale e il valore predefinito varia a seconda del runtime. Per i runtime.NET, l'impostazione predefinita è 32 richieste simultanee per vCPU, oppure è possibile configurare un valore personalizzato. Lambda regola automaticamente il numero di richieste simultanee fino al massimo configurato in base alla capacità di ciascun ambiente di esecuzione di assorbire tali richieste.

## Creazione di funzioni per la multiconcorrenza

È necessario applicare le stesse pratiche di sicurezza della concorrenza quando si utilizzano le istanze gestite Lambda come si farebbe in qualsiasi altro ambiente multi-concorrente. Poiché l'oggetto handler è condiviso tra tutte le attività, qualsiasi stato mutabile deve essere thread-safe. Ciò include raccolte, connessioni al database e qualsiasi oggetto statico modificato durante l'elaborazione della richiesta.

AWS I client SDK sono thread-safe e non richiedono una gestione speciale.

Esempio: pool di connessioni al database

Il codice seguente utilizza un oggetto di connessione statico al database condiviso tra richieste simultanee. L'`SqlConnection` oggetto non è thread-safe.

```
public class DBQueryHandler
{
    // Single connection shared across threads - NOT SAFE
    private SqlConnection connection;

    public DBQueryHandler()
    {
        connection = new SqlConnection("your-connection-string-here");
        connection.Open();
    }

    public string Handle(object input, ILambdaContext context)
    {
        using var cmd = connection.CreateCommand();
        cmd.CommandText = "SELECT ..."; // your query

        using var reader = cmd.ExecuteReader();

        ...
    }
}
```

Per risolvere questo problema, utilizzate una connessione separata per ogni richiesta, ricavata da un pool di connessioni. ADO.NET ad esempio, i provider supportano `Microsoft.Data.SqlClient` automaticamente il pool di connessioni quando l'oggetto di connessione viene aperto.

```
public class DBQueryHandler
{
    public DBQueryHandler()
    {
    }

    public string Handle(object input, ILambdaContext context)
    {
        using var connection = new SqlConnection("your-connection-string-here");
```

```
        connection.Open();
        using var cmd = connection.CreateCommand();
        cmd.CommandText = "SELECT ..."; // your query

        using var reader = cmd.ExecuteReader();

        ...
    }
}
```

## Esempio: collezioni

Le raccolte .NET standard non sono thread-safe:

```
public class Handler
{
    private static List<string> items = new List<string>();
    private static Dictionary<string, object> cache = new Dictionary<string, object>();

    public string FunctionHandler(object input, ILambdaContext context)
    {
        items.Add(context.AwsRequestId);
        cache["key"] = input;

        return "Success";
    }
}
```

Usa le raccolte dallo spazio dei `System.Collections.Concurrent` nomi per la sicurezza della concorrenza:

```
public class Handler
{
    private static ConcurrentBag<string> items = new ConcurrentBag<string>();
    private static ConcurrentDictionary<string, object> cache = new
    ConcurrentDictionary<string, object>();

    public string FunctionHandler(object input, ILambdaContext context)
    {
        items.Add(context.AwsRequestId);
        cache["key"] = input;

        return "Success";
    }
}
```

```
}  
}
```

## Directory /tmp condivisa

La /tmp directory è condivisa tra tutte le richieste simultanee nell'ambiente di esecuzione. Le scritture simultanee sullo stesso file possono causare il danneggiamento dei dati, ad esempio se un'altra richiesta sovrascrive il file. Per risolvere questo problema, implementate il blocco dei file per i file condivisi o utilizzate nomi di file univoci per richiesta per evitare conflitti. Ricordati di ripulire i file non necessari per evitare di esaurire lo spazio disponibile.

## Registrazione dei log

L'interlacciamento dei log (le voci di registro di diverse richieste vengono interlacciate nei log) è normale nei sistemi multi-concorrenti. [Le funzioni che utilizzano Lambda Managed Instances utilizzano sempre il formato di registro JSON strutturato introdotto con controlli di registrazione avanzati](#). Questo formato include `requestId`, che consente di correlare le voci di registro a una singola richiesta. Quando si utilizza `context.Logger` per generare registri, `requestId` viene automaticamente incluso in ogni voce di registro. Per ulteriori informazioni, vedere [the section called "Utilizzo dei controlli di registrazione avanzati di Lambda con .NET"](#).

## Contesto della richiesta

Utilizza la `context.AwsRequestId` proprietà per accedere all'ID della richiesta corrente.

Utilizzate la `context.TraceId` proprietà per accedere all'ID di X-Ray traccia. Ciò fornisce un accesso sicuro per la concorrenza all'ID di traccia per la richiesta corrente. Lambda non supporta la variabile di `_X_AMZN_TRACE_ID` ambiente con Lambda Managed Instances. L'ID di X-Ray traccia viene propagato automaticamente quando si utilizza l'SDK AWS.

Utilizzato `ILambdaContext.RemainingTime` per rilevare i timeout. Per ulteriori informazioni, consulta [the section called "Gestione e ripristino degli errori"](#).

## Esempio: gestione dei timeout

Controlla il tempo rimanente prima di ogni unità di lavoro e interrompi l'elaborazione prima che scada il timeout. Configura il buffer in base alla durata prevista della prossima parte di lavoro.

```
private static readonly TimeSpan Buffer = TimeSpan.FromMilliseconds(2000);
```

```
public APIGatewayProxyResponse FunctionHandler(APIGatewayProxyRequest request,
    ILambdaContext context)
{
    var items = JsonSerializer.Deserialize<List<string>>(request.Body);
    foreach (var item in items)
    {
        if (context.RemainingTime < Buffer)
            return new APIGatewayProxyResponse { StatusCode = 206, Body = "Timeout
approaching, stopping early" };
        ProcessItem(item);
    }
    return new APIGatewayProxyResponse { StatusCode = 200, Body = "Done" };
}
```

### Esempio: propagazione delle scadenze alle chiamate downstream

Quando effettui chiamate ai servizi downstream, propaga il tempo rimanente come timeout per evitare di dover affrontare chiamate di rete che potrebbero durare più a lungo della chiamata. Usa a `CancellationToken` con un client condiviso anziché creare un nuovo client per chiamata:

```
using Amazon.S3;

private static readonly IAmazonS3 s3 = new AmazonS3Client();

public async Task<string> FunctionHandler(APIGatewayProxyRequest request,
    ILambdaContext context)
{
    var timeout = context.RemainingTime - TimeSpan.FromMilliseconds(500);
    if (timeout < TimeSpan.FromSeconds(1)) timeout = TimeSpan.FromSeconds(1);

    using var cts = new CancellationTokenSource(timeout);
    await s3.GetObjectAsync("my-bucket", "my-key", cts.Token);
    return "Done";
}
```

### Inizializzazione e spegnimento

L'inizializzazione della funzione avviene una volta per ambiente di esecuzione. Gli oggetti creati durante l'inizializzazione vengono condivisi tra le richieste.

Per le funzioni Lambda con estensioni, l'ambiente di esecuzione emette un segnale `SIGTERM` durante lo spegnimento. Questo segnale viene utilizzato dalle estensioni per attivare attività di pulizia,

come lo svuotamento dei buffer. È possibile sottoscrivere eventi SIGTERM per attivare attività di pulizia delle funzioni, come la chiusura delle connessioni al database. Per ulteriori informazioni sul ciclo di vita dell'ambiente di esecuzione, consulta [the section called “Ambiente di esecuzione”](#).

## Versioni di dipendenza

Lambda Managed Instances richiede le seguenti versioni minime del pacchetto:

- Amazon.Lambda.Core: versione 2.7.1 o successiva
- Amazon.Lambda.RuntimeSupport: versione 1.14.1 o successiva
- OpenTelemetry.Instrumentation.AWSLambda: versione 1.14.0 o successiva
- AWSXRayRecorder.Core: versione 2.16.0 o successiva
- AWSSDK.Core: versione 4.0.0.32 o successiva

## Utensili elettrici per AWS Lambda (.NET)

[Powertools for AWS Lambda \(.NET\)](#) [AWS e Distro OpenTelemetry for - Instrumentation for DotNet](#) attualmente non supportano le istanze gestite Lambda.

## Fasi successive

- Esamina il [runtime Java per le istanze gestite Lambda](#)
- Esamina [il Node.js runtime per le istanze gestite Lambda](#)
- [Esamina il runtime Python per le istanze gestite Lambda](#)
- Scopri come [scalare le istanze gestite Lambda](#)

## Supporto Rust per le istanze gestite Lambda

### Configurazione della concorrenza

Il numero massimo di richieste simultanee che Lambda invia a ciascun ambiente di esecuzione è controllato dall'`PerExecutionEnvironmentMaxConcurrency` impostazione nella configurazione della funzione. Questa è un'impostazione opzionale e il valore predefinito per Rust è 8 richieste simultanee per vCPU, oppure puoi configurare il tuo valore. Questo valore determina il numero di task Tokio generati dal runtime ed è statico per tutta la durata dell'ambiente di esecuzione. Ogni lavoratore gestisce esattamente una richiesta in corso alla volta, senza multiplexing per lavoratore. Lambda



regola automaticamente il numero di richieste simultanee fino al numero massimo configurato in base alla capacità di ciascun ambiente di esecuzione di assorbire tali richieste.

## Creazione di funzioni per la multi-concorrenza

È necessario applicare le stesse pratiche di sicurezza dei thread quando si utilizzano le istanze gestite Lambda come si farebbe in qualsiasi altro ambiente multithread. Poiché l'oggetto handler è condiviso su tutti i thread di lavoro, qualsiasi stato mutabile deve essere thread-safe. Ciò include raccolte, connessioni al database e qualsiasi oggetto statico modificato durante l'elaborazione delle richieste.

Per abilitare la gestione simultanea delle richieste, aggiungi il flag di `concurrency-tokio` funzionalità al tuo `Cargo.toml` file.

```
[dependencies]
lambda_runtime = { version = "1", features = ["concurrency-tokio"] }
```

Il punto di `lambda_runtime::run_concurrent(...)` ingresso deve essere chiamato dall'interno di un runtime di Tokio, in genere fornito dall'#[tokio::main] attributo sulla funzione principale. La chiusura del gestore deve implementare [Clone](#) + `Send`. <https://doc.rust-lang.org/std/marker/trait.Send.html> Ciò consente al framework di condividere il gestore tra più attività asincrone in modo sicuro. Se questi limiti non vengono soddisfatti, il codice non verrà compilato.

Quando hai bisogno di uno stato condiviso tra le invocazioni (un pool di database, una struttura di configurazione), racchiudilo in [Arc](#) e clonalo in ogni chiamata. `Arc`

Tutti i client AWS SDK per Rust sono conformi alla concorrenza e non richiedono una gestione speciale.

## Esempio: AWS Client SDK

L'esempio seguente utilizza un client S3 per caricare un oggetto su ogni chiamata. Il client viene clonato direttamente nella chiusura senza `Arc`

```
let config = aws_config::load_defaults(BehaviorVersion::latest()).await;
let s3_client = aws_sdk_s3::Client::new(&config);

run_concurrent(service_fn(move |event: LambdaEvent<Request>| {
    let s3_client = s3_client.clone(); // cheap clone, no Arc needed
```

```

    async move {
        s3_client.put_object()
            .bucket(&event.payload.bucket)
            .key(&event.payload.key)
            .body(event.payload.body.into_bytes().into())
            .send()
            .await?;
        Ok(Response { message: "uploaded".into() })
    }
}))
.await

```

## Esempio: pool di connessioni al database

Quando il gestore ha bisogno di accedere a uno stato condiviso, ad esempio un client e una configurazione, inseriscilo in [Arc](#) e clonali Arc in ogni chiamata:

```

#[derive(Debug)]
struct AppState {
    dynamodb_client: DynamoDbClient,
    table_name: String,
    cache_ttl: Duration,
}

let config = aws_config::load_defaults(BehaviorVersion::latest()).await;
let state = Arc::new(AppState {
    dynamodb_client: DynamoDbClient::new(&config),
    table_name: std::env::var("TABLE_NAME").expect("TABLE_NAME must be set"),
    cache_ttl: Duration::from_secs(300),
});

run_concurrent(service_fn(move |event: LambdaEvent<Request>| {
    let state = state.clone();
    async move { handle(event, state).await }
}))
.await

```

## Cartella condivisa /tmp

La /tmp directory è condivisa tra tutte le chiamate simultanee nello stesso ambiente di esecuzione. Utilizza nomi di file univoci per chiamata (ad esempio, includi l'ID della richiesta) o implementa il blocco esplicito dei file per evitare il danneggiamento dei dati.

## Registrazione dei log

L'interlacciamento dei log (voci di log provenienti da richieste diverse che vengono interlacciate nei log) è normale nei sistemi con più concorrenti. Le funzioni che utilizzano Lambda Managed Instances supportano il formato di log JSON strutturato tramite i controlli di registrazione avanzati di Lambda.

??? Questo formato include il `requestId`, che consente di correlare le voci di registro a una singola richiesta. Per ulteriori informazioni, vedere [the section called “Registrazione avanzata con la cassa Tracing”](#).

## Contesto della richiesta

L'`Context` oggetto viene passato direttamente a ogni chiamata del gestore.

`event.context.request_id` Da utilizzare per accedere all'ID della richiesta corrente.

`event.context.xray_trace_id` Usare per accedere all'ID di X-Ray tracciamento. Lambda non supporta la variabile di `_X_AMZN_TRACE_ID` ambiente con Lambda Managed Instances. L'ID di X-Ray traccia viene propagato automaticamente quando si utilizza l'SDK per Rust. AWS

Utilizzato `event.context.deadline` per rilevare i timeout: contiene la scadenza di richiamo in millisecondi.

## Inizializzazione e spegnimento

L'inizializzazione della funzione avviene una volta per ambiente di esecuzione. Gli oggetti creati durante l'inizializzazione vengono condivisi tra le richieste.

Per le funzioni Lambda con estensioni, l'ambiente di esecuzione emette un segnale `SIGTERM` durante l'arresto. Questo segnale viene utilizzato dalle estensioni per attivare attività di pulizia, come lo svuotamento dei buffer. `lambda_runtime` offre un aiuto per semplificare la configurazione di una corretta gestione del segnale di spegnimento, `spawn_graceful_shutdown_handler()`. [https://docs.rs/lambda\\_runtime/latest/lambda\\_runtime/fn.spawn\\_graceful\\_shutdown\\_handler.html](https://docs.rs/lambda_runtime/latest/lambda_runtime/fn.spawn_graceful_shutdown_handler.html) Per ulteriori informazioni sul ciclo di vita dell'ambiente di esecuzione, consulta [the section called “Ambiente di esecuzione”](#).

## Versioni di dipendenza

Lambda Managed Instances richiede la seguente versione minima del pacchetto:

- `lambda_runtime`: versione 1.1.1 o successiva, con la funzionalità abilitata `concurrency-tokio`
- La versione minima supportata di Rust (MSRV) è 1.84.0.

## Rete per istanze gestite Lambda

Quando si eseguono le funzioni Lambda Managed Instances, è necessario configurare la connettività di rete per consentire alle funzioni di accedere alle risorse esterne al VPC. Ciò include AWS servizi come Amazon S3 e DynamoDB. La connettività è necessaria anche per trasmettere dati di telemetria a CloudWatch Logs e X-Ray.

### Opzioni di connettività

Esistono tre approcci principali per configurare la connettività VPC, ciascuno con diversi compromessi in termini di costi, sicurezza e complessità.

### Sottorete pubblica con un gateway Internet

Questa opzione utilizza una sottorete pubblica con accesso diretto a Internet tramite un gateway Internet. È possibile scegliere tra IPv4 e IPv6 configurazioni.

#### IPv4 con gateway internet

Per configurare la IPv4 connettività con un gateway Internet

1. Crea o usa una sottorete pubblica esistente con un blocco IPv4 CIDR.
2. Collegare un Internet Gateway al VPC.
3. Aggiorna la tabella delle rotte per indirizzare il `0.0.0.0/0` traffico verso il gateway Internet.
4. Assicurati che alle risorse siano assegnati IPv4 indirizzi pubblici o indirizzi IP elastici.
5. Configura i gruppi di sicurezza per consentire il traffico in uscita sulle porte richieste.

Questa configurazione fornisce una connettività bidirezionale, che consente sia le connessioni in uscita dalle tue funzioni sia le connessioni in entrata da Internet.

#### IPv6 con gateway internet

Per configurare la IPv6 connettività con un gateway Internet

1. Abilita IPv6 sul tuo VPC.
2. Crea o usa una sottorete pubblica esistente con un blocco IPv6 CIDR assegnato.
3. Collega un gateway Internet al tuo VPC (lo stesso gateway Internet può gestire entrambi IPv4 e IPv6).

4. Aggiorna la tabella delle rotte per indirizzare il : : /0 traffico verso il gateway Internet.
5. Verifica che i AWS servizi di cui hai bisogno per accedere all'assistenza IPv6 nella tua regione.
6. Configura i gruppi di sicurezza per consentire il traffico in uscita sulle porte richieste.

Questa configurazione fornisce una connettività bidirezionale tramite l'indirizzamento. IPv6

IPv6 con gateway Internet solo in uscita

Per configurare la IPv6 connettività con un gateway Internet solo in uscita

1. Abilita IPv6 sul tuo VPC.
2. Crea o usa una sottorete pubblica esistente con un blocco IPv6 CIDR assegnato.
3. Collega un gateway Internet solo in uscita al tuo VPC.
4. Aggiorna la tabella delle rotte per indirizzare il : : /0 traffico verso il gateway Internet solo in uscita.
5. Verifica che i AWS servizi di cui hai bisogno per accedere all'assistenza IPv6 nella tua regione.
6. Configura i gruppi di sicurezza per consentire il traffico in uscita sulle porte richieste.

Questa configurazione fornisce connettività solo in uscita, impedendo le connessioni in entrata da Internet e consentendo alle funzioni di avviare connessioni in uscita.

## Endpoint VPC

Gli endpoint VPC ti consentono di connettere privatamente il tuo VPC ai AWS servizi supportati senza richiedere un gateway Internet, un dispositivo NAT, una connessione VPN o una connessione Direct Connect. AWS Il traffico tra il tuo VPC e il AWS servizio non esce dalla rete Amazon.

Per configurare gli endpoint VPC

1. [Apri la console Amazon VPC all'indirizzo console.aws.amazon.com/vpc/](https://console.aws.amazon.com/vpc/).
2. Nel pannello di navigazione, seleziona Endpoints (Endpoint).
3. Seleziona Crea endpoint.
4. Per Service category (Categoria servizio), scegli AWS services.
5. Per Nome del servizio, seleziona l'endpoint del servizio di cui hai bisogno (ad esempio, `com.amazonaws.region.s3` per Amazon S3).
6. In VPC, seleziona il tuo VPC.

7. Per Sottoreti, seleziona le sottoreti in cui desideri creare interfacce di rete per endpoint. Per un'elevata disponibilità, seleziona le sottoreti in più zone di disponibilità.
8. Per Security groups (Gruppi di sicurezza), seleziona i gruppi di sicurezza da associare alle interfacce di rete dell'endpoint. I gruppi di sicurezza devono consentire il traffico in entrata dal gruppo di sicurezza della funzione sulle porte richieste.
9. Seleziona Crea endpoint.

Ripeti questi passaggi per ogni AWS servizio a cui le tue funzioni devono accedere.

## Sottorete privata con gateway NAT

Questa opzione utilizza un gateway NAT per fornire l'accesso a Internet alle risorse nelle sottoreti private, mantenendo le risorse private.

Per configurare una sottorete privata con gateway NAT

1. Crea una sottorete pubblica (se non ne esiste già una) con un blocco CIDR.
2. Collegare un Internet Gateway al VPC.
3. Crea un gateway NAT nella sottorete pubblica e assegna un indirizzo IP elastico.
4. Aggiorna la tabella delle rotte della sottorete pubblica per aggiungere una route: `0.0.0.0/0` → internet gateway.
5. Crea o usa una sottorete privata esistente con un blocco CIDR.
6. Aggiorna la tabella di routing della sottorete privata per aggiungere una rotta: `0.0.0.0/0` → gateway NAT.
7. Configura i gruppi di sicurezza per consentire il traffico in uscita sulle porte richieste.

Per un'elevata disponibilità, implementa un gateway NAT in ogni zona di disponibilità e configura le tabelle di routing per zona di disponibilità per utilizzare il gateway NAT locale. In questo modo si evitano costi di trasferimento dati Cross-AZ e si migliora la resilienza.

## Scelta di un'opzione di connettività

Quando scegli un'opzione di connettività, considera i seguenti fattori:

Sottorete pubblica con gateway Internet

- Configurazione più semplice con il costo più basso

- Adatto per ambienti di sviluppo e test
- Le risorse possono ricevere connessioni in entrata da Internet (considerazioni relative alla sicurezza)
- Supporta sia IPv4 e IPv6

### Endpoint VPC

- Massima sicurezza, il traffico rimane all'interno della AWS rete
- Latenza inferiore rispetto al routing Internet
- Consigliato per ambienti di produzione con requisiti di sicurezza rigorosi
- Costo più elevato per endpoint, per zona di disponibilità e per GB di elaborazione
- Richiede un endpoint in ogni zona di disponibilità per un'elevata disponibilità

### Sottorete privata con gateway NAT

- Le risorse rimangono private senza accesso a Internet in entrata
- Modello di architettura aziendale standard
- Supporta tutto il traffico IPv4 Internet
- Costo moderato con tariffe orarie per il gateway NAT e per l'elaborazione dei dati
- Supporta solo IPv4

### Fasi successive

- Scopri i [provider di capacità per le istanze gestite Lambda](#)
- Comprendi la [scalabilità per le istanze gestite Lambda](#)
- [Consulta le guide specifiche di runtime per Java, Node.js e Python](#)
- Comprendi [la sicurezza e le autorizzazioni per le istanze gestite Lambda](#)

## Monitoraggio delle istanze gestite Lambda

Lambda fornisce due tipi di dati di monitoraggio per i fornitori di capacità: metriche e log di sistema. Insieme, offrono visibilità sull'utilizzo delle risorse, sugli eventi del ciclo di vita delle istanze e sui problemi operativi.

- **Metriche:** Lambda pubblica automaticamente le CloudWatch metriche per i fornitori di capacità per aiutarti a monitorare l'utilizzo delle risorse, tenere traccia dei costi e ottimizzare le prestazioni. Per ulteriori informazioni, consulta [CloudWatch metriche per le istanze gestite Lambda](#).
- **Log di sistema:** Lambda invia automaticamente i log di sistema ai CloudWatch log che registrano gli eventi e gli errori del ciclo di vita delle istanze per il fornitore di capacità. Per ulteriori informazioni, consulta [Log di sistema del fornitore di capacità per le istanze gestite Lambda](#).

## CloudWatch metriche per le istanze gestite Lambda

Puoi monitorare le istanze gestite Lambda utilizzando le metriche. CloudWatch Lambda pubblica automaticamente le metriche per aiutarti CloudWatch a monitorare l'utilizzo delle risorse, tenere traccia dei costi e ottimizzare le prestazioni.

### Metriche disponibili

Lambda Managed Instances fornisce metriche a due livelli: livello di fornitore di capacità e livello di ambiente di esecuzione.

### Metriche a livello di fornitore di capacità

Le metriche a livello di fornitore di capacità forniscono visibilità sull'utilizzo complessivo delle risorse nelle tue istanze. Queste metriche utilizzano le seguenti dimensioni:

- **CapacityProviderName-** Il nome del tuo fornitore di capacità
- **InstanceType-** Il tipo di istanza EC2

Sono disponibili le seguenti metriche di utilizzo delle risorse:

- **Utilizzo della CPU:** percentuale di utilizzo della CPU tra le istanze del fornitore di capacità
- **MemoryUtilization-** La percentuale di utilizzo della memoria tra le istanze del fornitore di capacità

Sono disponibili le seguenti metriche di capacità:

- **vCPUsAvailable:** la quantità di vCPU disponibile sulle istanze per l'allocazione (in numero)
- **MemoryAvailable-** La quantità di memoria disponibile sulle istanze per l'allocazione (in byte)
- **vCPUAllocated:** la quantità di vCPU allocata sulle istanze per gli ambienti di esecuzione (in numero)



- **MemoryAllocated**- La quantità di memoria allocata sulle istanze per gli ambienti di esecuzione (in byte)

## Metriche a livello di ambiente di esecuzione

Le metriche a livello di ambiente di esecuzione forniscono visibilità sull'utilizzo delle risorse e sulla concorrenza per le singole funzioni. Queste metriche utilizzano le seguenti dimensioni:

- **CapacityProviderName**- Il nome del tuo fornitore di capacità
- **FunctionName**- Il nome della tua funzione Lambda
- **Risorsa**: per risorsa, visualizza le metriche per una versione specifica di una funzione.

### Note

Per Lambda Managed Instances (LMI), la **Resource** dimensione supporta solo le versioni delle funzioni. Il formato è `<FunctionName>:<FunctionVersion>`.

## Metriche dell'ambiente di esecuzione disponibili:

- **ExecutionEnvironmentConcurrency**- La concorrenza massima in un periodo di campionamento di 1 minuto
- **ExecutionEnvironmentConcurrencyLimit**- Il limite massimo di concorrenza per ambiente di esecuzione
- **ExecutionEnvironmentCPUUtilization**- La percentuale di utilizzo della CPU per gli ambienti di esecuzione della funzione
- **ExecutionEnvironmentMemoryUtilization**- La percentuale di utilizzo della memoria per gli ambienti di esecuzione della funzione

## Metriche di Throttle Reason

Lambda Managed Instances emette metriche granulari sul motivo dell'accelerazione che identificano il vincolo di risorse che ha causato l'interruzione. Per ogni accelerazione, Lambda emette esattamente una delle seguenti metriche secondarie con un valore pari a 1, mentre le restanti tre vengono emesse con un valore pari a 0:

- **ConcurrencyThrottles**— L'ambiente di esecuzione ha raggiunto il limite massimo di concorrenza. Questo problema può essere risolto aumentando [ExecutionEnvironmentMaxConcurrency](#) o [ridimensionando gli ambienti di esecuzione in modo più aggressivo](#).
- **CPUThrottles**: l'ambiente di esecuzione ha esaurito le risorse CPU allocate. Questo problema può essere risolto aumentando l'allocazione [della](#) vCPU della funzione o riducendo il carico per ambiente. [ExecutionEnvironmentMaxConcurrency](#)
- **MemoryThrottles**— L'ambiente di esecuzione ha esaurito la memoria allocata. Questo problema può essere risolto aumentando l'allocazione di [memoria della funzione](#) o riducendo il carico [ExecutionEnvironmentMaxConcurrency](#) per ambiente.
- **DiskThrottles**— L'ambiente di esecuzione ha esaurito lo spazio su disco allocato. Questo problema può essere risolto aumentando lo spazio di archiviazione [temporaneo della funzione](#) o riducendo il carico [ExecutionEnvironmentMaxConcurrency](#) per ambiente.

Lambda emette sempre la metrica standard insieme a queste metriche secondarie. **Throttles** Usa queste metriche per capire perché la tua funzione è scalabile e se modificare la concorrenza massima, la memoria della funzione, l'allocazione della vCPU o la configurazione del disco. Per l'elenco completo delle metriche Lambda, consulta. [Tipi di parametri per le funzioni Lambda](#)

## Frequenza e ritenzione delle metriche

Le metriche Lambda Managed Instances vengono pubblicate a intervalli di 1 minuto e conservate per 15 mesi.

## Visualizzazione delle metriche in CloudWatch

Per visualizzare le metriche di Lambda Managed Instances nella console CloudWatch

1. Apri la CloudWatch console all'indirizzo console.aws.amazon. [com/cloudwatch/](https://console.aws.amazon.com/cloudwatch/).
2. Nel riquadro di navigazione, seleziona Parametri.
3. Nella scheda Tutte le metriche, scegli AWS/Lambda.
4. Scegli la dimensione della metrica che desideri visualizzare:
  - Per le metriche a livello di fornitore di capacità, filtra per e CapacityProviderName InstanceType
  - Per le metriche a livello di ambiente di esecuzione, filtra per CapacityProviderName FunctionName, e Risorsa
5. Seleziona le metriche che desideri monitorare.

## Utilizzo delle metriche per ottimizzare le prestazioni

Monitora l'utilizzo della CPU e della memoria per capire se le tue funzioni sono dimensionate correttamente. Un utilizzo elevato potrebbe indicare la necessità di tipi di istanze più grandi o di una maggiore allocazione della memoria funzionale. Tieni traccia delle metriche di concorrenza per comprendere il comportamento di scalabilità e identificare le potenziali limitazioni.

Monitora le metriche di capacità per verificare che siano disponibili risorse sufficienti per i tuoi carichi di lavoro. La `vCPUAvailable` e le `MemoryAvailable` metriche ti aiutano a comprendere la capacità residua nelle tue istanze.

### Fasi successive

- Scopri come scalare le istanze gestite Lambda [???](#)
- Consulta le guide specifiche per il runtime per [Java e Python](#) [Node.js](#) [???](#)
- Configura la connettività [VPC per i tuoi fornitori di capacità](#)
- Comprendi [la sicurezza e le autorizzazioni per le istanze gestite Lambda](#)

## Log di sistema del fornitore di capacità per le istanze gestite Lambda

Per aiutarti a monitorare e risolvere i problemi del tuo ambiente Lambda Managed Instances, Lambda genera automaticamente i log di sistema per i tuoi fornitori di capacità. Puoi visualizzare i log per i fornitori di capacità utilizzando la console Lambda, la console, la CloudWatch AWS Command Line Interface ()AWS CLI o l'API. CloudWatch Puoi anche indirizzare i log ad Amazon S3 e Firehose utilizzando i filtri di abbonamento CloudWatch Logs.

Se il ruolo di [operatore del fornitore di capacità](#) dispone delle autorizzazioni necessarie, Lambda acquisisce i log di sistema e li invia ad Amazon Logs, che è la CloudWatch destinazione predefinita.

### Note

I log di sistema del fornitore di capacità sono abilitati per impostazione predefinita. Lambda invia automaticamente i CloudWatch log di sistema a Logs quando si crea un fornitore di capacità.

- CloudWatch Logs è la destinazione di registrazione predefinita per i log di sistema del fornitore di capacità. CloudWatch Logs fornisce funzionalità di visualizzazione e analisi dei log in tempo reale, con supporto per la creazione di metriche e allarmi in base ai dati di registro.

- Amazon S3 è economico per lo storage a lungo termine e servizi come Athena possono essere utilizzati per analizzare i log. La latenza è in genere superiore.
- Firehose offre lo streaming gestito dei log verso varie destinazioni. Se devi inviare i log ad altri AWS servizi (ad esempio, OpenSearch Service o Redshift Data API) o piattaforme di terze parti (come Datadog, New Relic o Splunk), Firehose semplifica tale processo fornendo integrazioni predefinite.

## Configurazione delle destinazioni dei log

Lambda supporta più destinazioni per i log di sistema del fornitore di capacità. Indipendentemente dalla destinazione scelta, Lambda offre opzioni per controllare il filtraggio e la consegna dei log.

I log di sistema dei fornitori di capacità utilizzano esclusivamente il formato strutturato JSON. I log strutturati JSON offrono una migliore ricercabilità e consentono un'analisi automatica. Puoi controllare quali log Lambda invia alla destinazione prescelta configurando il livello di registro del sistema. Il filtraggio aiuta a gestire i costi di archiviazione e semplifica la ricerca delle voci di registro pertinenti durante il debug.

Per ulteriori informazioni sulla configurazione di ciascuna destinazione, consulta i seguenti argomenti:

- [Invio dei log di sistema del fornitore di capacità a CloudWatch Logs](#)
- [Invio dei log di sistema del fornitore di capacità a Firehose](#)
- [Invio dei log di sistema del provider di capacità ad Amazon S3](#)

## Configurazione dei controlli di registrazione per i fornitori di capacità

Per darti un maggiore controllo su come i log di sistema del tuo fornitore di capacità vengono acquisiti, elaborati e consumati, Lambda offre le seguenti opzioni di configurazione della registrazione:

- Livello di registro: scegli il livello di dettaglio dei log di sistema che Lambda invia a CloudWatch Logs: DEBUG,, INFO o. WARN
- Gruppo di log: scegli il gruppo di CloudWatch log Logs a cui il tuo capacity provider invia i log.

Per ulteriori informazioni sulla configurazione dei controlli di registrazione, consulta i seguenti argomenti:

- [Log-level filtraggio per i log di sistema del fornitore di capacità](#)

- [Formato del registro di sistema del fornitore di capacità](#)
- [Riferimento agli eventi del registro di sistema del fornitore di capacità](#)

## Formato del registro di sistema del fornitore di capacità

I log di sistema del fornitore di capacità utilizzano il formato JSON strutturato. I log strutturati JSON offrono una migliore ricercabilità e consentono l'analisi automatica degli eventi operativi del fornitore di capacità.

### Formato JSON per i log di sistema

Ogni voce del registro di sistema viene acquisita come oggetto JSON che contiene coppie chiave-valore con i seguenti campi di primo livello:

| Campo     | Tipo    | Description                                                                             |
|-----------|---------|-----------------------------------------------------------------------------------------|
| timestamp | Stringa | L'ora in cui è stato generato l'evento di log, in formato ISO 8601.                     |
| level     | Stringa | Il livello di registro dell'evento.                                                     |
| message   | Stringa | Una descrizione dell'evento leggibile dall'uomo.                                        |
| metadata  | Oggetto | Metadati strutturati sull'evento. Contiene campi che variano in base al tipo di evento. |

### Campi di metadati

L'oggetto metadata contiene i seguenti campi. Non tutti i campi sono presenti in ogni evento.

| Campo             | Tipo              | Description                                        |
|-------------------|-------------------|----------------------------------------------------|
| instanceId        | Stringa           | L'ID dell'istanza Amazon EC2 associato all'evento. |
| availabilityZones | Gamma di stringhe | Le zone di disponibilità associate all'evento.     |

| Campo               | Tipo              | Description                                        |
|---------------------|-------------------|----------------------------------------------------|
| instanceTypes       | Gamma di stringhe | I tipi di istanza Amazon EC2 associati all'evento. |
| capacityProviderArn | Stringa           | L'ARN del fornitore di capacità.                   |

## Esempio di voce di log

L'esempio seguente mostra una voce del registro di sistema per il corretto avvio dell'istanza:

```
{
  "timestamp": "2025-07-01T14:31:15.456Z",
  "level": "INFO",
  "message": "Successfully launched c7g.xlarge instance i-0abc123def456789 in us-east-1a",
  "metadata": {
    "instanceId": "i-0abc123def456789",
    "availabilityZones": ["us-east-1a"],
    "instanceTypes": ["c7g.xlarge"],
    "capacityProviderArn": "arn:aws:lambda:us-east-1:123456789012:capacity-provider:my-capacity-provider"
  }
}
```

## Log-level filtraggio per i log di sistema del fornitore di capacità

Lambda può filtrare i log di sistema del fornitore di capacità in modo che solo i log con un determinato livello di dettaglio o inferiore vengano inviati a CloudWatch Logs. Puoi scegliere tra i seguenti livelli di registro.

| Livello di log            | Utilizzo                                                                   |
|---------------------------|----------------------------------------------------------------------------|
| DEBUG (dettaglio massimo) | Informazioni dettagliate per il debug del sistema                          |
| INFO                      | Messaggi che registrano il normale funzionamento del fornitore di capacità |

| Livello di log          | Utilizzo                                                                                               |
|-------------------------|--------------------------------------------------------------------------------------------------------|
| WARN (dettaglio minimo) | Messaggi relativi a potenziali errori che potrebbero portare a comportamenti imprevisti se non risolti |

Quando si seleziona un livello di log, Lambda invia i log di quel livello e di livello inferiore. Ad esempio, se imposti il livello di registro di sistema di un provider di capacità su INFO, Lambda non invia output di log a livello DEBUG.

Per impostazione predefinita, Lambda imposta il livello di log del sistema su INFO. Per ricevere log più dettagliati, imposta il livello su DEBUG. Per ricevere meno registri, imposta il livello su WARN. Per visualizzare un elenco dei livelli di log a cui Lambda mappa i diversi log eventi di sistema, consulta la pagina [Riferimento agli eventi del registro di sistema del fornitore di capacità](#).

### Configurazione del livello di registro del sistema

Per configurare il livello di log di sistema per il tuo capacity provider, puoi utilizzare la console Lambda o AWS Command Line Interface (AWS CLI). Puoi anche configurare il livello di log di sistema utilizzando i comandi API `CreateCapacityProvider` e `UpdateCapacityProvider` Lambda.

Per configurare il livello di log di sistema (console)

1. Apri la pagina [dei fornitori di](#) capacità della console Lambda.
2. Scegli un fornitore di capacità.
3. Scegli Additional configuration (Configurazione aggiuntiva).
4. Nella sezione Configurazione della registrazione, scegli Modifica.
5. Seleziona il livello di registro di sistema desiderato per il tuo capacity provider.
6. Scegli Save (Salva).

Per impostare il livello di registro di sistema durante la creazione di un fornitore di capacità (AWS CLI)

- Per configurare il livello di registro di sistema quando si crea un nuovo fornitore di capacità, utilizzare l'opzione `--telemetry-config` nel comando <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/create-capacity-provider.html> `create-capacity-provider`. Imposta `SystemLogLevel` su DEBUG, INFO o WARN.

```
aws lambda create-capacity-provider \  
  --capacity-provider-name my-capacity-provider \  
  --vpc-config SubnetIds=subnet-12345,subnet-67890,SecurityGroupIds=sg-12345 \  
  --permissions-config  
CapacityProviderOperatorRoleArn=arn:aws:iam::123456789012:role/MyOperatorRole \  
  --telemetry-config LoggingConfig={SystemLogLevel=DEBUG}
```

Per aggiornare il livello di registro di sistema di un fornitore di capacità esistente (AWS CLI)

- Per modificare il livello di registro di sistema di un fornitore di capacità esistente, utilizzare il comando <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/update-capacity-provider.html> `update-capacity-provider`.

```
aws lambda update-capacity-provider \  
  --capacity-provider-name my-capacity-provider \  
  --telemetry-config LoggingConfig={SystemLogLevel=WARN}
```

Riferimento agli eventi del registro di sistema del fornitore di capacità

Lambda pubblica gli eventi del registro di sistema per le operazioni del ciclo di vita delle istanze del fornitore di capacità. La tabella seguente mostra gli eventi pubblicati da Lambda e se ogni evento viene registrato in base al livello di registro di sistema configurato.

Per ulteriori informazioni sulla configurazione del livello di registro di sistema, vedere [Log-level filtraggio per i log di sistema del fornitore di capacità](#). Per dettagli sul formato di immissione del registro, vedere [Formato del registro di sistema del fornitore di capacità](#).

Mappatura a livello di eventi del registro di sistema

| Event | Description                                                                      | DEBUG      | INFO       | WARN        |
|-------|----------------------------------------------------------------------------------|------------|------------|-------------|
| Avvio | Emesso quando Lambda avvia il lancio di un'istanza per il fornitore di capacità. | Registrato | Registrato | Non loggato |



| Event             | Description                                                                                            | DEBUG      | INFO       | WARN        |
|-------------------|--------------------------------------------------------------------------------------------------------|------------|------------|-------------|
| Lanciato          | Emesso quando un'istanza passa con successo a uno stato attivo ed è pronta per servire le invocazioni. | Registrato | Registrato | Non loggato |
| LaunchFailed      | Emesso quando un tentativo di avvio dell'istanza fallisce.                                             | Registrato | Registrato | Registrato  |
| Terminando        | Emesso quando Lambda contrassegna un'istanza per la terminazione.                                      | Registrato | Registrato | Non loggato |
| Terminato         | Emesso quando un'istanza viene terminata con successo.                                                 | Registrato | Registrato | Non loggato |
| TerminateFailed   | Emesso quando Lambda non riesce a terminare un'istanza.                                                | Registrato | Registrato | Registrato  |
| HealthCheckFailed | Emesso quando un'istanza non supera il controllo di integrità ed è contrassegnata come non integra.    | Registrato | Registrato | Registrato  |

#### Note

Questo elenco si espande man mano che Lambda aggiunge nuove funzionalità per i fornitori di capacità.

### Invio dei log di sistema del fornitore di capacità a CloudWatch Logs

Per impostazione predefinita, Lambda acquisisce automaticamente i log di sistema per il fornitore di capacità e li invia a CloudWatch Logs, a condizione che il ruolo di [operatore del fornitore di capacità disponga delle](#) autorizzazioni necessarie. Questi log sono archiviati in un gruppo di log denominato `/aws/lambda/capacity-provider/<capacity-provider-name>`. Se necessario, puoi configurare il tuo fornitore di capacità per inviare i log a un gruppo di log diverso utilizzando la console Lambda o l'AWS CLI API Lambda. Per ulteriori informazioni, consulta [Configurazione dei gruppi di CloudWatch log](#).

Puoi visualizzare i log per il tuo fornitore di capacità utilizzando la console Lambda, la CloudWatch console, la AWS Command Line Interface (AWS CLI) o l'API. CloudWatch Per ulteriori informazioni, consulta [Visualizzazione dei CloudWatch log per i fornitori di capacità](#).

## Autorizzazioni IAM richieste

Il ruolo di [operatore del tuo fornitore di capacità](#) richiede le seguenti autorizzazioni per caricare i log su Logs: CloudWatch

- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`

Per ulteriori informazioni, consulta [Using Identity-Based Policy \(politiche IAM\) for CloudWatch Logs](#) nella Amazon User Guide. CloudWatch

Puoi aggiungere queste autorizzazioni per CloudWatch i log utilizzando la policy `AWSLambdaManagedEC2ResourceOperator` AWS gestita fornita da Lambda. Per aggiungere questa policy al ruolo, esegui il seguente comando:

```
aws iam attach-role-policy --role-name your-role --policy-arn arn:aws:iam::aws:policy/AWSLambdaManagedEC2ResourceOperator
```

Per ulteriori informazioni, consulta [the section called “Ruolo di operatore”](#).

### Important

Se il ruolo dell'operatore non dispone delle autorizzazioni richieste, Lambda non fornisce i log di sistema e non restituisce un errore. Assicurati che il tuo ruolo di operatore disponga delle autorizzazioni corrette per evitare errori di consegna silenziosa dei log.

## Prezzi

Non sono previsti costi aggiuntivi per l'utilizzo dei log di sistema del fornitore di capacità; tuttavia, si applicano le tariffe standard CloudWatch dei log. Per ulteriori informazioni, consultare [Prezzi di CloudWatch](#).

## Configurazione dei gruppi di CloudWatch log

Per impostazione predefinita, Lambda crea un gruppo di log denominato in base `/aws/lambda/capacity-provider/<capacity-provider-name>` al fornitore di capacità. Per inviare i log a un gruppo di log diverso, usa la console Lambda o AWS CLI i comandi `CreateCapacityProvider` e `UpdateCapacityProvider` API.

Quando si configura un gruppo di log personalizzato, il nome deve seguire le regole di denominazione [CloudWatch dei log](#). Inoltre, i nomi dei gruppi di log personalizzati non devono iniziare con la stringa `aws/`. Se crei un gruppo di log personalizzato che inizia con `aws/`, Lambda non può creare il gruppo di log e il tuo capacity provider non invia i log a CloudWatch.

### Flussi di log

All'interno del gruppo di log, Lambda crea flussi di log utilizzando il seguente formato di denominazione:

```
<capacity-provider-name>/managed-instances
```

Ad esempio, per un fornitore di capacità denominato `my-capacity-provider`, il flusso di log è:

```
my-capacity-provider/managed-instances
```

### Configurazione di un gruppo di log personalizzato

Per modificare il gruppo di log (console) di un fornitore di capacità

1. Apri la pagina [dei fornitori di](#) capacità della console Lambda.
2. Scegli un fornitore di capacità.
3. Scegli Additional configuration (Configurazione aggiuntiva).
4. Nella sezione Configurazione della registrazione, scegli Modifica.
5. Inserisci il nome del gruppo di CloudWatch log Logs a cui desideri che il tuo capacity provider invii i log. Se si immette il nome di un gruppo di log esistente, il fornitore di capacità utilizza tale gruppo. Se non esiste alcun gruppo di log con il nome immesso, Lambda crea un nuovo gruppo di log con quel nome.
6. Scegli Save (Salva).

Per modificare il gruppo di log di un fornitore di capacità (AWS CLI)

- Per modificare il gruppo di log di un fornitore di capacità esistente, utilizzare il comando <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/update-capacity-provider.html> `update-capacity-provider`.

```
aws lambda update-capacity-provider \  
  --capacity-provider-name my-capacity-provider \  
  --telemetry-config LoggingConfig={LogGroup=myLogGroup}
```

Per specificare un gruppo di log personalizzato quando si crea un fornitore di capacità (AWS CLI)

- Per specificare un gruppo di log personalizzato quando si crea un nuovo provider di capacità, utilizzare l'opzione `--telemetry-config` nel comando <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/create-capacity-provider.html> `create-capacity-provider`.

```
aws lambda create-capacity-provider \  
  --capacity-provider-name my-capacity-provider \  
  --vpc-config SubnetIds=subnet-12345,subnet-67890,SecurityGroupIds=sg-12345 \  
  --permissions-config  
  CapacityProviderOperatorRoleArn=arn:aws:iam::123456789012:role/MyOperatorRole \  
  --telemetry-config LoggingConfig={LogGroup=myLogGroup}
```

Autorizzazioni relative al ruolo dell'operatore

Affinché il fornitore di capacità possa inviare i CloudWatch log a Logs, il ruolo di operatore deve disporre dell'PutLogEvents [autorizzazione](#) logs. Lambda non aggiunge questa autorizzazione automaticamente. Se il ruolo di operatore non l'ha già, aggiungila prima di creare il fornitore di capacità. Questa autorizzazione è inclusa nella policy `AWSLambdaManagedEC2ResourceOperator` gestita.

Per ulteriori informazioni sul ruolo dell'operatore, vedere [the section called “Ruolo di operatore”](#).

Visualizzazione dei CloudWatch log per i fornitori di capacità

Puoi visualizzare CloudWatch i log di Amazon Logs per il tuo fornitore di capacità utilizzando la console Lambda, la CloudWatch console o AWS Command Line Interface (AWS CLI). Segui le istruzioni nelle sezioni seguenti per accedere ai log del tuo fornitore di capacità.

## Registri in streaming con CloudWatch Logs Live Tail

Amazon CloudWatch Logs Live Tail ti aiuta a risolvere rapidamente i problemi dei tuoi fornitori di capacità visualizzando in streaming un elenco di nuovi eventi di registro in tempo reale. Puoi visualizzare e filtrare i log inseriti dal tuo fornitore di capacità, aiutandoti a rilevare e risolvere rapidamente i problemi.

### Note

Le sessioni Live Tail comportano costi al minuto in base al tempo di utilizzo della sessione. Per ulteriori informazioni sui prezzi, consulta [Amazon CloudWatch Pricing](#).

Per iniziare una sessione Live Tail, apri il gruppo di log del tuo capacity provider nella CloudWatch console e scegli Live Tail. Per ulteriori informazioni, consulta [Use Live Tail per visualizzare i log in tempo quasi reale](#) nella Amazon CloudWatch User Guide.

### Visualizza i log nella console Lambda

Per visualizzare i log nella console Lambda

1. Apri la pagina dei fornitori di [capacità](#) della console Lambda.
2. Scegli un fornitore di capacità.
3. Selezionare la scheda Monitor (Monitora).
4. Scegli Visualizza i log in CloudWatch per aprire il gruppo di log nella CloudWatch console.

### Visualizza i log nella CloudWatch console

Per visualizzare i log nella CloudWatch console

1. Apri la CloudWatch console all'indirizzo [console.aws.amazon.com/cloudwatch/](https://console.aws.amazon.com/cloudwatch/).
2. Nel riquadro di navigazione, scegli Log, quindi Log groups.
3. Scegli il gruppo di log per il tuo capacity provider. Per impostazione predefinita, il gruppo di log è denominato `/aws/lambda/capacity-provider/<capacity-provider-name>`.
4. Scegli un flusso di log per visualizzare gli eventi di log.

## Visualizza i log con AWS CLI

Per visualizzare i log usando il AWS CLI, usa i `aws logs` comandi. L'esempio seguente recupera gli eventi di registro più recenti per un fornitore di capacità:

```
aws logs get-log-events \  
  --log-group-name /aws/lambda/capacity-provider/my-capacity-provider \  
  --log-stream-name my-capacity-provider/managed-instances
```

È inoltre possibile utilizzare CloudWatch Logs Insights per interrogare e analizzare i log di sistema del fornitore di capacità. L'esempio seguente interroga tutti gli eventi di livello ERROR:

```
aws logs start-query \  
  --log-group-name /aws/lambda/capacity-provider/my-capacity-provider \  
  --start-time $(date -d '1 hour ago' +%s) \  
  --end-time $(date +%s) \  
  --query-string 'fields @timestamp, @message | filter level = "ERROR" | sort  
@timestamp desc'
```

## Invio dei log di sistema del fornitore di capacità a Firehose

Puoi inviare i log di sistema del fornitore di capacità ad Amazon Data Firehose. Con Firehose, puoi trasmettere i log in tempo reale verso varie destinazioni, inclusi strumenti di analisi di terze parti ed endpoint personalizzati.

### Note

È possibile configurare i log di sistema del fornitore di capacità da inviare a Firehose utilizzando AWS CLI AWS CloudFormation, e tutti gli SDK. AWS

## Prezzi

Per ulteriori informazioni sui prezzi, consulta i prezzi di [Amazon CloudWatch](#).

## Configurazione della consegna dei log di Firehose

Per inviare i log di sistema del fornitore di capacità a Firehose, create un filtro di abbonamento CloudWatch Logs nel gruppo di log del vostro fornitore di capacità. Il filtro degli abbonamenti indirizza gli eventi di registro al tuo flusso di consegna Firehose.

Per impostazione predefinita, il gruppo di log del fornitore di capacità è denominato `/aws/lambda/capacity-provider/<capacity-provider-name>`. Utilizzate questo gruppo di log durante la creazione del filtro di sottoscrizione.

Per istruzioni sulla configurazione di un filtro di sottoscrizione per Firehose, inclusa la creazione dei ruoli IAM e le autorizzazioni, consulta Filtri di [sottoscrizione con Firehose](#) nella Amazon CloudWatch User Guide.

## Cross-account registrazione

È possibile configurare il fornitore di capacità per inviare i log a un flusso di distribuzione Firehose in un altro AWS account. Ciò richiede l'impostazione di una destinazione e la configurazione delle autorizzazioni appropriate in entrambi gli account.

Per istruzioni sulla configurazione della registrazione tra più account, inclusi i ruoli e le policy IAM richiesti, consulta [Configurazione di una nuova sottoscrizione tra account](#) nella Amazon User Guide. CloudWatch

## Invio dei log di sistema del provider di capacità ad Amazon S3

Puoi inviare i log di sistema del fornitore di capacità ad Amazon Simple Storage Service. Con Amazon S3, puoi archiviare i log in modo conveniente per conservarli a lungo termine e analizzarli utilizzando servizi come Athena.

### Note

Puoi configurare i log di sistema del fornitore di capacità da inviare ad Amazon S3 utilizzando gli SDK AWS CLI AWS CloudFormation, e tutti gli SDK AWS.

## Prezzi

Per ulteriori informazioni sui prezzi, consulta i prezzi di [Amazon CloudWatch](#).

## Configurazione della distribuzione dei log di Amazon S3

Per inviare i log di sistema del provider di capacità ad Amazon S3, crea un filtro di sottoscrizione CloudWatch Logs nel gruppo di log del provider di capacità. Il filtro di sottoscrizione indirizza gli eventi di log al tuo bucket Amazon S3.

Per impostazione predefinita, il gruppo di log del fornitore di capacità è denominato `/aws/lambda/capacity-provider/<capacity-provider-name>`. Utilizzate questo gruppo di log durante la creazione del filtro di sottoscrizione.

Per istruzioni sulla configurazione di un filtro di sottoscrizione per Amazon S3, inclusa la creazione dei ruoli IAM e le autorizzazioni dei bucket, consulta [Invio di log delle funzioni Lambda ad Amazon S3](#)

## Cross-account registrazione

Puoi configurare il tuo fornitore di capacità per inviare i log a un bucket Amazon S3 in un account diverso. AWS Ciò richiede l'impostazione di una destinazione e la configurazione delle autorizzazioni appropriate in entrambi gli account.

Per istruzioni sulla configurazione della registrazione tra più account, inclusi i ruoli e le policy IAM richiesti, consulta [Configurazione di una nuova sottoscrizione tra account](#) nella Amazon User Guide. CloudWatch

## Quote delle istanze gestite Lambda

Questa pagina descrive le quote di servizio per le istanze gestite AWS Lambda. Queste quote sono separate dalle quote AWS Lambda (predefinite). Alcune quote possono essere aumentate su richiesta.

Oltre alle quote elencate in questa pagina, le istanze gestite Lambda sono soggette alle quote di [servizio Amazon EC2 applicate al tuo account](#). Le quote che di solito riguardano i clienti di Lambda Managed Instances [includono i limiti vCPU del tipo di istanza EC2, i limiti di volume di Amazon EBS e gli indirizzi IP disponibili nelle sottoreti VPC](#).

## Quote di richieste API Lambda

Queste quote controllano la velocità con cui è possibile effettuare chiamate API per gestire i provider di capacità di Lambda Managed Instances. I limiti di velocità delle API di lettura e scrittura si applicano a tutte le operazioni dei provider di capacità combinate, tra cui la creazione, l'aggiornamento, la descrizione e l'eliminazione dei provider di capacità.

| Risorsa                                                                                                   | Quota                                              |
|-----------------------------------------------------------------------------------------------------------|----------------------------------------------------|
| La velocità combinata massima (richieste al secondo) per tutte le API di lettura dei provider di capacità | 15 richieste al secondo. Non può essere aumentato. |



| Risorsa                                                                                                     | Quota                                             |
|-------------------------------------------------------------------------------------------------------------|---------------------------------------------------|
| La velocità combinata massima (richieste al secondo) per tutte le API di scrittura dei provider di capacità | 1 richiesta al secondo. Non può essere aumentato. |

## Quote di risorse per Lambda Managed Instances

Queste quote definiscono i limiti per le risorse principali di Lambda Managed Instances all'interno del tuo account. AWS Esse regolano il numero di provider di capacità che è possibile creare e il numero di versioni di funzioni che possono essere associate a ciascun provider di capacità.

| Risorsa                                          | Quota                                                                                              |
|--------------------------------------------------|----------------------------------------------------------------------------------------------------|
| fornitori di capacità                            | 1.000. Il numero massimo di fornitori di capacità creati in un account.                            |
| Versioni delle funzioni per provider di capacità | 100. Il numero massimo di versioni di funzioni per provider di capacità. Non può essere aumentato. |
| vCPU per provider di capacità                    | 15.000. Il numero massimo di vCPU per provider di capacità.                                        |

## Quote di mappatura delle sorgenti degli eventi

Queste quote controllano il throughput e i limiti di configurazione per l'elaborazione degli eventi da vari AWS servizi sulle istanze gestite Lambda. I limiti di throughput garantiscono prestazioni prevedibili, mentre i limiti del numero di mappature aiutano a mantenere la stabilità del servizio. Le mappature delle sorgenti degli eventi su Lambda Managed Instances supportano Amazon SQS, DynamoDB Streams, Amazon Kinesis Data Streams, Amazon MSK e Apache Kafka autogestito come sorgenti di eventi.

| Risorsa                                                                                               | Quota                                                                |
|-------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|
| Throughput di mappatura delle sorgenti di eventi SQS standard su istanze gestite Lambda               | 5 MB al secondo. Non può essere aumentato.                           |
| Velocità effettiva di mappatura delle sorgenti di eventi Kafka standard su istanze gestite Lambda     | 1 MB al secondo. Non può essere aumentato.                           |
| Mappature standard delle sorgenti degli eventi Kafka su istanze gestite Lambda                        | 100 mappature delle sorgenti degli eventi. Non può essere aumentato. |
| Throughput di mappatura delle sorgenti di eventi Kinesis su istanze gestite Lambda                    | 25 MB al secondo. Può essere aumentato.                              |
| Throughput di mappatura delle sorgenti di eventi DynamoDB su istanze gestite Lambda                   | 10 MB al secondo. Può essere aumentato.                              |
| Richiama la velocità effettiva delle richieste per le chiamate asincrone sulle istanze gestite Lambda | 5 MB al secondo. Può essere aumentato.                               |

## Richiedere un aumento della quota

Per le quote che possono essere aumentate, puoi richiedere un aumento tramite la console Service Quotas.

Richiesta di un aumento delle quote

1. [Apri la console Service Quotas all'indirizzo console.aws.amazon.com/servicequotas/](https://console.aws.amazon.com/servicequotas/).
2. Nel pannello di navigazione, scegliere servizi AWS .
3. Selezionare AWS Lambda.
4. Seleziona la quota che desideri aumentare.
5. Scegliere Request quota increase (Richiedi aumento di quota).
6. Inserisci il nuovo valore della quota e fornisci una giustificazione per l'aumento.
7. Scegli Richiedi.

## Fasi successive

- Scopri i [provider di capacità per le istanze gestite Lambda](#)
- Comprendi la [scalabilità per le istanze gestite Lambda](#)
- [Consulta le guide specifiche di runtime per Java e Node.jsPython](#)
- Configura la [connettività VPC per i tuoi provider](#) di capacità

## Best practice per le istanze gestite Lambda

### configurazione del provider di capacità

Separa i fornitori di capacità in base al livello di affidabilità. Crea diversi fornitori di capacità per carichi di lavoro con requisiti di sicurezza diversi. Tutte le funzioni assegnate allo stesso fornitore di capacità devono essere attendibili reciprocamente, poiché i fornitori di capacità fungono da limite di sicurezza.

Usa nomi descrittivi. Indicare i fornitori di capacità per indicare chiaramente l'uso previsto e il livello di fiducia (ad esempio `production-trusted`, `dev-sandbox`). Questo aiuta i team a comprendere lo scopo e lo stato di sicurezza di ciascun fornitore di capacità.

Utilizza più zone di disponibilità. Specifica le sottoreti su più zone di disponibilità durante la creazione dei fornitori di capacità. Per impostazione predefinita, Lambda avvia tre istanze per la resilienza AZ, garantendo un'elevata disponibilità delle funzioni.

### Selezione del tipo di istanza

Lascia che Lambda scelga i tipi di istanza. Per impostazione predefinita, Lambda sceglie i tipi di istanza migliori per il carico di lavoro. Ti consigliamo di lasciare che Lambda Managed Instances scelga i tipi di istanza per te, poiché limitare il numero di possibili tipi di istanza potrebbe comportare una minore disponibilità.

Specifica i tipi di istanza per requisiti specifici. Se hai requisiti hardware specifici, imposta i tipi di istanza consentiti su un elenco di istanze compatibili. Ad esempio:

- Per le applicazioni che richiedono un'elevata larghezza di banda di rete, seleziona diversi tipi di istanze
- Per ambienti di test o sviluppo con vincoli di costo, scegli tipi di istanze più piccoli come `m7a.large`

## Configurazione delle funzioni

Scegliete le impostazioni appropriate per la memoria e la vCPU. Seleziona le configurazioni di memoria e vCPU che supportano l'esecuzione multi-simultanea della tua funzione. La dimensione minima della funzione supportata è di 2 GB e 1 vCPU.

- Per le applicazioni Python, scegli un rapporto più elevato tra memoria e vCPU (ad esempio 4 a 1 o 8 a 1) a causa del modo in cui Python gestisce la multi-concorrenza
- Per CPU-intensive operazioni o funzioni che eseguono un I/O ridotto, scegli più di una vCPU
- Per IO-heavy applicazioni come servizi web o processi batch, la multi-concorrenza offre i maggiori vantaggi

Configura la massima concorrenza in modo appropriato. Lambda sceglie impostazioni predefinite ragionevoli per la massima concorrenza che bilanciano il consumo di risorse e la produttività.

Modifica questa impostazione in base all'utilizzo delle risorse della tua funzione:

- Aumentate la concorrenza massima (fino a 64 per vCPU) se le chiamate di funzione utilizzano pochissima CPU
- Diminuisci la concorrenza massima se l'applicazione consuma una grande quantità di memoria e pochissima CPU

Tieni presente che gli ambienti di esecuzione con una concorrenza molto bassa potrebbero presentare rallentamenti e difficoltà di scalabilità.

## Scaling configuration (Configurazione dimensionamento)

Imposta un utilizzo mirato delle risorse appropriato. Per impostazione predefinita, Lambda mantiene un margine di manovra sufficiente per consentire al traffico di raddoppiare entro 5 minuti senza rallentamenti. Modifica questa impostazione in base alle caratteristiche del carico di lavoro:

- Per carichi di lavoro molto costanti o applicazioni non sensibili alle accelerazioni, stabilisci l'obiettivo su un livello elevato per ottenere un utilizzo maggiore e costi inferiori
- Per i carichi di lavoro con potenziali interruzioni del traffico, stabilisci obiettivi di risorse su un livello basso per mantenere un margine di manovra aggiuntivo

Pianifica la crescita del traffico. Se il traffico raddoppia più che in 5 minuti, potresti riscontrare rallentamenti man mano che Lambda aumenta le istanze e gli ambienti di esecuzione. Progetta la tua applicazione in modo da gestire le potenziali limitazioni durante i rapidi periodi di scalabilità.

## Sicurezza

Applica il privilegio minimo per le autorizzazioni. PassCapacityProvider Concedi `lambda:PassCapacityProvider` le autorizzazioni solo ai fornitori di capacità necessari. Utilizza le autorizzazioni a livello di risorsa per limitare i fornitori di capacità che gli utenti possono assegnare alle funzioni.

Monitora l'utilizzo dei fornitori di capacità. Utilizzato AWS CloudTrail per monitorare le assegnazioni dei fornitori di capacità e i modelli di accesso. Questo aiuta a identificare i tentativi di accesso non autorizzati e garantisce la conformità alle politiche di sicurezza.

Separa i carichi di lavoro non attendibili. Non fare affidamento sui contenitori per l'isolamento di sicurezza tra carichi di lavoro non attendibili. Utilizza diversi fornitori di capacità per separare i carichi di lavoro che non sono reciprocamente affidabili.

## Ottimizzazione dei costi

Utilizza le opzioni di prezzo di EC2. Sfrutta i piani di risparmio e le istanze riservate di EC2 per ridurre i costi. Queste opzioni di prezzo si applicano al calcolo EC2 sottostante (la commissione di gestione del 15% non è scontata).

Ottimizza per carichi di lavoro stazionari. Le istanze gestite Lambda sono più adatte per funzioni stazionarie con traffico prevedibile ad alto volume. Per modelli di traffico intensi, Lambda (impostazione predefinita) potrebbe essere più conveniente.

Monitora l'utilizzo delle risorse. Tieni traccia delle CloudWatch metriche per comprendere l'utilizzo della CPU e della memoria. Regola l'allocazione della memoria delle funzioni e la selezione del tipo di istanza in base ai modelli di utilizzo effettivi per ottimizzare i costi.

## Monitoraggio e osservabilità

Monitora le metriche dei fornitori di capacità. Tieni traccia delle metriche a livello di fornitore di capacità, tra cui l'utilizzo della CPU e MemoryUtilization vCPUAvailable, e verifica che siano disponibili risorse sufficienti MemoryAvailable per i tuoi carichi di lavoro.

Monitora le metriche dell'ambiente di esecuzione. Tieni traccia delle metriche a livello dell'ambiente di esecuzione, tra cui `ExecutionEnvironmentConcurrency` e `ExecutionEnvironmentConcurrencyLimit` per comprendere il comportamento di scalabilità e identificare il potenziale throttling.

Imposta gli allarmi. CloudWatch Crea CloudWatch allarmi per le metriche chiave per identificare in modo proattivo i problemi:

- Elevato utilizzo della CPU o della memoria
- Bassa capacità disponibile
- Ci avviciniamo ai limiti di concorrenza

## Language-specific considerazioni

Segui le best practice specifiche della lingua. Ogni linguaggio di programmazione gestisce la multi-concorrenza in modo diverso. Consulta le guide specifiche della lingua per consigli dettagliati:

- Java: utilizza raccolte thread-safe e per lo stato specifico della richiesta `AtomicInteger` `ThreadLocal`
- Node.js: utilizza `InvokeStore` per tutti gli stati specifici della richiesta ed evita le variabili globali
- Python: utilizza nomi di file univoci `/tmp` con gli ID delle richieste e considera l'isolamento della memoria basato sui processi
- Rust: usa `run_concurrent` al posto di `run`, con la `concurrency-tokio` funzione abilitata. Il gestore deve essere `Clone +Send`.

Verifica i problemi di sicurezza e concorrenza dei thread. Prima di passare alla produzione, testate accuratamente le vostre funzioni per individuare eventuali problemi di sicurezza dei thread, condizioni di gara e corretto isolamento dello stato in caso di carico simultaneo.

## Fasi successive

- Scopri i fornitori di [capacità per le istanze gestite Lambda](#)
- Scopri la [scalabilità per le istanze gestite Lambda](#)
- Consulta le guide specifiche per il runtime per [Java e Python](#) [Node.js](#) [???](#)
- Configura la connettività [VPC per i tuoi fornitori di capacità](#)
- Monitora le istanze gestite Lambda con le metriche [CloudWatch](#)

# Risoluzione dei problemi relativi alle istanze gestite Lambda

## Problemi di limitazione e scalabilità

### Limitazione occasionale

Problema: si verificano errori di limitazione (HTTP 429) durante le normali operazioni.

Causa: le istanze gestite da Lambda potrebbero rifiutare nuove invocazioni per proteggere le invocazioni già in corso. Se i tuoi ambienti di esecuzione hanno un utilizzo costantemente elevato, le nuove chiamate potrebbero essere limitate.

Soluzione::

- Monitora le metriche di scalabilità: esamina il grafico Throttle Reasons per comprendere il motivo delle limitazioni e dei problemi di scalabilità della capacità. Nel seguente esempio, un aumento della CPU causa rallentamenti.
- Verifica la configurazione delle funzioni: assicurati che la memoria funzionale e le impostazioni della vCPU supportino le esecuzioni simultanee multiple. Aumentate la memoria funzionale o l'allocazione della vCPU, se necessario. Se `ExecutionEnvironmentVCPUUtilization` è alto, prova ad aggiungere più vCPU per funzione.

```
aws lambda update-function \  
  --function-name my-function \  
  ...  
  --memory-size 8192
```

Se `ExecutionEnvironmentMemoryUtilization` è alto, prova ad aggiungere più memoria per vCPU:

```
aws lambda update-function \  
  --function-name my-function \  
  --capacity-provider-config '{  
    "LambdaManagedInstancesCapacityProviderConfig": {  
      "ExecutionEnvironmentMemoryGiBPerVCpu": 4.0  
    }  
  }'
```

## Accelera durante lo scale-up

**Problema:** si verificano errori di limitazione (HTTP 429) quando il traffico aumenta rapidamente.

**Causa:** le istanze gestite Lambda sono scalabili in modo asincrono in base all'utilizzo delle risorse della CPU e alla saturazione di più concorrenti. Se il traffico raddoppia più che in 5 minuti, potresti riscontrare dei rallentamenti man mano che Lambda aumenta le istanze e gli ambienti di esecuzione per soddisfare la domanda.

**Soluzioni:**

- **Modifica l'utilizzo delle risorse target:** se il carico di lavoro ha modelli di traffico prevedibili, imposta un utilizzo target inferiore per mantenere un margine aggiuntivo per i picchi di traffico.

```
aws lambda create-capacity-provider \  
  --name my-capacity-provider \  
  ...  
  --capacity-provider-scaling-config '{  
    "ScalingMode": "Manual",  
    "ScalingPolicies": [  
      {  
        "PredefinedMetricType": "LambdaCapacityProviderAverageCPUUtilization",  
        "TargetValue": 30.0  
      }  
    ]  
  }'
```

- **Pre-warm capacità:** per aumentare il traffico pianificato, utilizza l'API per preriscaldare la `PutFunctionScalingConfig` capacità aggiuntiva.

```
aws lambda put-function-scaling-config \  
  --function-name my-function \  
  --qualifier 1 \  
  --function-scaling-config '{  
    "MinExecutionEnvironments": 100  
  }'
```

## Scalabilità lenta

**Problema:** le istanze richiedono molto tempo per essere ridimensionate dopo la diminuzione del traffico.



**Causa:** le istanze gestite Lambda si riducono gradualmente per mantenere la disponibilità ed evitare rapide modifiche della capacità che potrebbero influire sulle prestazioni.

**Soluzione::**

Questo è il comportamento previsto. Lambda ridimensiona le istanze in modo conservativo per garantire la stabilità. Monitora le tue CloudWatch metriche per tenere traccia del numero di istanze in esecuzione.

## Problemi di concorrenza

Ambienti di esecuzione con basse limitazioni all'esperienza di concorrenza

**Problema:** le tue funzioni subiscono limitazioni nonostante la capacità disponibile.

**Causa:** gli ambienti di esecuzione con una concorrenza massima molto bassa potrebbero avere difficoltà a scalare in modo efficace. Le istanze gestite Lambda sono progettate per applicazioni con più concorrenti.

**Soluzioni:**

- Aumenta la concorrenza massima: se le chiamate di funzione utilizzano pochissima CPU, aumenta l'impostazione di concorrenza massima fino a 64 per vCPU.

```
aws lambda update-function \  
  --function-name ordering-api-backend \  
  --capacity-provider-config '{  
    "LambdaManagedInstancesCapacityProviderConfig": {  
      "PerExecutionEnvironmentMaxConcurrency": 32  
    }  
  }'
```

- Ottimizza il codice funzione: rivedi il codice della funzione per ridurre il consumo di CPU per chiamata, consentendo una maggiore concorrenza.
- Regola la memoria della funzione e la vCPU: assicurati che la tua funzione disponga di risorse sufficienti per gestire più chiamate simultanee.

## Problemi di sicurezza dei thread (runtime Java)

**Problema:** la funzione Java produce risultati errati o presenta condizioni di gara sotto carico.

Causa: più thread eseguono il metodo handler contemporaneamente e lo stato condiviso non è thread-safe.

Soluzione::

- Usa `AtomicInteger` o per i contatori invece dei tipi `AtomicLong` primitivi
- Sostituisci con `HashMap` `ConcurrentHashMap`
- `Collections.synchronizedList()` Usare per avvolgere `ArrayList`
- Usare `ThreadLocal` per uno stato specifico della richiesta
- Accedi agli ID di traccia dall'oggetto `Lambda Context`, non dalle variabili di ambiente

Per una guida dettagliata, consulta la documentazione di [Java runtime for Lambda Managed Instances](#).

Problemi di isolamento dello stato (runtime) Node.js

Problema: la Node.js funzione restituisce dati provenienti da diverse richieste o presenta un danneggiamento dei dati.

Causa: le variabili globali sono condivise tra chiamate simultanee sullo stesso thread di lavoro. Quando le operazioni asincrone cedono il controllo, altre chiamate possono modificare lo stato condiviso.

Soluzione::

- Installazione e utilizzo `@aws/lambda-invoke-store` per tutti gli stati specifici della richiesta
- Sostituisci le variabili globali con `InvokeStore.set()` `InvokeStore.get()`
- Usa nomi di file univoci negli `/tmp` ID delle richieste
- Accedi agli ID di traccia utilizzando `InvokeStore.getXRayTraceId()` al posto delle variabili di ambiente

Per una guida dettagliata, consulta la documentazione relativa al [Node.js runtime per Lambda Managed Instances](#).

Conflitti tra file (runtime Python)

Problema: la tua funzione Python legge dati errati dai file in `/tmp`

**Causa:** più processi condividono la `/tmp` directory. Le scritture simultanee sullo stesso file possono causare il danneggiamento dei dati.

**Soluzione::**

- Utilizza nomi di file univoci con ID di richiesta: `/tmp/request_{context.request_id}.txt`
- Usa il blocco dei file con `fcntl.flock()` per i file condivisi
- Pulisci i file temporanei con `os.remove()` After Use

Per una guida dettagliata, consulta la documentazione di [Python runtime for Lambda Managed Instances](#).

## Problemi di prestazioni

### Elevato utilizzo della memoria

**Problema:** nelle funzioni si verifica un elevato utilizzo della memoria o si verificano errori di esaurimento della memoria.

**Causa:** ogni richiesta simultanea in Python viene eseguita in un processo separato con il proprio spazio di memoria. L'utilizzo totale della memoria equivale alla memoria per processo moltiplicata per i processi concorrenti.

**Soluzione::**

- `MemoryUtilizationMonitor` la metrica in CloudWatch
- Riduci l'`MaxConcurrency` impostazione se l'utilizzo della memoria si avvicina al limite di memoria della funzione
- Aumentate l'allocazione della memoria della funzione per supportare una maggiore concorrenza
- Ottimizza l'utilizzo della memoria caricando i dati su richiesta anziché durante l'inizializzazione

### Prestazioni incoerenti

**Problema:** le prestazioni delle funzioni variano notevolmente tra una chiamata e l'altra.

**Causa:** Lambda potrebbe selezionare diversi tipi di istanze in base alla disponibilità oppure le funzioni potrebbero essere eseguite su istanze con disponibilità di risorse variabile.

**Soluzione::**

- Specifica i tipi di istanze consentiti: se hai requisiti prestazionali specifici, configura i tipi di istanze consentiti nel tuo capacity provider per limitare i tipi di istanze che Lambda può selezionare.
- Monitora le metriche a livello di istanza: monitora `CPUUtilization` e `MemoryUtilization` a livello di fornitore di capacità per identificare i vincoli relativi alle risorse.
- Rivedi le metriche sulla capacità: verifica `vCPUsAvailable` e `MemoryAvailable` assicurati che siano disponibili risorse sufficienti sulle tue istanze.

## Problemi con i fornitori di capacità

La versione della funzione non riesce a diventare ATTIVA

Problema: la versione della funzione rimane in sospeso dopo la pubblicazione.

Causa: Lambda sta avviando le istanze gestite e avviando gli ambienti di esecuzione. Questo processo richiede tempo, soprattutto per la prima versione della funzione su un nuovo fornitore di capacità.

Soluzione::

Attendi che Lambda completi il processo di inizializzazione. Lambda avvia tre istanze di default per la resilienza AZ e avvia tre ambienti di esecuzione prima di contrassegnare la versione della funzione come ACTIVE. Questo richiede in genere diversi minuti.

Impossibile eliminare il fornitore di capacità

Problema: viene visualizzato un errore quando si tenta di eliminare un fornitore di capacità.

Causa: non è possibile eliminare un fornitore di capacità a cui sono associate versioni di funzioni.

Soluzione::

1. Identifica tutte le versioni delle funzioni utilizzando il fornitore di capacità con `ListFunctionVersionsByCapacityProviderAPI`.
2. Elimina o aggiorna tali versioni della funzione per rimuovere l'associazione del fornitore di capacità.
3. Riprova a eliminare il fornitore di capacità.

Messaggi di errore generici durante la pubblicazione delle funzioni

Problema: durante la pubblicazione delle funzioni vengono visualizzati messaggi di errore generici come «Si è verificato un errore interno durante la pubblicazione».

## Soluzione::

- Verifica le autorizzazioni IAM: assicurati di disporre dell'`lambda:PassCapacityProvider` autorizzazione per il fornitore di capacità che stai cercando di utilizzare.
- Verifica la configurazione del fornitore di capacità: conferma che il tuo fornitore di capacità sia attivo utilizzando l'`GetCapacityProviderAPI`.
- Verifica la configurazione del VPC: assicurati che le sottoreti e i gruppi di sicurezza specificati nel tuo fornitore di capacità siano configurati correttamente e accessibili.
- AWS CloudTrail Registri di controllo: rivedi CloudTrail i registri per informazioni dettagliate sugli errori relativi all'operazione non riuscita.

## Problemi di monitoraggio e osservabilità

### Metriche mancanti CloudWatch

**Problema:** non vengono visualizzate le metriche previste CloudWatch per il fornitore di capacità o le funzioni.

**Causa:** le metriche vengono pubblicate a intervalli di 1 minuto. I nuovi fornitori di capacità o le nuove funzioni potrebbero non disporre di metriche immediatamente disponibili.

## Soluzione::

Attendi almeno 5-10 minuti dopo la pubblicazione di una versione della funzione prima di aspettarti che le metriche vengano visualizzate. CloudWatch Verifica di visualizzare il namespace (`AWS/Lambda`) e le dimensioni (`CapacityProviderName,, o`) corretti. `FunctionName` `InstanceType`

### Impossibile trovare i log CloudWatch

**Problema:** la funzione viene eseguita correttamente, ma non è possibile trovare i CloudWatch log in Logs.

**Causa:** le istanze gestite Lambda vengono eseguite nel tuo VPC e richiedono la connettività di rete per inviare i log ai log. CloudWatch Senza una corretta configurazione della connettività VPC, le tue funzioni non possono raggiungere l'endpoint del CloudWatch servizio Logs.

## Soluzione::

Configura la connettività VPC per consentire alle tue funzioni di inviare i log ai log. CloudWatch Sono disponibili tre opzioni:

Opzione 1: VPC endpoint for CloudWatch Logs (consigliato per la produzione)

1. Apri la console Amazon VPC all'indirizzo [console.aws.amazon.com/vpc/](https://console.aws.amazon.com/vpc/).
2. Nel pannello di navigazione, seleziona Endpoints (Endpoint).
3. Seleziona Crea endpoint.
4. Per Service category (Categoria servizio), scegli AWS services.
5. Per Nome del servizio, seleziona `com.amazonaws.region.logs` (sostituisci `region` con la tua AWS regione).
6. Per VPC, seleziona il VPC utilizzato dal tuo fornitore di capacità.
7. Per le sottoreti, seleziona le sottoreti in cui desideri creare le interfacce di rete degli endpoint. Per un'elevata disponibilità, utilizza sottoreti in più zone di disponibilità.
8. Per i gruppi di sicurezza, seleziona i gruppi di sicurezza che consentono il traffico HTTPS in entrata (porta 443) dal gruppo di sicurezza della tua funzione.
9. Abilita il DNS privato per l'endpoint.
10. Seleziona Crea endpoint.

Opzione 2: sottorete pubblica con gateway Internet

Se il tuo fornitore di capacità utilizza sottoreti pubbliche, assicurati che:

1. Un gateway Internet è collegato al tuo VPC
2. La tabella di routing indirizza il `0.0.0.0/0` traffico verso il gateway Internet
3. I gruppi di sicurezza consentono il traffico HTTPS in uscita sulla porta 443

Opzione 3: sottorete privata con gateway NAT

Se il tuo fornitore di capacità utilizza sottoreti private, assicurati che:

1. Un gateway NAT esiste in una sottorete pubblica
2. La tabella di routing della sottorete privata indirizza il `0.0.0.0/0` traffico verso il gateway NAT
3. La tabella di routing della sottorete pubblica indirizza il `0.0.0.0/0` traffico verso un gateway Internet

#### 4. I gruppi di sicurezza consentono il traffico HTTPS in uscita sulla porta 443

Per una guida dettagliata sulle opzioni di connettività VPC, consulta Connettività [VPC per le istanze gestite Lambda](#).

Difficoltà a correlare i log delle richieste simultanee

Problema: i log di richieste diverse sono interlacciati, rendendo difficile il tracciamento delle singole richieste.

Causa: l'interlacciamento dei log è previsto e si comporta come standard nei sistemi con più concorrenti.

Soluzione::

- Usa la registrazione strutturata con formato JSON: includi l'ID della richiesta in tutte le istruzioni di registro
- Java: usa Log4j con ThreadContext per includere automaticamente l'ID della richiesta
- Node.js: utilizzare `console.log()` con la formattazione JSON e includere `InvokeStore.getRequestId()`
- Python: usa il modulo di registrazione standard con formattazione JSON e includi `context.request_id`

Per una guida dettagliata, consulta le pagine di documentazione specifiche del runtime.

### Ottenere ulteriore assistenza

Se continui a riscontrare problemi dopo aver provato queste soluzioni:

1. Rivedi CloudWatch le metriche: controlla le metriche del fornitore di capacità e dell'ambiente di esecuzione per identificare i vincoli delle risorse o i problemi di scalabilità.
2. Controlla AWS CloudTrail i log: rivedi CloudTrail i log per informazioni dettagliate sulle chiamate e sugli errori delle API.
3. Contatta l' AWS assistenza: se non riesci a risolvere il problema, contatta l' AWS assistenza con i dettagli sulla configurazione del tuo capacity provider, sulla configurazione delle funzioni e sui messaggi di errore specifici che ricevi.

## Fasi successive

- Scopri i fornitori di [capacità per le istanze gestite Lambda](#)
- Scopri la [scalabilità per le istanze gestite Lambda](#)
- Consulta le guide specifiche per il runtime per [Java e Python](#) [Node.js](#) [???](#)
- Monitora le istanze gestite da Lambda con le metriche [CloudWatch](#)
- Consulta [le best practice per le istanze gestite Lambda](#)

## Gestione delle dipendenze Lambda con i livelli

Un livello Lambda è un archivio di file .zip che può contenere codice o dati aggiuntivi. I livelli di solito contengono dipendenze dalla libreria, un [runtime personalizzato](#) o file di configurazione.

Esistono diversi motivi per cui potresti prendere in considerazione l'utilizzo dei livelli:

- Per ridurre le dimensioni dei pacchetti di implementazione. Invece di includere tutte le dipendenze delle funzioni insieme al codice della funzione nel pacchetto di implementazione, inseriscile in un livello. Ciò mantiene i pacchetti di implementazione piccoli e organizzati.
- Per separare la logica delle funzioni di base dalle dipendenze. Con i livelli, puoi aggiornare le dipendenze delle funzioni indipendentemente dal codice della funzione e viceversa. Ciò favorisce la separazione dei problemi e ti aiuta a concentrarti sulla logica funzionale.
- Per condividere le dipendenze tra più funzioni. Dopo aver creato un livello, puoi applicarlo a qualsiasi numero di funzioni del tuo account. Senza livelli, è necessario includere le stesse dipendenze in ogni singolo pacchetto di implementazione.
- Per utilizzare l'editor di codice della console Lambda. L'editor di codice è uno strumento utile per testare rapidamente aggiornamenti minori del codice funzionale. Tuttavia, non è possibile utilizzare l'editor se le dimensioni del pacchetto di distribuzione sono troppo grandi. L'uso dei livelli riduce le dimensioni del pacchetto e può sbloccare l'utilizzo dell'editor di codice.
- Per bloccare una versione SDK incorporata. Gli SDK integrati potrebbero cambiare senza preavviso man mano che vengono AWS rilasciati nuovi servizi e funzionalità. Puoi bloccare una versione dell'SDK [creando un livello Lambda](#) con la versione specifica necessaria. La funzione utilizza quindi sempre la versione nel livello, anche se la versione incorporata nel servizio cambia.

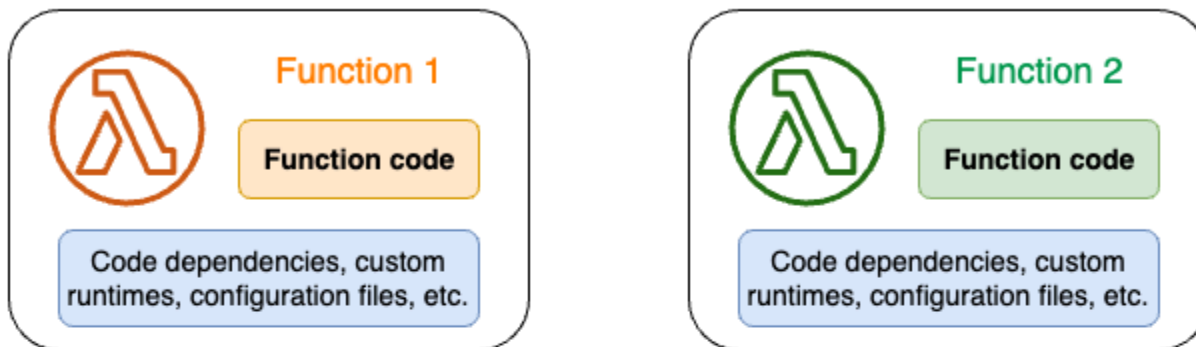
Se utilizzi le funzioni Lambda in Go o Rust, ti consigliamo di non utilizzare i livelli. Per le funzioni Go e Rust, fornisci il codice della funzione come eseguibile, che include il codice di funzione compilato



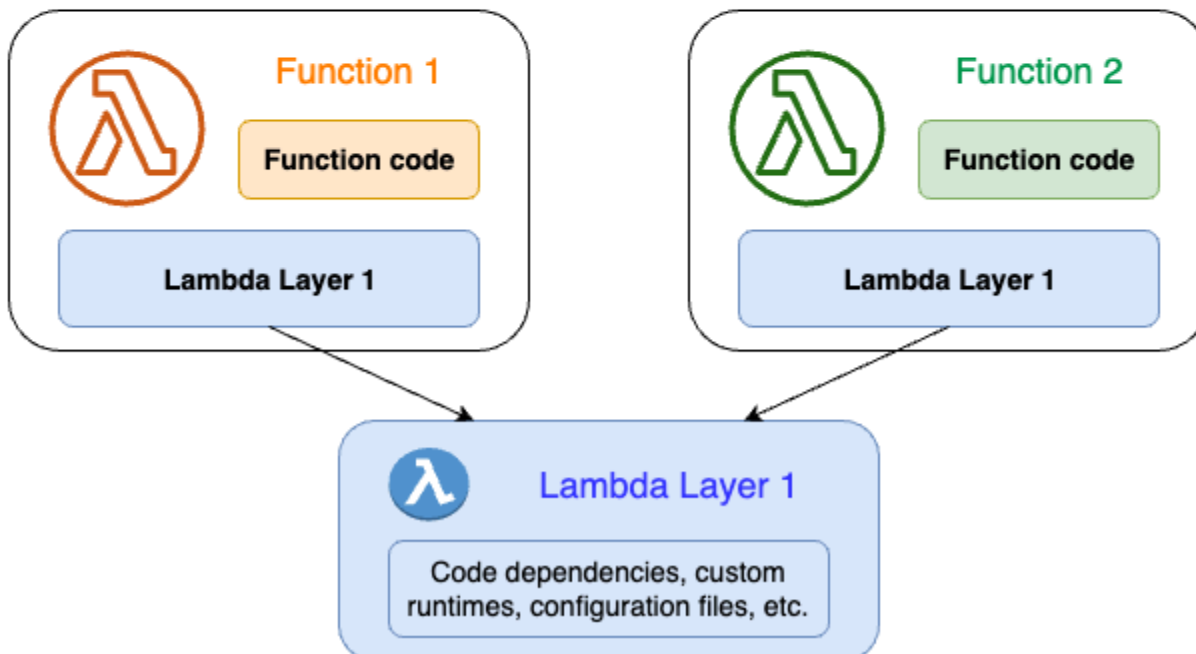
insieme a tutte le sue dipendenze. L'inserimento delle dipendenze in un livello impone alla funzione di caricare manualmente gli assembly aggiuntivi durante la fase di inizializzazione, per cui i tempi di avvio a freddo possono aumentare. Per prestazioni ottimali per le funzioni Go e Rust, includi le tue dipendenze insieme al pacchetto di implementazione.

Il diagramma seguente illustra le principali differenze di architettura tra due funzioni che condividono dipendenze. Una utilizza i livelli Lambda e l'altra no.

### Lambda function components: Without layers



### Lambda function components: With layers



Quando aggiungi un livello a una funzione, Lambda estrae il contenuto del livello nella `/opt` directory dell'ambiente di esecuzione della [funzione](#). Tutti i runtime Lambda supportati in modo nativo

includono percorsi a directory specifiche all'interno della directory `/opt`. Ciò consente alla funzione di accedere al contenuto dei livelli. Per ulteriori informazioni su questi percorsi specifici e su come creare correttamente i pacchetti per i livelli, consulta [the section called “Creazione di pacchetti dei livelli”](#).

Puoi includere fino a cinque livelli per funzione. Inoltre, è possibile utilizzare i livelli solo con funzioni Lambda [implementate come archivio di file con estensione zip](#). Per funzioni [definite come immagine del container](#), quando si crea l'immagine del container viene creato un pacchetto del runtime preferito e tutte le dipendenze del codice. Per ulteriori informazioni, consulta [Lavorare con i livelli e le estensioni Lambda nelle immagini dei contenitori](#) sul blog di Compute AWS .

## Argomenti

- [Come usare i livelli](#)
- [Livelli e versioni di livelli](#)
- [Creazione di pacchetti del contenuto dei livelli](#)
- [Creazione ed eliminazione di livelli in Lambda](#)
- [Aggiunta di livelli alle funzioni](#)
- [Utilizzo AWS CloudFormation con strati](#)
- [Utilizzo di AWS SAM con i livelli](#)

## Come usare i livelli

Per creare un livello, raccogli le tue dipendenze in un file `.zip`, in modo simile a come [crei un normale pacchetto di implementazione](#). Più specificamente, il processo generale di creazione e utilizzo dei livelli prevede questi tre passaggi:

- Innanzitutto, crea un pacchetto per il contenuto del livello. Ciò significa creare un archivio di file `.zip`. Per ulteriori informazioni, consulta [the section called “Creazione di pacchetti dei livelli”](#).
- Quindi, crea il livello in Lambda. Per ulteriori informazioni, consulta [the section called “Creazione ed eliminazione di livelli”](#).
- Aggiungi il livello alla tua funzione. Per ulteriori informazioni, consulta [the section called “Aggiunta di livelli”](#).

## Livelli e versioni di livelli

Una versione di livello è un'istantanea immutabile di una versione specifica di un livello. Quando si crea un nuovo livello, Lambda crea una nuova versione del livello con un numero di versione pari a 1. Ogni volta che si pubblica un aggiornamento del livello, Lambda incrementa il numero di versione e crea una nuova versione.

Ogni flusso è identificato in modo univoco da un nome della risorsa Amazon (ARN) univoco. Quando si aggiunge un livello alla funzione, è necessario specificare l'esatta versione del livello che si desidera utilizzare (ad esempio, `arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`).

## Creazione di pacchetti del contenuto dei livelli

Un livello Lambda è un archivio di file .zip che può contenere codice o dati aggiuntivi. I livelli di solito contengono dipendenze dalla libreria, un [runtime personalizzato](#) o file di configurazione.

In questa sezione viene descritto come creare pacchetti del contenuto dei livelli. Per ulteriori informazioni concettuali sui livelli e su come e perché utilizzarli, consulta [the section called “Livelli”](#).

Il primo passaggio per creare un livello consiste nel raggruppare tutto il contenuto del livello in un archivio di file .zip. Perché le funzioni Lambda vengano eseguite su [Amazon Linux](#), il contenuto del livello deve essere in grado di compilare e creare in un ambiente Linux.

Per garantire che il contenuto del livello funzioni correttamente in un ambiente Linux, si consiglia di creare il contenuto utilizzando uno strumento come [Docker](#).

### Argomenti

- [Percorsi dei livelli per ciascun runtime Lambda](#)

### Percorsi dei livelli per ciascun runtime Lambda

Quando si aggiunge un livello a una funzione, Lambda carica il contenuto del livello nella directory /opt di quell'ambiente di esecuzione. Per ogni runtime Lambda, la variabile PATH include percorsi di cartelle specifici nella directory /opt. Per garantire che la variabile raccolga il contenuto del livello, il file .zip del livello dovrebbe avere le sue dipendenze nei seguenti percorsi di cartella:

| Runtime | Percorso                                                            |
|---------|---------------------------------------------------------------------|
| Node.js | nodejs/node_modules                                                 |
|         | nodejs/node18/node_modules (NODE_PATH )                             |
|         | nodejs/node20/node_modules (NODE_PATH )                             |
|         | nodejs/node22/node_modules (NODE_PATH )                             |
| Python  | python                                                              |
|         | python/lib/ <i>python3.x</i> /site-packages<br>(directory del sito) |

| Runtime         | Percorso                   |
|-----------------|----------------------------|
| Java            | java/lib (CLASSPATH )      |
| Ruby            | ruby/gems/3.4.0 (GEM_PATH) |
|                 | ruby/lib (RUBYLIB)         |
|                 | bin (PATH)                 |
| Tutti i runtime | lib (LD_LIBRARY_PATH )     |

Negli esempi seguenti viene illustrato come strutturare le cartelle per l'archivio .zip del livello.

## Node.js

Example struttura di file per l'SDK AWS X-Ray per Node.js

```
xray-sdk.zip
# nodejs/node_modules/aws-xray-sdk
```

## Python

Example

```
python/                # Required top-level directory
### requests/
### boto3/
### numpy/
### (dependencies of the other packages)
```

## Ruby

Example struttura dei file per gem JSON

```
json.zip
# ruby/gems/3.4.0/
| build_info
| cache
| doc
```

```
| extensions
| gems
| # json-2.1.0
# specifications
# json-2.1.0.gemspec
```

## Java

Example struttura dei file per il file JAR Jackson

```
layer_content.zip
# java
# lib
# jackson-core-2.17.0.jar
# <other potential dependencies>
# ...
```

## All

Example struttura dei file per la libreria JQ

```
jq.zip
# bin/jq
```

Per istruzioni specifiche del linguaggio sull'impacchettamento, la creazione e l'aggiunta di un livello, consulta le pagine seguenti:

- Node.js – [the section called “Livelli”](#)
- Python – [the section called “Livelli”](#)
- Ruby – [the section called “Livelli”](#)
- Java: [the section called “Livelli”](#)

Consigliamo di non utilizzare i livelli per gestire le dipendenze per le funzioni Lambda scritte in Go. Questo perché le funzioni Lambda in Go vengono compilate in un unico eseguibile, che viene fornito a Lambda quando si distribuisce la funzione. Questo eseguibile contiene il codice di funzione compilato, insieme a tutte le sue dipendenze. L'uso dei livelli non solo complica questo processo, ma comporta anche un aumento dei tempi di avvio a freddo, poiché le funzioni devono caricare manualmente assieme aggiuntivi in memoria durante la fase di inizializzazione.

Per utilizzare dipendenze esterne con i gestori Go, includile direttamente nel pacchetto di implementazione.

## Creazione ed eliminazione di livelli in Lambda

Un livello Lambda è un archivio di file .zip che può contenere codice o dati aggiuntivi. I livelli di solito contengono dipendenze dalla libreria, un [runtime personalizzato](#) o file di configurazione.

In questa sezione viene descritto come creare ed eliminare livelli in Lambda. Per ulteriori informazioni concettuali sui livelli e su come e perché utilizzarli, consulta [the section called “Livelli”](#).

Dopo aver [impacchettato il contenuto del layer](#), il passaggio successivo è creare il layer in Lambda. In questa sezione viene descritto come creare ed eliminare i livelli utilizzando solo la console Lambda o l'API Lambda. Per creare un livello utilizzando AWS CloudFormation, consulta [the section called “Strati con CloudFormation”](#). Per creare un livello utilizzando AWS Serverless Application Model (AWS SAM), consulta [the section called “Livelli con AWS SAM”](#).

### Argomenti

- [Creazione di un livello](#)
- [Eliminazione della versione di un livello](#)

### Creazione di un livello

Per creare un livello, puoi caricare l'archivio di file con estensione .zip dal computer locale o da Amazon Simple Storage Service (Amazon S3). Quando effettui il caricamento da Amazon S3, puoi scegliere se Lambda copia il codice nello Lambda-managed storage o vi fa riferimento direttamente dal bucket S3. Per ulteriori informazioni, consulta [the section called “Self-managed Archiviazione del codice S3”](#).

Lambda estrae il contenuto del livello nella directory /opt durante la configurazione dell'ambiente di esecuzione per la funzione.

I livelli possono avere una o più [versioni](#). Quando si crea un livello, Lambda imposta la versione del livello su 1. È possibile modificare le autorizzazioni su una versione di livello esistente in qualsiasi momento. Tuttavia, per aggiornare il codice o apportare altre modifiche alla configurazione, è necessario creare una nuova versione del livello.

### Creazione di un livello (console)

1. Apri la [pagina Layers](#) (Livelli) nella console Lambda.
2. Scegli Create layer (Crea livello).
3. In Layer configuration (Configurazione livello), per Nome, immettere un nome del livello.



4. (Facoltativo) In Description (Descrizione), immetti una descrizione per il livello.
5. Per caricare il codice del livello, esegui una delle seguenti operazioni:
  - Per caricare un file con estensione .zip dal computer, scegliere Carica un file .zip. Scegli Carica per selezionare il file con estensione .zip locale.
  - Per caricare un file da Amazon S3, scegli Upload a file from Amazon S3 (Carica un file da Amazon S3). Quindi, per l'URL del link Amazon S3, immetti un collegamento al file.
6. Se hai scelto di caricare da Amazon S3, in modalità di archiviazione del codice, scegli una delle seguenti opzioni:
  - Modalità di copia (predefinita)
  - Modalità di riferimento
7. (Facoltativo) Per Architetture compatibili, scegli un valore o entrambi i valori. Per ulteriori informazioni, consulta [the section called “Set di istruzioni \(ARM/x86\)”](#).
8. (Facoltativo) Per Runtime compatibili, scegli i runtime con cui il tuo livello è compatibile.
9. (Facoltativo) Per Licenza, inserisci tutte le informazioni necessarie sulla licenza.
10. Scegli Create (Crea).

In alternativa, è possibile eseguire il comando [publish-layer-version](#) AWS Command Line Interface (CLI). Quando effettui il caricamento da Amazon S3, includi l'utilizzo dello storage S3objectStorageMode=REFERENCE di codice S3 [autogestito oppure omettilo \(impostazione predefinita\) per utilizzare lo storage](#). COPY Lambda-managed Esempio:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip --  
compatible-runtimes python3.14
```

Ogni volta che esegui publish-layer-version, viene creata una nuova [versione del livello](#).

## Eliminazione della versione di un livello

Per eliminare una versione a livelli, utilizza l'operazione API. [DeleteLayerVersion](#) Ad esempio, eseguite il [AWS CLI comando](#) delete-layer-version con il nome e la versione del livello specificati.

```
aws lambda delete-layer-version --layer-name my-layer --version-number 1
```

Quando si elimina una versione del livello, non è più possibile configurare nuove funzioni per utilizzarla. Tuttavia, qualsiasi funzione che fa già riferimento alla versione del livello eliminata continua

a essere eseguita con il contenuto di quel livello. Lambda non riutilizza mai i numeri di versione per il nome di un livello.

### Impatto sulla quota di archiviazione dell'eliminazione dei livelli

L'impatto sulla quota di archiviazione dell'eliminazione di una versione del livello dipende dalla modalità di archiviazione utilizzata dal livello e dalle funzioni che vi fanno riferimento.

#### Il livello utilizza l' Lambda-managed archiviazione (COPY)

Quando elimini una versione del layer che utilizza l' Lambda-managed archiviazione, il codice del layer viene rimosso dalla quota di Lambda-managed archiviazione del tuo account. Tuttavia, qualsiasi funzione che fa ancora riferimento alla versione del livello eliminata continua ad avere accesso al contenuto del livello. Il contenuto del livello viene conteggiato ai fini della quota di dimensione del pacchetto di distribuzione della funzione (250 MB decompressi) ma non viene più conteggiato per la quota di archiviazione a livello di account Lambda-managed .

#### Layer utilizza uno storage S3 autogestito (REFERENCE)

I livelli che utilizzano lo storage di codice S3 [autogestito](#) non consumano Lambda-managed spazio, pertanto la loro eliminazione non ha alcun impatto sulla quota di archiviazione. Lambda-managed Le funzioni che fanno riferimento a una versione di layer autogestita eliminata continuano a funzionare normalmente. Tuttavia, se elimini anche l'oggetto sorgente dal bucket S3 o revochi l'accesso di Lambda ad esso, non puoi più aggiornare la configurazione della funzione con quel livello.

#### Note

L'eliminazione di una versione del livello non disattiva mai le funzioni che vi fanno riferimento. Le funzioni rimangono invariate Active indipendentemente dal livello utilizzato Lambda-managed o dallo storage S3 autogestito. Lo Inactive stato si applica solo quando Lambda perde l'accesso al codice sorgente di una funzione archiviato in un bucket S3 autogestito.

## Aggiunta di livelli alle funzioni

Un livello Lambda è un archivio di file .zip che può contenere codice o dati aggiuntivi. I livelli di solito contengono dipendenze dalla libreria, un [runtime personalizzato](#) o file di configurazione.

In questa sezione viene spiegato come aggiungere un livello a una funzione Lambda. Per ulteriori informazioni concettuali sui livelli e su come e perché utilizzarli, consulta [the section called “Livelli”](#).

Prima di poter configurare una funzione Lambda per utilizzare un livello, è necessario:

- [Crea un pacchetto per il contenuto del livello](#)
- [Crea un livello in Lambda](#)
- Assicurati di avere l'autorizzazione per chiamare l'[GetLayerVersion](#) API sulla versione del livello. Per le funzioni incluse nella tua Account AWS, devi disporre di questa autorizzazione nella tua politica [utente](#). Per utilizzare un livello in un altro account, il proprietario di quell'account deve concedere l'autorizzazione per l'account in una [policy basata sulle risorse](#). Per alcuni esempi, consulta [the section called “Accesso ai livelli per altri account”](#).

Puoi aggiungere fino a cinque livelli a una funzione Lambda. La dimensione totale non decompressa della funzione e di tutti i livelli non può superare la quota della dimensione del pacchetto di distribuzione non compresso di 250 MB. Per ulteriori informazioni, consulta [Quote di Lambda](#).

Le tue funzioni possono continuare a utilizzare qualsiasi versione del layer che hai già aggiunto, anche dopo che la versione del layer è stata eliminata o dopo la revoca dell'autorizzazione ad accedere al layer. Non è tuttavia possibile creare una nuova funzione che utilizza la versione di un livello eliminato.

### Aggiunta di un livello a una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione.
3. Scorri verso il basso fino alla sezione Livelli, quindi scegli Aggiungi un livello.
4. In Scegli un livello, scegli un'origine del livello:
  - a. AWS livelli: scegli dall'elenco delle estensioni [AWS gestite](#).
  - b. Livelli personalizzati: scegli un livello creato nel tuo Account AWS.
  - c. Specifica un ARN: per utilizzare un livello [di un altro livello Account AWS](#), ad esempio un'estensione di [terze parti](#), inserisci Amazon Resource Name (ARN).

## 5. Scegliere Aggiungi.

L'ordine in cui si aggiungono i livelli è l'ordine in cui Lambda unisce il contenuto del livello nell'ambiente di esecuzione. Puoi modificare l'ordine di unione dei livelli utilizzando la console.

Aggiornamento dell'ordine di unione dei livelli per la tua funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione da configurare.
3. In Layers (Livelli), scegli Edit (Modifica)
4. Scegli uno dei livelli.
5. Scegli Merge earlier (Unisci prima) o Merge later (Unisci in un secondo momento) per regolare l'ordine dei livelli.
6. Scegli Save (Salva).

I livelli sono suddivisi in versioni. Il contenuto di ogni versione di livello non è modificabile. Il proprietario del livello può rilasciare nuove versioni del livello in modo da fornire contenuto appropriato. È possibile utilizzare la console per aggiornare la versione del livello collegata alle funzioni.

Aggiornamento delle versioni del livello per la funzione (console)

1. Apri la [pagina Layers](#) (Livelli) nella console Lambda.
2. Scegli il livello per il quale desideri aggiornare la versione.
3. Seleziona la scheda Funzioni che utilizzano questa versione.
4. Scegli le funzioni che desideri modificare, quindi scegli Modifica.
5. Da Versione livello, seleziona la versione del livello a cui passare.
6. Scegliere Update functions (Aggiorna funzioni).

Non è possibile aggiornare le versioni del Function Layer tra più account AWS .

## Ricerca di informazioni sul livello

Per trovare livelli nel tuo account compatibili con il runtime della tua funzione, usa l'[ListLayers](#) API. Ad esempio, puoi utilizzare il seguente comando [list-layers](#) AWS Command Line Interface (CLI):

```
aws lambda list-layers --compatible-runtime python3.14
```

Verrà visualizzato un output simile al seguente:

```
{
  "Layers": [
    {
      "LayerName": "my-layer",
      "LayerArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer",
      "LatestMatchingVersion": {
        "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:2",
        "Version": 2,
        "Description": "My layer",
        "CreateDate": "2025-04-15T00:37:46.592+0000",
        "CompatibleRuntimes": [
          "python3.14"
        ]
      }
    }
  ]
}
```

Per elencare tutti i livelli nell'account, ometti l'opzione `--compatible-runtime`. I dettagli della risposta mostrano la versione più recente di ogni livello.

È inoltre possibile ottenere la versione più recente di un layer utilizzando l'API. [ListLayerVersions](#) Ad esempio, puoi utilizzare il seguente comando `list-layer-versions` della CLI:

```
aws lambda list-layer-versions --layer-name my-layer
```

Verrà visualizzato un output simile al seguente:

```
{
  "LayerVersions": [
    {
      "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:2",
      "Version": 2,
      "Description": "My layer",
      "CreateDate": "2023-11-15T00:37:46.592+0000",

```

```
        "CompatibleRuntimes": [
            "java11"
        ],
    },
    {
        "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:1",
        "Version": 1,
        "Description": "My layer",
        "CreateDate": "2023-11-15T00:27:46.592+0000",
        "CompatibleRuntimes": [
            "java11"
        ]
    }
]
```

## Utilizzo AWS CloudFormation con strati

Puoi usare CloudFormation per creare un livello e associare il livello alla tua funzione Lambda. Nel modello di esempio seguente viene creato un livello denominato `my-lambda-layer` che viene collegato alla funzione Lambda utilizzando la proprietà `Layers` (Livelli).

In questo esempio, il modello specifica il nome della risorsa Amazon (ARN) di un [ruolo di esecuzione](#) IAM esistente. Puoi anche creare un nuovo ruolo di esecuzione nel modello utilizzando la risorsa CloudFormation [AWS::IAM::Role](#).

La funzione non necessita di autorizzazioni speciali per utilizzare i livelli.

```
---
Description: CloudFormation Template for Lambda Function with Lambda Layer
Resources:
  MyLambdaLayer:
    Type: AWS::Lambda::LayerVersion
    Properties:
      LayerName: my-lambda-layer
      Description: My Lambda Layer
      Content:
        S3Bucket: amzn-s3-demo-bucket
        S3Key: my-layer.zip
      CompatibleRuntimes:
        - python3.9
        - python3.10
        - python3.11

  MyLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: my-lambda-function
      Runtime: python3.9
      Handler: index.handler
      Timeout: 10
      Role: arn:aws:iam::111122223333:role/my_lambda_role
      Layers:
        - !Ref MyLambdaLayer
```

Puoi anche usarlo AWS Serverless Application Model per gestire i livelli. Per ulteriori informazioni, consulta [the section called “Livelli con AWS SAM”](#).

## Utilizzo di AWS SAM con i livelli

È possibile utilizzare AWS Serverless Application Model (AWS SAM) per automatizzare la creazione dei livelli nella tua applicazione. Il tipo di risorsa `AWS::Serverless::LayerVersion` crea una versione del layer a cui è possibile fare riferimento dalla configurazione della funzione Lambda.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: AWS SAM Template for Lambda Function with Lambda Layer

Resources:
  MyLambdaLayer:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: my-lambda-layer
      Description: My Lambda Layer
      ContentUri: s3://amzn-s3-demo-bucket/my-layer.zip
      CompatibleRuntimes:
        - python3.9
        - python3.10
        - python3.11

  MyLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: MyLambdaFunction
      Runtime: python3.9
      Handler: app.handler
      CodeUri: s3://amzn-s3-demo-bucket/my-function
      Layers:
        - !Ref MyLambdaLayer
```

## Aumentare le funzioni Lambda utilizzando le estensioni Lambda

È possibile utilizzare le estensioni Lambda per potenziare le funzioni Lambda. Ad esempio, utilizza le estensioni Lambda per integrare le funzioni con i tuoi strumenti di monitoraggio, osservabilità, sicurezza e governance preferiti. È possibile scegliere tra un ampio set di strumenti forniti dai [Partner AWS Lambda](#) oppure [creare estensioni Lambda personalizzate](#).

Lambda supporta estensioni interne ed esterne. Un'estensione esterna viene eseguita come processo indipendente nell'ambiente di esecuzione e continua a funzionare dopo che l'invocazione



della funzione è completamente elaborata. Poiché le estensioni vengono eseguite come processi separati, è possibile scriverle in un linguaggio diverso da quello della funzione. Tutti i [Runtime Lambda](#) supportano le estensioni.

Un'estensione interna viene eseguita come parte del processo di runtime. La funzione accede alle estensioni interne utilizzando script wrapper o meccanismi in-process come `JAVA_TOOL_OPTIONS`. Per ulteriori informazioni, consulta [Modifica dell'ambiente di runtime](#).

Puoi aggiungere estensioni a una funzione utilizzando la console Lambda, AWS Command Line Interface (AWS CLI) o servizi e strumenti Infrastructure as code (IaC) come CloudFormation, AWS Serverless Application Model (AWS SAM) e Terraform.

Ti viene addebitato il runtime consumato dall'estensione (in incrementi di 1 ms). Non c'è alcun costo per installare le proprie estensioni. Per ulteriori informazioni sui prezzi delle estensioni, consulta la sezione [Prezzi AWS Lambda](#). Per informazioni sui prezzi per le estensioni dei partner, consulta i siti Web dei partner. Consulta [the section called “Partner con estensioni”](#) per l'elenco delle estensioni ufficiali dei partner.

## Argomenti

- [Ambiente di esecuzione](#)
- [Impatto su prestazioni e risorse](#)
- [Permissions](#)
- [Configurazione delle estensioni Lambda](#)
- [AWS Lambda estensioni partner](#)
- [Utilizzo dell'API Extensions di Lambda per creare estensioni](#)
- [Accesso ai dati di telemetria in tempo reale per le estensioni tramite l'API Telemetry](#)

## Ambiente di esecuzione

Lambda richiama la funzione in un [ambiente di esecuzione](#), che fornisce un ambiente di runtime sicuro e isolato. L'ambiente di esecuzione gestisce le risorse necessarie per eseguire la funzione e fornisce il supporto del ciclo di vita per il runtime e le estensioni della funzione.

Il ciclo di vita dell'ambiente di esecuzione prevede le seguenti fasi:

- **Init:** in questa fase, Lambda crea o sblocca un ambiente di esecuzione con le risorse configurate, scarica il codice per la funzione e tutti i livelli, inizializza le estensioni, inizializza il runtime e quindi

esegue il codice di inizializzazione della funzione (il codice al di fuori del gestore principale). La fase `Init` si verifica durante la prima invocazione o prima delle invocazioni di funzione se è stata abilitata la [concorrenza con provisioning](#).

La fase `Init` è suddivisa in tre sottofasi: `Extension init`, `Runtime init` e `Function init`. Queste sottofasi assicurano che tutte le estensioni e il runtime completino le loro attività di configurazione prima dell'esecuzione del codice della funzione.

Quando [SnapStart](#) è attivato, la fase `Init` si verifica quando si pubblica una versione della funzione. Lambda salva uno snapshot della memoria e dello stato del disco dell'ambiente di esecuzione inizializzato, mantiene lo snapshot crittografato e lo memorizza nella cache per l'accesso a bassa latenza. Se disponi di un [hook di runtime](#) prima del checkpoint, il codice viene eseguito alla fine della fase `Init`.

- **Restore**(SnapStart solo): quando richiami una [SnapStart](#) funzione per la prima volta e man mano che la funzione aumenta, Lambda riprende i nuovi ambienti di esecuzione dall'istantanea persistente invece di inizializzare la funzione da zero. Se disponi di un [hook di runtime](#) dopo il ripristino, il codice viene eseguito alla fine della fase `Restore`. Ti sarà addebitata la durata degli hook di runtime dopo il ripristino. Il runtime deve essere caricato e gli hook di runtime dopo il ripristino devono essere completati entro il limite di timeout (10 secondi). `SnapStartTimeoutException`Altrimenti, otterrai un. Al termine della fase `Restore`, Lambda chiama il gestore della funzione ([Invoca fase](#)).
- **Invoke**: in questa fase, Lambda invoca il gestore della funzione. Dopo che la funzione è stata completata, Lambda si prepara a gestire un'altra invocazione di funzione.
- **Shutdown**: questa fase viene attivata se la funzione Lambda non riceve alcuna invocazione per un certo periodo. Nella fase `Shutdown`, Lambda chiude il runtime, avvisa le estensioni per farle fermare in modo pulito, e poi rimuove l'ambiente. Lambda invia un evento `Shutdown` a ogni estensione; l'evento comunica all'estensione che l'ambiente sta per essere chiuso.

Durante la fase `Init`, Lambda estrae i livelli contenenti estensioni nella directory `/opt` nell'ambiente di esecuzione. Lambda cerca le estensioni nella directory `/opt/extensions/`, interpreta ogni file come un bootstrap eseguibile per avviare l'estensione e avvia tutte le estensioni in parallelo.

## Impatto su prestazioni e risorse

Le dimensioni delle estensioni della funzione vengono conteggiate per il limite di dimensioni del pacchetto di distribuzione. Per un archivio di file `.zip`, la dimensione totale decompressa della

funzione e di tutte le estensioni non può superare il limite della dimensione decompressa del pacchetto di distribuzione pari a 250 MB.

Le estensioni possono influire sulle prestazioni della funzione perché condividono risorse funzionali quali CPU, memoria e archiviazione. Ad esempio, se un'estensione esegue operazioni ad alta intensità di calcolo, potresti vedere aumentare la durata di esecuzione della tua funzione.

Ogni estensione deve completare la sua inizializzazione prima che Lambda richiami la funzione. Pertanto, un'estensione che consuma tempo di inizializzazione significativo può aumentare la latenza della chiamata di funzione.

Per misurare il tempo aggiuntivo impiegato dall'estensione dopo l'esecuzione della funzione, è possibile utilizzare la `PostRuntimeExtensionsDuration` [metrica della funzione](#). Per misurare l'aumento della memoria utilizzata, è possibile utilizzare la metrica `MaxMemoryUsed`. Per comprendere l'impatto di un'estensione specifica, è possibile eseguire diverse versioni delle funzioni affiancate.

#### Note

La metrica `MaxMemoryUsed` è una delle [Metriche raccolte da Lambda Insights](#) e non una metrica nativa di Lambda.

## Permissions

Le estensioni hanno accesso alle stesse risorse delle funzioni. Poiché le estensioni vengono eseguite nello stesso ambiente della funzione, le autorizzazioni vengono condivise tra la funzione e l'estensione.

Per un archivio di file .zip, puoi creare un CloudFormation modello per semplificare l'operazione di allegare la stessa configurazione di estensione, incluse le autorizzazioni ( AWS Identity and Access Management IAM), a più funzioni.

# Configurazione delle estensioni Lambda

## Configurazione delle estensioni (archivio di file .zip)

È possibile aggiungere un'estensione alla funzione come [livello Lambda](#). Utilizzando i livelli, puoi condividere le estensioni all'interno dell'organizzazione o con l'intera comunità di sviluppatori Lambda. È possibile aggiungere una o più estensioni a un livello. È possibile registrare fino a 10 estensioni per una funzione.

Si aggiunge l'estensione alla funzione utilizzando lo stesso metodo che si farebbe per qualsiasi livello. Per ulteriori informazioni, consulta [the section called "Livelli"](#).

Aggiungi un'estensione alla tua funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Seleziona la scheda Codice se non è già selezionata.
4. In Livelli, scegli Modifica.
5. Per Scegli un livello, scegliere Specifica un ARN.
6. In Specifica un ARN, inserire l'Amazon Resource Name (ARN) di un livello di estensione.
7. Scegliere Aggiungi.

## Utilizzo delle estensioni nelle immagini di container

È possibile aggiungere estensioni all'[immagine di container](#). L'impostazione ENTRYPOINT dell'immagine di container specifica il processo principale per la funzione. Configurare l'impostazione ENTRYPOINT nel Dockerfile o come sostituzione nella configurazione della funzione.

È possibile eseguire più processi all'interno di un container. Lambda gestisce il ciclo di vita del processo principale ed eventuali processi aggiuntivi. Lambda utilizza l'[API Estensioni](#) per gestire il ciclo di vita dell'estensione.

Esempio: aggiunta di un'estensione esterna

Un'estensione esterna viene eseguita in un processo separato dalla funzione Lambda. Lambda avvia un processo per ogni estensione nella directory `/opt/extensions/`. Lambda utilizza l'API Estensioni per gestire il ciclo di vita dell'estensione. Dopo che la funzione è stata eseguita completamente, Lambda invia un evento Shutdown a ciascuna estensione esterna.

## Example di aggiungere un'estensione esterna a un'immagine di base Python

```
FROM public.ecr.aws/lambda/python:3.11

# Copy and install the app
COPY /app /app
WORKDIR /app
RUN pip install -r requirements.txt

# Add an extension from the local directory into /opt/extensions
ADD my-extension.zip /opt/extensions
CMD python ./my-function.py
```

### Fasi successive

Per ulteriori informazioni sulle estensioni, si consiglia di utilizzare le seguenti risorse:

- Per un esempio di funzionamento di base, consulta [Building Extensions for AWS Lambda](#) nel blog di AWS Compute.
- Per informazioni sulle estensioni fornite da AWS Lambda Partners, consulta [Introducing AWS Lambda Extensions](#) sul blog di AWS Compute.
- Per visualizzare le estensioni di esempio e gli script wrapper disponibili, consulta [AWS Lambda Estensioni](#) nel repository Samples. AWS GitHub

## AWS Lambda estensioni partner

AWS Lambda ha collaborato con diverse entità terze per fornire estensioni da integrare con le funzioni Lambda. L'elenco seguente descrive le estensioni di terze parti pronte all'uso in qualsiasi momento.

- [AppDynamics](#)— Fornisce la strumentazione automatica delle Node.js nostre funzioni Python Lambda, fornendo visibilità e avvisi sulle prestazioni delle funzioni.
- [Axiom](#): fornisce i pannelli di controllo per il monitoraggio delle prestazioni della funzione Lambda e i parametri aggregati a livello di sistema.
- [Datadog](#): fornisce visibilità completa e in tempo reale alle applicazioni serverless tramite l'uso di metriche, tracce e log.
- [Dynatrace](#)— Fornisce visibilità su tracce e metriche e utilizza l'intelligenza artificiale per il rilevamento automatico degli errori e l'analisi delle cause principali nell'intero stack di applicazioni.
- [Elastic](#): fornisce il monitoraggio delle prestazioni delle applicazioni (APM) per identificare e risolvere i problemi principali utilizzando tracce, parametri e log correlati.
- [Epsagon](#): attende gli eventi di richiamo, memorizza le tracce e le invia in parallelo alle esecuzioni delle funzioni Lambda.
- [Fastly](#)— Protegge le funzioni Lambda da attività sospette, come attacchi di tipo injection, acquisizione di account tramite riempimento di credenziali, bot dannosi e abuso di API.
- [HashiCorp Vault](#): gestisce i segreti e li rende disponibili affinché gli sviluppatori possano utilizzarli all'interno del codice della funzione, senza rendere le funzioni consapevoli di Vault.
- [Honeycomb](#): strumento di osservabilità per il debug dello stack di app.
- [Lumigo](#): profila i richiami delle funzioni Lambda e raccoglie le metriche per la risoluzione dei problemi in ambienti serverless e di microservizi.
- [New Relic](#): funziona insieme alle funzioni Lambda, raccogliendo, migliorando e trasportando automaticamente la telemetria alla piattaforma di osservabilità unificata di New Relic.
- [Sedai](#)— Una piattaforma di gestione del cloud autonoma, alimentata da AI/ML, che offre un'ottimizzazione continua per i team operativi del cloud per massimizzare i risparmi sui costi, le prestazioni e la disponibilità del cloud su larga scala.
- [Sentry](#): esegue la diagnostica, corregge e ottimizza le prestazioni delle funzioni Lambda.
- [Site24x7](#): ottiene osservabilità in tempo reale negli ambienti Lambda
- [Splunk](#): raccoglie parametri ad alta risoluzione e bassa latenza per un monitoraggio efficiente ed efficace delle funzioni Lambda.

- [Sumo Logic](#): fornisce visibilità sullo stato e sulle prestazioni delle applicazioni serverless.
- [Salt Security](#): semplifica la governance dello stato delle API e la sicurezza delle API per le funzioni Lambda attraverso la configurazione e il supporto automatizzati per diversi runtime.

## AWS estensioni gestite

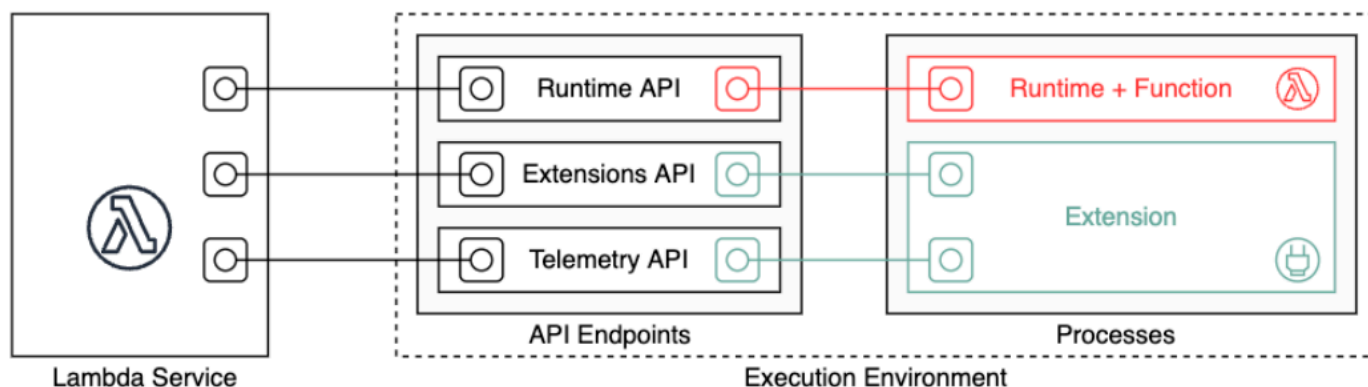
AWS fornisce le proprie estensioni gestite, tra cui:

- [AWS AppConfig](#)— Usa i flag delle funzionalità e i dati dinamici per aggiornare le tue funzioni Lambda. È inoltre possibile utilizzare questa estensione per aggiornare altre configurazioni dinamiche, come la limitazione e l'ottimizzazione delle operazioni.
- [Amazon CodeGuru Profiler](#): migliora le prestazioni delle applicazioni e riduce i costi individuando la riga di codice più costosa di un'applicazione e fornendo consigli per migliorarla.
- [CloudWatch Lambda Insights](#): monitora, risolvi i problemi e ottimizza le prestazioni delle tue funzioni Lambda tramite dashboard automatizzate.
- [AWS Distro for OpenTelemetry \(ADOT\)](#): consente alle funzioni di inviare dati di traccia a servizi di AWS monitoraggio come e verso destinazioni che supportano come AWS X-Ray Honeycomb e Lightstep. OpenTelemetry
- [AWS Parametri e segreti](#): recupera in modo sicuro i parametri da AWS Systems Manager Parameter Store e i segreti dalle funzioni Lambda. AWS Secrets Manager

Per ulteriori esempi di estensioni e progetti dimostrativi, consultare [Estensioni AWS Lambda](#).

## Utilizzo dell'API Extensions di Lambda per creare estensioni

Gli autori di funzioni Lambda utilizzano le estensioni per integrare Lambda con gli strumenti preferiti per il monitoraggio, l'osservabilità, la sicurezza e la governance. Gli autori delle funzioni possono utilizzare estensioni di AWS, AWS Partner e progetti open source. Per ulteriori informazioni sull'utilizzo delle estensioni, consulta [Introducing AWS Lambda Extensions](#) on the AWS Compute Blog. In questa sezione viene descritto come utilizzare l'API di estensione Lambda, il ciclo di vita dell'ambiente di esecuzione Lambda e la documentazione di riferimento delle API di estensione Lambda.



In qualità di autore dell'estensione, l'utente utilizza l'API estensioni per ottenere un'integrazione profonda nell'[ambiente di esecuzione](#) Lambda. L'estensione può registrarsi per funzioni ed eventi del ciclo di vita dell'ambiente di esecuzione. In risposta a questi eventi, è possibile avviare nuovi processi, eseguire la logica e controllare e partecipare a tutte le fasi del ciclo di vita di Lambda: inizializzazione, invocazione e arresto. Inoltre, è possibile utilizzare l'[API Log di runtime](#) per ricevere un flusso di log.

Un'estensione esterna viene eseguita come processo indipendente nell'ambiente di esecuzione e continua a funzionare anche dopo che la chiamata della funzione è stata completamente elaborata. Poiché le estensioni esterne vengono eseguite come processi, è possibile scriverle in un linguaggio diverso da quella della funzione. Si consiglia di implementare le estensioni utilizzando un linguaggio compilato.

In questo caso, l'estensione è un binario autonomo compatibile con tutti i tempi di esecuzione supportati. Tutti i [Runtime Lambda](#) supportano le estensioni. Se si utilizza un linguaggio non compilato, assicurarsi di includere un runtime compatibile nell'estensione.

Lambda supporta anche le estensioni interne. Un'estensione interna viene eseguita come thread separato nel processo di runtime. Il runtime avvia e arresta l'estensione interna. Un modo alternativo



per integrarsi con l'ambiente Lambda consiste nell'utilizzare [variabili d'ambiente e script wrapper](#) specifici del linguaggio. Le estensioni interne consentono di configurare l'ambiente di runtime e modificare il comportamento di startup del processo di runtime.

È possibile aggiungere estensioni a una funzione in due modi. Per una funzione distribuita come [archivio di file con estensione zip](#), è possibile distribuire l'estensione come [livello](#). Per una funzione definita come immagine di container, [le estensioni](#) vengono aggiunte all'immagine del container.

#### Note

Per esempi di estensioni e script wrapper, vedi [AWS Lambda Estensioni](#) nel repository Samples. AWS GitHub

## Argomenti

- [Ciclo di vita dell'ambiente di esecuzione Lambda](#)
- [Riferimento all'API delle estensioni](#)

## Ciclo di vita dell'ambiente di esecuzione Lambda

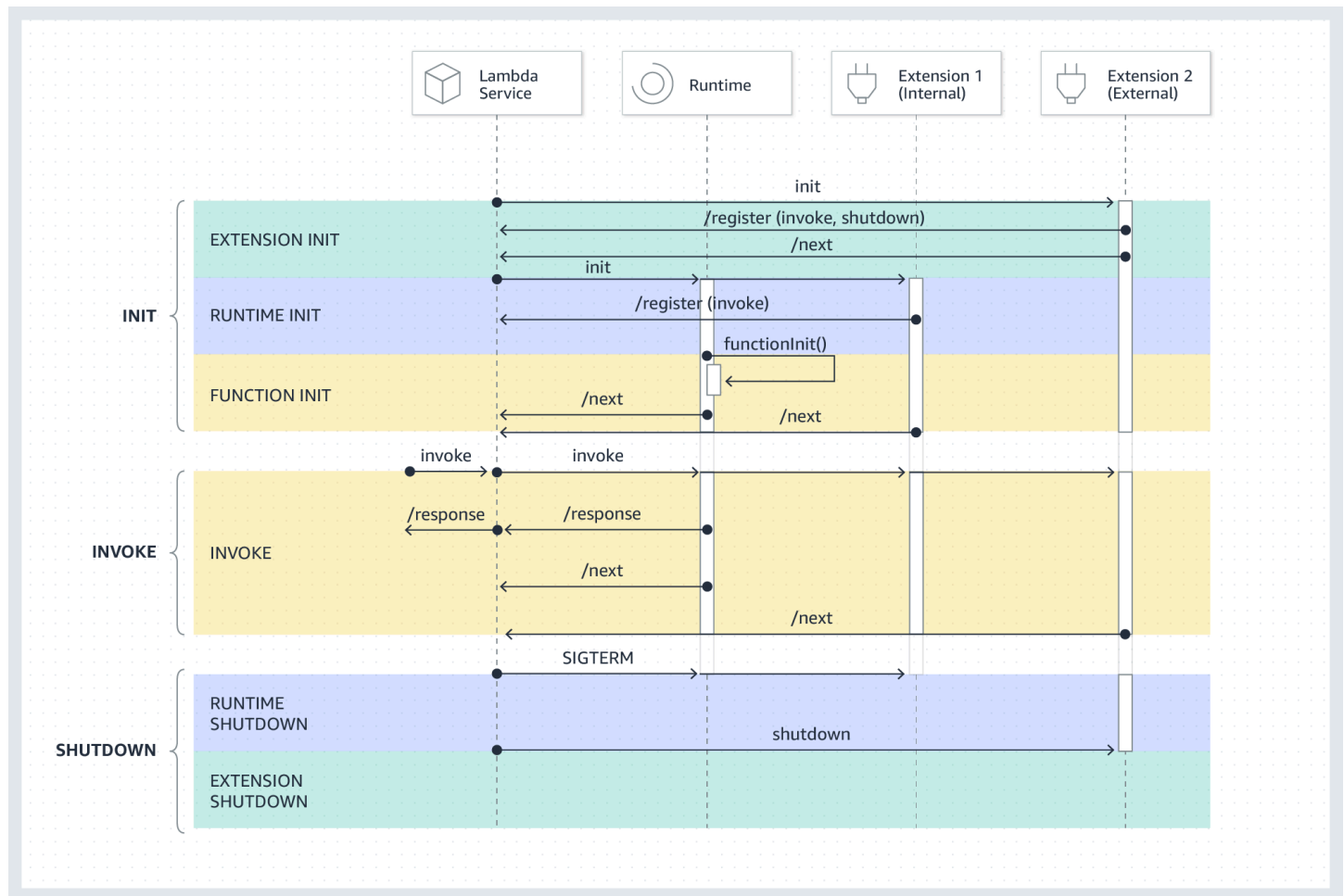
Il ciclo di vita dell'ambiente di esecuzione prevede le seguenti fasi:

- **Init:** in questa fase, Lambda crea o sblocca un ambiente di esecuzione con le risorse configurate, scarica il codice per la funzione e tutti i livelli, inizializza qualsiasi estensione, inizializza il runtime e quindi esegue il codice di inizializzazione della funzione (il codice all'esterno del gestore principale). La fase Init si verifica durante la prima invocazione o prima delle invocazioni di funzione se è stata abilitata la [concorrenza con provisioning](#).

La fase Init è suddivisa in tre sotto fasi: `Extension init`, `Runtime init` e `Function init`. Queste sotto fasi assicurano che tutte le estensioni e il runtime completino le loro attività di configurazione prima dell'esecuzione del codice della funzione.

- **Invoke:** in questa fase, Lambda invoca il gestore della funzione. Dopo che la funzione è stata completata, Lambda si prepara a gestire un'altra invocazione di funzione.
- **Shutdown:** questa fase viene attivata se la funzione Lambda non riceve alcuna invocazione per un certo periodo. Nella fase Shutdown, Lambda chiude il runtime, avvisa le estensioni per farle fermare in modo pulito, e poi rimuove l'ambiente. Lambda invia un evento Shutdown a ogni estensione; l'evento comunica all'estensione che l'ambiente sta per essere chiuso.

Ogni fase inizia con un evento da Lambda al runtime e a tutte le estensioni registrate. Il runtime e ogni estensione segnalano il completamento inviando una richiesta API Next. Lambda congela l'ambiente di esecuzione quando ogni processo è stato completato e non ci sono eventi in sospeso.



## Argomenti

- Fase di init
- Invoca fase
- Fase di arresto
- Autorizzazioni e configurazione
- Gestione dei guasti
- Risoluzione dei problemi delle estensioni

## Fase di init

Durante la fase `Extension init`, per ricevere gli eventi ogni estensione deve registrarsi con Lambda. Lambda utilizza il nome file completo dell'estensione per confermare che l'estensione abbia completato la sequenza di bootstrap. Pertanto, ogni chiamata `Register API` deve includere l'intestazione `Lambda-Extension-Name` con il nome file completo dell'estensione.

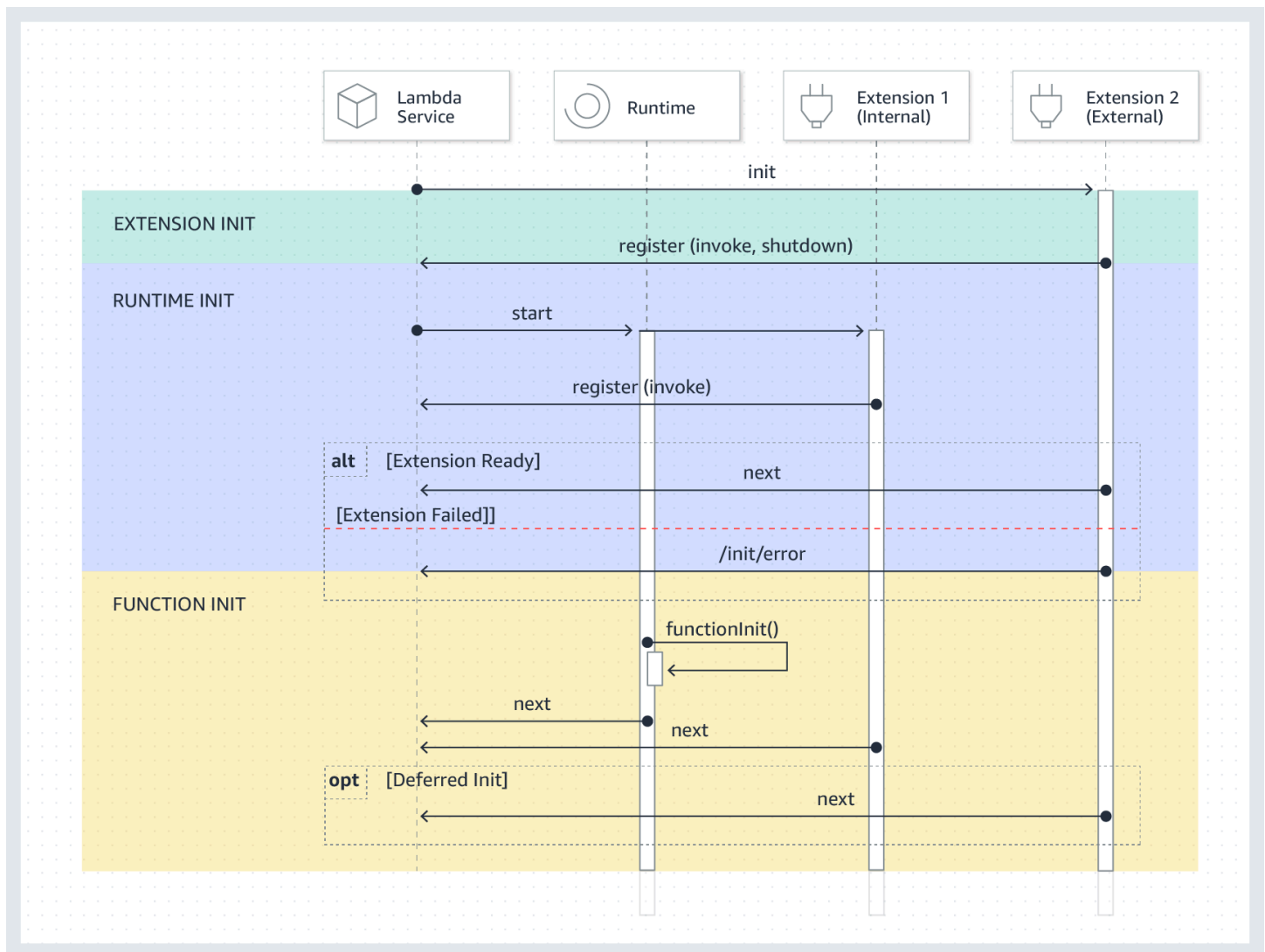
È possibile registrare fino a 10 estensioni per una funzione. Questo limite viene applicato tramite la chiamata `Register API`.

Dopo la registrazione di ogni estensione, Lambda inizia la fase `Runtime init`. Il processo di runtime chiama `functionInit` per avviare la fase `Function init`.

La fase `Init` viene completata dopo il runtime e ogni estensione registrata indica il completamento inviando una richiesta `Next API`.

### Note

Le estensioni possono completare la loro inizializzazione in qualsiasi punto della fase `Init`.



## Invoca fase

Quando viene richiamata una funzione Lambda in risposta a una richiesta API Next, Lambda invia un evento `Invoke` al runtime e a ogni estensione registrata per l'evento `Invoke`.

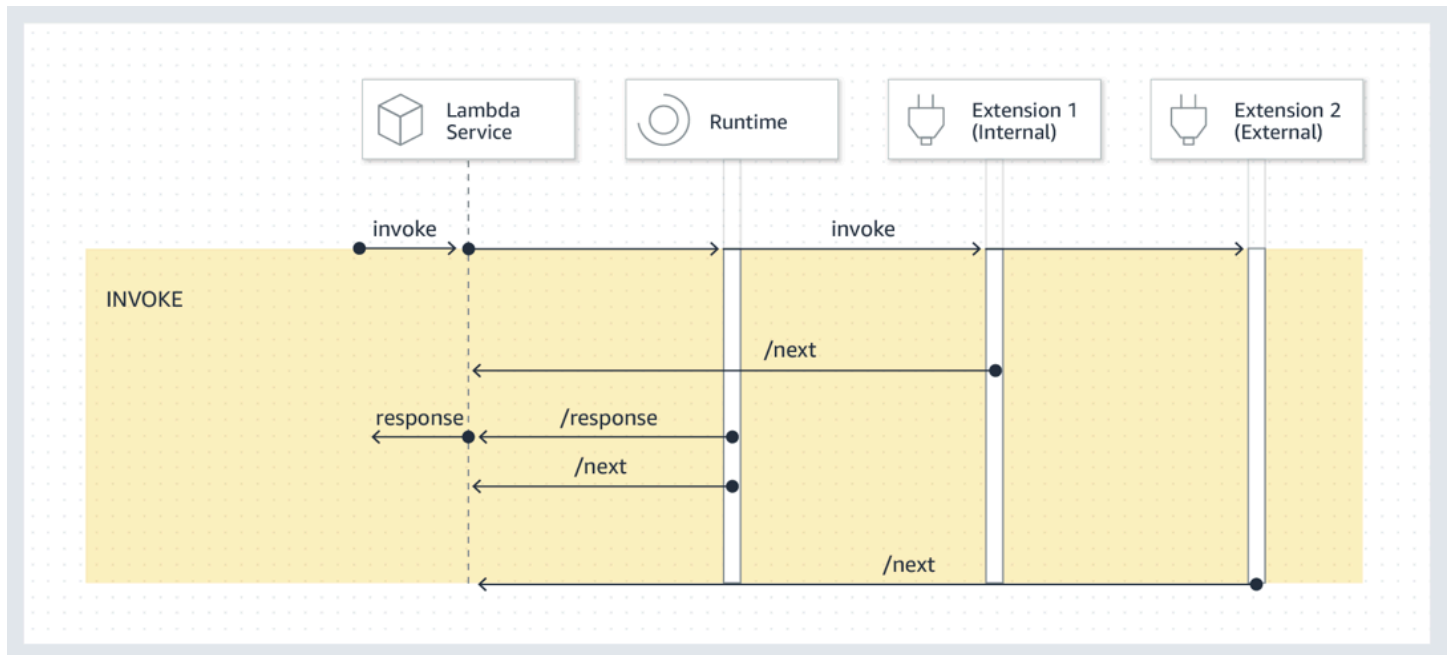
### Note

Istanze gestite Lambda: le estensioni per le funzioni Lambda Managed Instances non possono registrarsi per l'evento `Invoke`. Poiché Lambda Managed Instances supporta chiamate simultanee all'interno di un singolo ambiente di esecuzione, l'evento non è supportato. Le estensioni possono solo registrarsi all'evento `Shutdown`. Se devi tenere traccia dell'inizio e della fine delle chiamate, utilizza l'evento della `platform.report` piattaforma tramite l'API di [telemetry](#).

Durante la chiamata, le estensioni esterne vengono eseguite in parallelo con la funzione. Continuano anche a funzionare dopo che la funzione è stata completata. Puoi utilizzarla per acquisire informazioni diagnostiche o per inviare registri, metriche e tracce a una posizione a tua scelta.

Dopo aver ricevuto la risposta della funzione dal runtime, Lambda restituisce la risposta al client, anche se le estensioni sono ancora in esecuzione.

La fase Invoke termina dopo il runtime e tutte le estensioni segnalano che vengono eseguite inviando una richiesta Next API.



Payload evento: l'evento inviato al runtime (e alla funzione Lambda) contiene l'intera richiesta, le intestazioni (ad esempio `RequestId`) e il payload. L'evento inviato a ciascuna estensione contiene metadati che descrivono il contenuto dell'evento. Questo evento del ciclo di vita include il tipo di evento, l'ora di timeout della funzione (`deadlineMs`), il `requestId`, la funzione richiamata Amazon Resource Name (ARN) e le intestazioni di traccia.

Le estensioni che desiderano accedere al corpo dell'evento funzione possono utilizzare un SDK in runtime che comunica con l'estensione. Gli sviluppatori di funzioni utilizzano l'SDK in runtime per inviare il payload all'estensione quando viene richiamata la funzione.

Ecco un esempio di payload:

```
{
  "eventType": "INVOKE",
  "deadlineMs": 676051,
```

```
"requestId": "3da1f2dc-3222-475e-9205-e2e6c6318895",
"invokedFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:ExtensionTest",
"tracing": {
  "type": "X-Amzn-Trace-Id",
  "value":
    "Root=1-5f35ae12-0c0fec141ab77a00bc047aa2;Parent=2be948a625588e32;Sampled=1"
}
```

**Duration limit (Limite di durata):** l'impostazione di timeout della funzione limita la durata dell'intera fase Invoke. Ad esempio, se si imposta il timeout della funzione come 360 secondi, la funzione e tutte le estensioni devono essere completate entro 360 secondi. Si noti che non esiste una fase post-richiamo indipendente. La durata è la somma di tutto il tempo necessario per il completamento del runtime e di tutte le invocazioni delle estensioni e non viene calcolata fino a quando la funzione e tutte le estensioni non hanno terminato l'esecuzione.

**Impatto sulle prestazioni e sovraccarico di estensione:** le estensioni possono influire sulle prestazioni delle funzioni. In qualità di autore dell'estensione, hai il controllo sull'impatto sulle prestazioni della tua estensione. Ad esempio, se una estensione svolge operazioni con uso intensivo dell'elaborazione, la durata della funzione aumenta poiché l'estensione e il codice della funzione condividono le stesse risorse della CPU. Inoltre, se l'estensione esegue operazioni estese al termine dell'invocazione della funzione, la durata della funzione aumenta perché la fase Invoke continua fino a quando tutte le estensioni non segnalano che sono state completate.

#### Note

Lambda alloca la potenza della CPU in proporzione all'impostazione della memoria della funzione. Potresti avere una maggiore durata di esecuzione e inizializzazione a impostazioni di memoria inferiori perché i processi di funzione e estensione sono in concorrenza per le stesse risorse della CPU. Per ridurre la durata di esecuzione e inizializzazione, prova ad aumentare l'impostazione della memoria.

Per aiutare a identificare l'impatto sulle prestazioni introdotto dalle estensioni sulla fase Invoke, Lambda esegue l'output del parametro `PostRuntimeExtensionsDuration`. Questa metrica misura il tempo cumulativo trascorso tra la richiesta Next API runtime e l'ultima richiesta Next API di estensione. Per misurare l'aumento della memoria utilizzata, utilizzare la metrica `MaxMemoryUsed`.

Per ulteriori informazioni sugli stati delle funzioni, consultare [Utilizzo delle CloudWatch metriche con Lambda](#).

Gli sviluppatori di funzioni possono eseguire diverse versioni delle loro funzioni fianco a fianco per comprendere l'impatto di un'estensione specifica. Si consiglia agli autori di estensioni di pubblicare il consumo previsto di risorse per semplificare la scelta di un'estensione adatta per gli sviluppatori di funzioni.

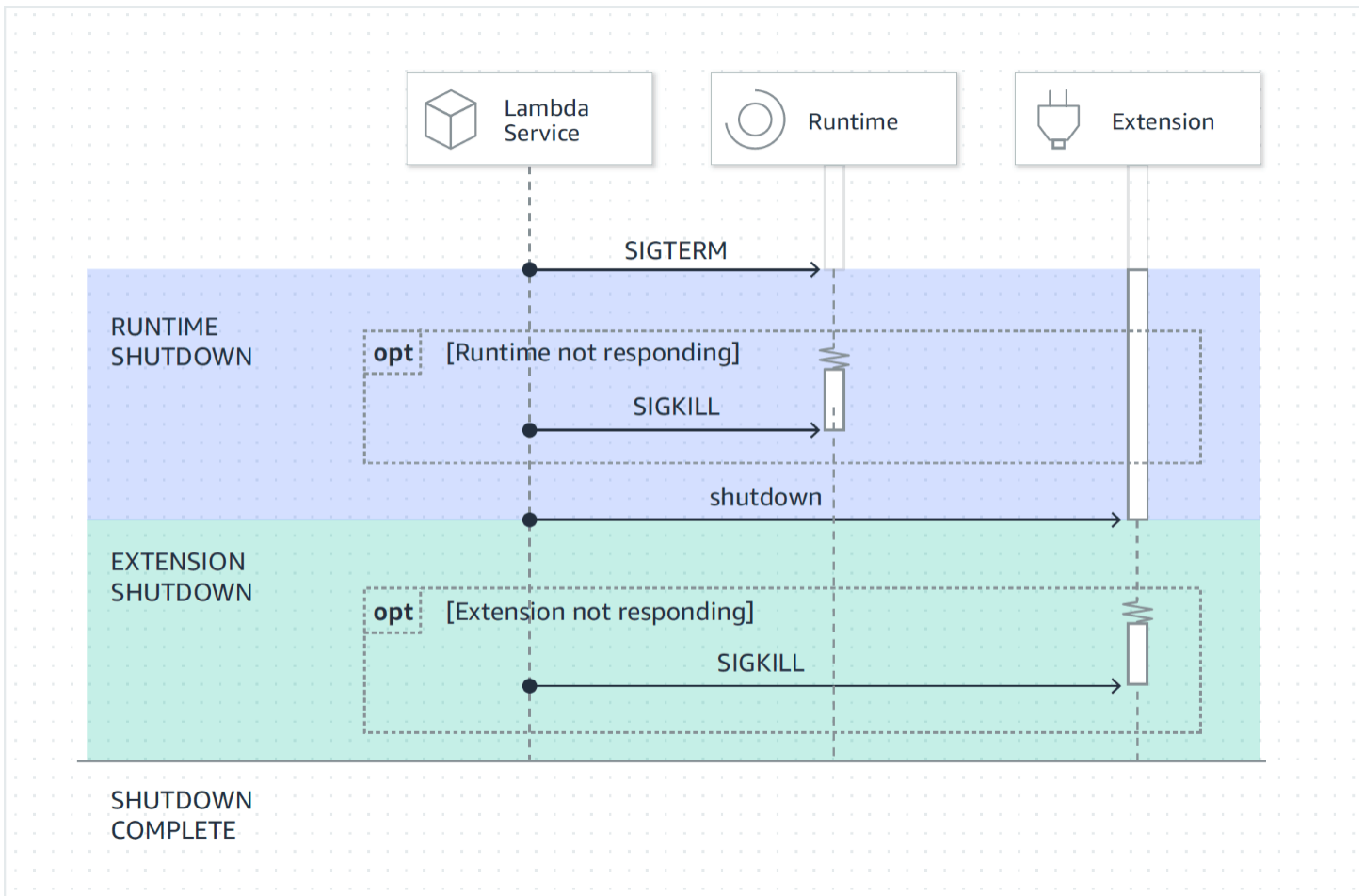
## Fase di arresto

Quando Lambda sta per chiudere il runtime, invia uno Shutdown a ciascuna estensione esterna registrata. Le estensioni possono utilizzare questo tempo per le attività di pulizia finali. L'evento Shutdown viene inviato in risposta a una richiesta Next API.

Limite di durata: la durata massima della fase Shutdown dipende dalla configurazione delle estensioni registrate:

- 0 ms: funzione senza estensioni registrate
- 500 ms: funzione con estensione interna registrata
- 2.000 ms: funzione con una o più estensioni esterne registrate

Se il runtime o un'estensione non risponde all'evento Shutdown entro il limite, Lambda termina il processo utilizzando un segnale SIGKILL.



Payload evento: l'evento Shutdown contiene il motivo dell'arresto e il tempo rimanente in millisecondi.

shutdownReason include quanto segue:

- SPINDOWN – Arresto normale
- TIMEOUT – Timeout del limite di durata
- FAILURE – Condizione di errore, ad esempio un evento out-of-memory

```
{
  "eventType": "SHUTDOWN",
  "shutdownReason": "reason for shutdown",
  "deadlineMs": "the time and date that the function times out in Unix time milliseconds"
}
```



## Autorizzazioni e configurazione

Le estensioni vengono eseguite nello stesso ambiente di esecuzione della funzione Lambda. Inoltre, le estensioni condividono con la funzione risorse quali CPU, memoria e archiviazione /tmp su disco. Inoltre, le estensioni utilizzano lo stesso ruolo AWS Identity and Access Management (IAM) e lo stesso contesto di sicurezza della funzione.

Autorizzazioni di accesso al file system e alla rete: le estensioni vengono eseguite nello stesso file system e nello stesso spazio dei nomi dei nomi di rete del runtime della funzione. Ciò significa che le estensioni devono essere compatibili con il sistema operativo associato. Se un'estensione richiede ulteriori regole di traffico di rete in uscita, è necessario applicare tali regole alla configurazione della funzione.

### Note

Poiché la directory del codice funzione è di sola lettura, le estensioni non possono modificare il codice della funzione.

Variabili di ambiente: le estensioni possono accedere alle [variabili di ambiente](#), della funzione, ad eccezione delle seguenti variabili specifiche del processo di runtime:

- AWS\_EXECUTION\_ENV
- AWS\_LAMBDA\_LOG\_GROUP\_NAME
- AWS\_LAMBDA\_LOG\_STREAM\_NAME
- AWS\_XRAY\_CONTEXT\_MISSING
- AWS\_XRAY\_DAEMON\_ADDRESS
- LAMBDA\_RUNTIME\_DIR
- LAMBDA\_TASK\_ROOT
- \_AWS\_XRAY\_DAEMON\_ADDRESS
- \_AWS\_XRAY\_DAEMON\_PORT
- \_HANDLER

**Note**

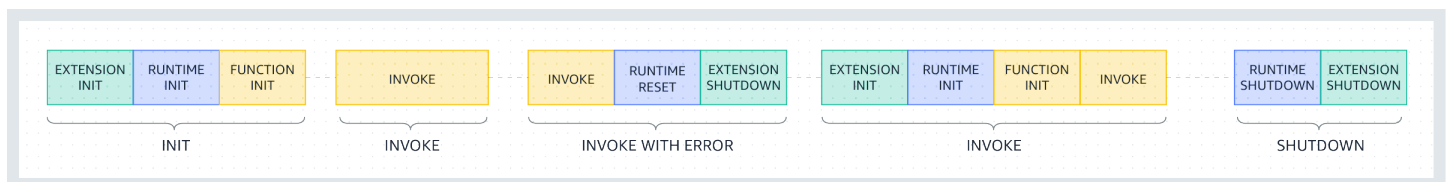
Rilevamento delle istanze gestite Lambda: le estensioni possono controllare la variabile di ambiente `AWS_LAMBDA_INITIALIZATION_TYPE` per determinare se sono in esecuzione su istanze gestite Lambda rispetto alle funzioni Lambda (predefinite). Questo è il metodo consigliato per le estensioni per adattare il loro comportamento in base al tipo di ambiente di esecuzione.

**Gestione dei guasti**

**Errori di inizializzazione:** se un'estensione genera un errore, Lambda riavvia l'ambiente di esecuzione per imporre un comportamento coerente e per incoraggiare la risposta immediata agli errori per le estensioni. Inoltre, per alcuni clienti, le estensioni devono soddisfare esigenze mission-critical quali registrazione, sicurezza, governance e raccolta di dati di telemetria.

**Invocare errori (come memoria esaurita, timeout funzione):** poiché le estensioni condividono le risorse con il runtime, sono influenzata dall'esaurimento della memoria. Quando il runtime fallisce, tutte le estensioni e il runtime stesso partecipano alla fase Shutdown. Inoltre, il runtime viene riavviato, automaticamente come parte della chiamata corrente o tramite un meccanismo di reinizializzazione differita.

Se si verifica un errore (ad esempio un timeout della funzione o un errore di runtime) durante Invoke, il servizio Lambda esegue un reset. Il reset si comporta come un evento Shutdown. Innanzitutto Lambda chiude il runtime, poi invia un evento Shutdown a ogni estensione esterna registrata. L'evento include il motivo dell'arresto. Se questo ambiente viene utilizzato per una nuova chiamata, l'estensione e il runtime vengono reinizializzati come parte della chiamata successiva.



Per una spiegazione più approfondita del diagramma precedente, consulta [Errori durante la fase di richiamo](#).

**Log delle estensioni:** Lambda invia l'output del log delle estensioni a Logs. CloudWatch Lambda genera anche un evento di log aggiuntivo per ogni estensione durante `Init`. L'evento di log registra il nome e la preferenza di registrazione (evento, configurazione) in caso di successo o il motivo dell'errore in caso di errore.

## Risoluzione dei problemi delle estensioni

- Se una richiesta `Register` non riesce, assicurarsi che l'intestazione `Lambda-Extension-Name` nella chiamata `Register API` contenga il nome file completo dell'estensione.
- Se la richiesta `Register` non riesce per un'estensione interna, assicurarsi che la richiesta non si registra per l'evento `Shutdown`.

## Riferimento all'API delle estensioni

La specifica OpenAPI per le estensioni API versione 2020-01-01 è disponibile qui: [extensions-api.zip](#)

È possibile recuperare il valore dell'endpoint API dalla variabile di ambiente `AWS_LAMBDA_RUNTIME_API`. Per inviare una richiesta `Register`, utilizzare il prefisso `2020-01-01/` prima di ogni percorso API. Ad esempio:

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
```

## Metodi API

- [Registrati](#)
- [Next](#)
- [Errore di init](#)
- [Errore di uscita](#)

## Registrati

Durante `Extension init`, per ricevere gli eventi tutte le estensioni devono registrarsi con Lambda. Lambda utilizza il nome file completo dell'estensione per confermare che l'estensione abbia completato la sequenza di bootstrap. Pertanto, ogni chiamata `Register API` deve includere l'intestazione `Lambda-Extension-Name` con il nome file completo dell'estensione.

Le estensioni interne vengono avviate e arrestate dal processo di runtime, pertanto non sono autorizzate a registrarsi per l'evento `Shutdown`.

Percorso – `/extension/register`

Metodo – `POST`

## Intestazioni della richiesta

- `Lambda-Extension-Name` – Il nome file completo dell'estensione. Campo obbligatorio: sì Tipo: stringa
- `Lambda-Extension-Accept-Feature`: si utilizza per specificare le funzionalità opzionali delle estensioni durante la registrazione. Campo obbligatorio: no. Tipo: stringa separata da virgole. Funzionalità disponibili da specificare tramite questa impostazione:
  - `accountId`— Se specificato, la risposta di registrazione dell'estensione contiene l'ID dell'account associato alla funzione Lambda per la quale stai registrando l'estensione.

## Parametri del corpo della richiesta

- `events` – Array degli eventi a cui registrarsi. Campo obbligatorio: no. Tipo: array di stringhe  
Stringhe valide: `INVOKE`, `SHUTDOWN`.

### Note

Istanze gestite Lambda: le estensioni per le funzioni Lambda Managed Instances possono registrarsi solo per l'evento. `SHUTDOWN` Il tentativo di registrazione all'`INVOKE` evento genera un errore. Questo perché Lambda Managed Instances supporta invocazioni simultanee all'interno di un singolo ambiente di esecuzione.

## Intestazioni di risposta

- `Lambda-Extension-Identifier` – Identificativo univoco dell'agente generato (stringa UUID) necessario per tutte le richieste successive.

## Codice di risposta

- 200 – Il corpo della risposta contiene il nome della funzione, la versione della funzione e il nome del gestore.
- 400 – Richiesta non valida
- 403 – Non consentito
- 500 — Errore nel contenitore. Non-recoverable stato. L'estensione dovrebbe uscire tempestivamente.

## Example Esempio di corpo della richiesta

```
{
  'events': [ 'INVOKE', 'SHUTDOWN' ]
}
```

## Example Esempio di corpo della risposta

```
{
  "functionName": "helloWorld",
  "functionVersion": "$LATEST",
  "handler": "lambda_function.lambda_handler"
}
```

## Example Esempio di corpo della risposta con funzione accountId facoltativa

```
{
  "functionName": "helloWorld",
  "functionVersion": "$LATEST",
  "handler": "lambda_function.lambda_handler",
  "accountId": "123456789012"
}
```

## Next

Le estensioni inviano una richiesta Next API per ricevere l'evento successivo, che può essere un evento Invoke o un evento Shutdown. Il corpo della risposta contiene il payload, che è un documento JSON che contiene i dati degli eventi.

L'estensione invia una richiesta Next API per segnalare che è pronta per ricevere nuovi eventi. Questa è una chiamata di blocco.

Non impostare un timeout sulla chiamata GET, poiché l'estensione può essere sospesa per un periodo di tempo fino a quando non c'è un evento da restituire.

Percorso – /extension/event/next

Metodo – GET

## Intestazioni della richiesta

- `Lambda-Extension-Identifier` – Identificativo univoco dell'estensione (stringa UUID). Campo obbligatorio: sì Tipo: stringa UUID.

## Intestazioni di risposta

- `Lambda-Extension-Event-Identifier`: identificatore univoco per l'evento (stringa UUID).

## Codice di risposta

- 200 – La risposta contiene informazioni sull'evento successivo (`EventInvoke` o `EventShutdown`).
- 403 – Non consentito
- 500 — Errore nel contenitore. Non-recoverable stato. L'estensione dovrebbe uscire tempestivamente.

## Errore di init

L'estensione utilizza questo metodo per segnalare un errore di inizializzazione a Lambda. Chiamalo quando l'estensione non riesce a inizializzare dopo che si è registrata. Dopo che Lambda riceve l'errore, le chiamate API successive hanno esito negativo. L'estensione dovrebbe terminare dopo aver ricevuto la risposta da Lambda.

Percorso – `/extension/init/error`

Metodo – POST

## Intestazioni della richiesta

- `Lambda-Extension-Identifier` – Identificativo univoco dell'estensione. Campo obbligatorio: sì Tipo: stringa UUID.
- `Lambda-Extension-Function-Error-Type` – Tipo di errore rilevato dall'estensione. Campo obbligatorio: sì L'intestazione è costituita da un valore stringa. Lambda accetta qualsiasi stringa, ma si consiglia di utilizzare il formato `<categoria.motivo>`. Ad esempio:
  - `Extension.NoSuchHandler`
  - `Extension.APIKeyNotFound`

- `Extension.ConfigInvalid`
- `Extension.UnknownReason`

### Parametri del corpo della richiesta

- `ErrorRequest` – Informazioni sull'errore. Campo obbligatorio: no.

Questo campo è un oggetto JSON con la seguente struttura:

```
{
    errorMessage: string (text description of the error),
    errorType: string,
    stackTrace: array of strings
}
```

NB: Lambda accetta qualsiasi valore per `errorType`.

Nell'esempio seguente viene mostrato un messaggio di errore della funzione Lambda in cui la funzione non è stata in grado di analizzare i dati evento forniti nell'invocazione.

### Example Errore di funzione

```
{
    "errorMessage" : "Error parsing event data.",
    "errorType" : "InvalidEventDataException",
    "stackTrace": [ ]
}
```

### Codice di risposta

- 202 – Accettato
- 400 – Richiesta non valida
- 403 – Non consentito
- 500 — Errore nel contenitore. Non-recoverable stato. L'estensione dovrebbe uscire tempestivamente.

## Errore di uscita

L'estensione utilizza questo metodo per segnalare un errore a Lambda prima di uscire. Chiamalo quando incontri un errore inaspettato. Dopo che Lambda riceve l'errore, le chiamate API successive hanno esito negativo. L'estensione dovrebbe terminare dopo aver ricevuto la risposta da Lambda.

Percorso – `/extension/exit/error`

Metodo – POST

Intestazioni della richiesta

- `Lambda-Extension-Identifier` – Identificativo univoco dell'estensione. Campo obbligatorio: sì  
Tipo: stringa UUID.
- `Lambda-Extension-Function-Error-Type` – Tipo di errore rilevato dall'estensione. Campo obbligatorio: sì  
L'intestazione è costituita da un valore stringa. Lambda accetta qualsiasi stringa, ma si consiglia di utilizzare il formato `<categoria.motivo>`. Ad esempio:
  - `Extension.NoSuchHandler`
  - `Extension.APIKeyNotFound`
  - `Extension.ConfigInvalid`
  - `Extension.UnknownReason`

Parametri del corpo della richiesta

- `ErrorRequest` – Informazioni sull'errore. Campo obbligatorio: no.

Questo campo è un oggetto JSON con la seguente struttura:

```
{
  errorMessage: string (text description of the error),
  errorType: string,
  stackTrace: array of strings
}
```

NB: Lambda accetta qualsiasi valore per `errorType`.

Nell'esempio seguente viene mostrato un messaggio di errore della funzione Lambda in cui la funzione non è stata in grado di analizzare i dati evento forniti nell'invocazione.



## Example Errore di funzione

```
{  
  "errorMessage" : "Error parsing event data.",  
  "errorType" : "InvalidEventDataException",  
  "stackTrace": [ ]  
}
```

## Codice di risposta

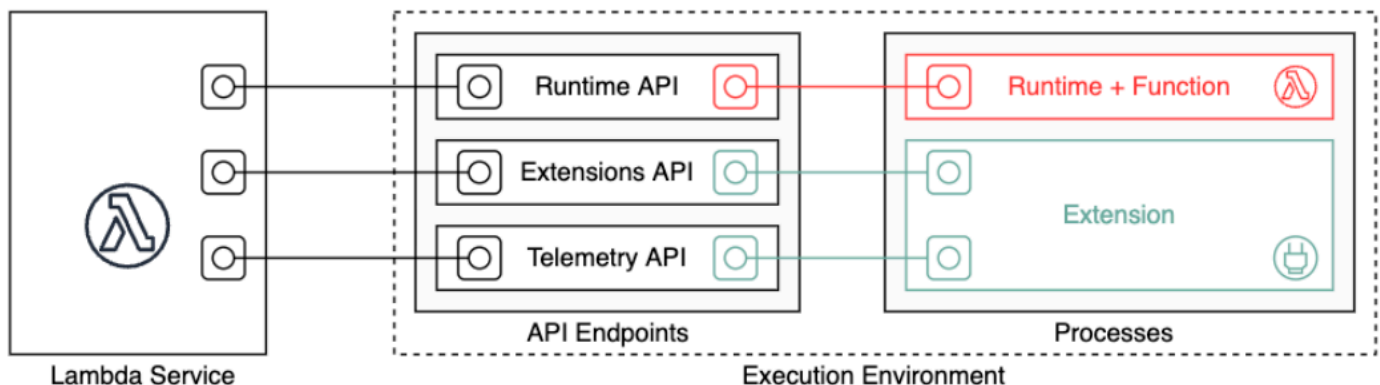
- 202 – Accettato
- 400 – Richiesta non valida
- 403 – Non consentito
- 500 — Errore nel contenitore. Non-recoverable stato. L'estensione dovrebbe uscire tempestivamente.

# Accesso ai dati di telemetria in tempo reale per le estensioni tramite l'API Telemetry

L'API Telemetry consente alle estensioni di ricevere dati di telemetria direttamente da Lambda. Durante l'inizializzazione e l'invocazione, Lambda acquisisce in automatico i dati di telemetria, tra cui log, parametri della piattaforma e tracce della piattaforma. L'API Telemetry consente alle estensioni di accedere a tali dati di telemetria direttamente da Lambda quasi in tempo reale.

All'interno dell'ambiente di esecuzione Lambda, puoi abbonare le estensioni Lambda ai flussi di telemetria. Dopo l'abbonamento, Lambda invia automaticamente tutti i dati di telemetria alle estensioni. Hai quindi la flessibilità di elaborare, filtrare e inviare i dati alla tua destinazione preferita, ad esempio un bucket Amazon Simple Storage Service (Amazon S3) o un fornitore di strumenti di osservabilità di terze parti.

Il diagramma seguente mostra come l'API Extensions e l'API Telemetry connettono le estensioni a Lambda dall'interno dell'ambiente di esecuzione. Inoltre L'API Runtime collega il runtime e la funzione a Lambda.



## ⚠ Important

L'API di telemetria Lambda sostituisce l'API Lambda Logs. Sebbene l'API Logs rimanga completamente funzionante, in futuro consigliamo di utilizzare solo l'API di telemetria. Puoi iscrivere la tua estensione a un flusso di telemetria utilizzando l'API di telemetria o l'API Logs. Dopo la sottoscrizione tramite una di queste API, qualsiasi tentativo di sottoscrizione utilizzando l'altra API restituirà un errore.

### Requisito della versione dello schema Lambda Managed Instances

Le istanze gestite Lambda supportano solo la versione 2025-01-29 dello schema dell'API di telemetria. Quando ci si iscrive ai flussi di telemetria per le funzioni di Managed Instance, è necessario utilizzarli nella richiesta di abbonamento. `"schemaVersion": "2025-01-29"`

L'utilizzo di versioni precedenti dello schema comporta il rifiuto degli eventi da parte di Lambda.

La versione 2025-01-29 dello schema è retrocompatibile e può essere utilizzata sia con le funzioni Lambda Managed Instances che con le funzioni Lambda (predefinite). Consigliamo di utilizzare questa versione per tutte le nuove estensioni per garantire la compatibilità tra entrambi i modelli di distribuzione.

Le estensioni possono utilizzare l'API di telemetria per sottoscrivere i tre diversi flussi di telemetria.

- Telemetria della piattaforma: registri, metriche e tracce, che descrivono eventi ed errori relativi al ciclo di vita del runtime dell'ambiente di esecuzione, al ciclo di vita delle estensioni e alle chiamate delle funzioni.
- Log delle funzioni: log personalizzati generati dal codice della funzione Lambda.
- Log di estensione: i log personalizzati generati dal codice di estensione Lambda.

### Note

Lambda invia log e metriche e tracce a X-Ray (se hai attivato il tracing) CloudWatch, anche se un'estensione si iscrive ai flussi di telemetria.

## Sections

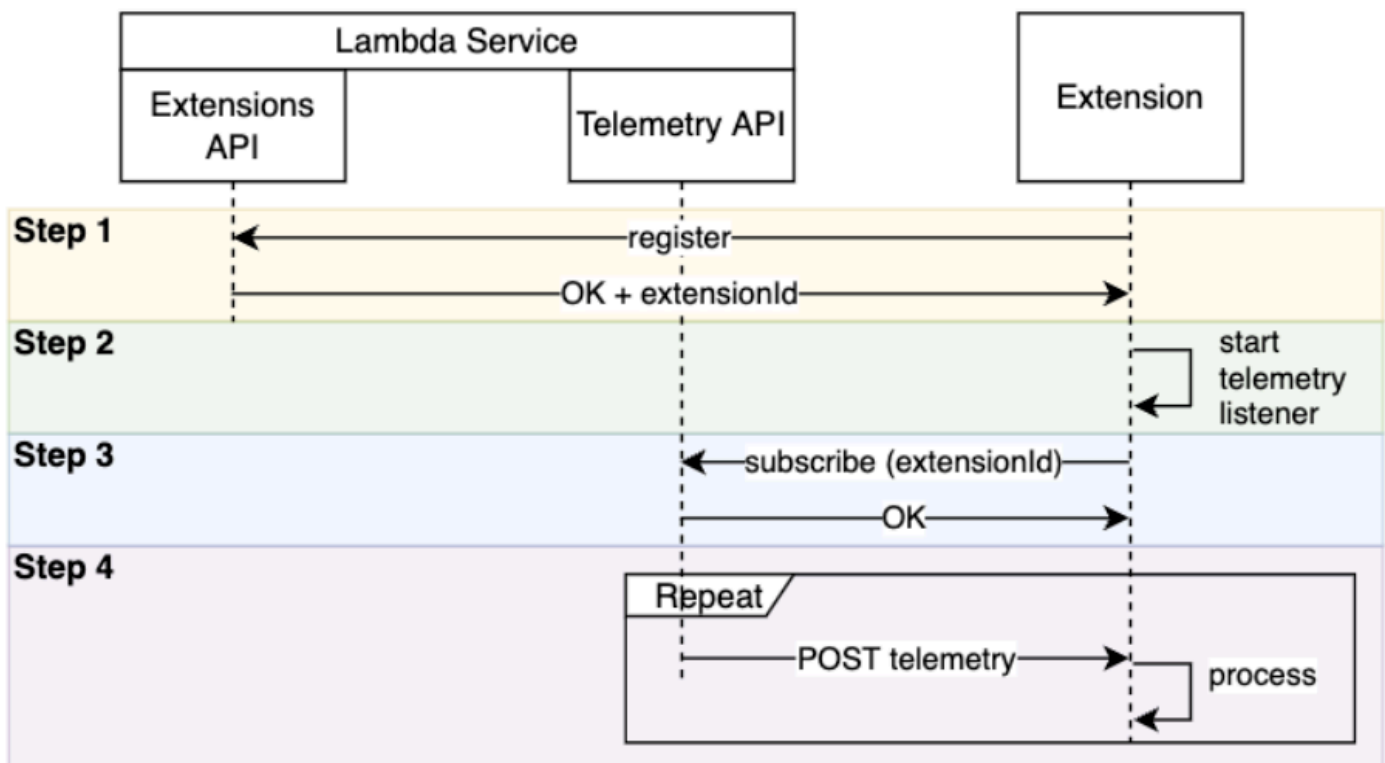
- [Creazione di estensioni tramite l'API di telemetria](#)
- [Registrazione delle estensioni](#)
- [Creazione di un listener di telemetria](#)
- [Specifica di un protocollo di destinazione](#)
- [Configurazione dell'utilizzo della memoria e del buffering](#)
- [Invio di una richiesta di sottoscrizione all'API di telemetria](#)
- [Messaggi dell'API di telemetria in entrata](#)

- [Riferimento all'API di telemetria Lambda](#)
- [Riferimento allo schema degli eventi dell'API di telemetria Lambda](#)
- [Conversione degli oggetti evento dell'API di telemetria Lambda in intervalli OpenTelemetry](#)
- [Utilizzo dell'API Logs di Lambda](#)

## Creazione di estensioni tramite l'API di telemetria

Le estensioni Lambda vengono eseguite come processi indipendenti nell'ambiente di esecuzione. L'esecuzione delle estensioni può proseguire dopo il completamento dell'invocazione della funzione. Poiché le estensioni vengono eseguite come processi separati, è possibile scriverle in un linguaggio diverso da quello del codice della funzione. Consigliamo di scrivere estensioni utilizzando un linguaggio compilato come Golang o Rust. In questo caso, l'estensione è un binario autonomo compatibile con qualsiasi runtime supportato.

Il diagramma seguente illustra un processo in quattro fasi per creare un'estensione che riceve ed elabora i dati di telemetria tramite l'API di telemetria.



Ecco ogni fase in modo più dettagliato:

1. Registra la tua estensione utilizzando l'[the section called “API estensioni”](#). Questo ti fornisce un `Lambda-Extension-Identifier`, di cui avrai bisogno nei passaggi seguenti. Per ulteriori informazioni su come eseguire la registrazione dell'estensione, consulta [the section called “Registrazione delle estensioni”](#).
2. Crea un listener di telemetria. Può trattarsi di un server HTTP o TCP di base. Lambda utilizza l'URI del listener di telemetria per inviare dati di telemetria all'estensione. Per ulteriori informazioni, consulta [the section called “Creazione di un listener di telemetria”](#).
3. Utilizzando l'API Subscribe nell'API Telemetry, esegui l'abbonamento all'estensione ai flussi di telemetria desiderati. Avrai bisogno dell'URI del tuo listener di telemetria per questo passaggio. Per ulteriori informazioni, consulta [the section called “Invio di una richiesta di sottoscrizione all'API di telemetria”](#).
4. Ottieni dati di telemetria da Lambda tramite il telemetry listener. Puoi eseguire qualsiasi elaborazione personalizzata di questi dati, ad esempio inviandoli ad Amazon S3 o a un servizio di osservabilità esterno.

#### Note

L'ambiente di esecuzione di una funzione Lambda può avviarsi e interrompersi più volte nell'ambito del suo [ciclo di vita](#). In generale, il codice dell'estensione viene eseguito durante le chiamate delle funzioni e anche fino a 2 secondi durante la fase di spegnimento. Consigliamo di raggruppare i dati di telemetria non appena arrivano all'listener. Quindi, utilizza gli eventi del ciclo di vita Invoke e Shutdown per inviare ogni batch alle destinazioni desiderate.

## Registrazione delle estensioni

Prima di poter eseguire l'abbonamento ai dati di telemetria, devi registrare l'estensione Lambda. La registrazione avviene durante la [fase di inizializzazione dell'estensione](#). L'esempio seguente mostra una richiesta HTTP per la registrazione di un'estensione.

```
POST http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
Lambda-Extension-Name: lambda_extension_name
{
  'events': [ 'INVOKE', 'SHUTDOWN' ]
}
```

Se la richiesta ha esito positivo, il sottoscrittore riceve una risposta riuscita HTTP 200. L'intestazione della risposta contiene l'`Lambda-Extension-Identifier`. Il corpo della risposta contiene altre proprietà della funzione.

```
HTTP/1.1 200 OK
Lambda-Extension-Identifier: a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
{
  "functionName": "lambda_function",
  "functionVersion": "$LATEST",
  "handler": "lambda_handler",
  "accountId": "123456789012"
}
```

Per ulteriori informazioni, consulta [the section called “Riferimento all'API delle estensioni”](#).

## Creazione di un listener di telemetria

L'estensione Lambda deve disporre di un listener che gestisca le richieste in entrata dall'API di telemetria. Il codice seguente mostra un esempio di implementazione dell'listener di telemetria in Golang:

```
// Starts the server in a goroutine where the log events will be sent
func (s *TelemetryApiListener) Start() (string, error) {
    address := listenOnAddress()
    l.Info("[listener:Start] Starting on address", address)
    s.httpServer = &http.Server{Addr: address}
    http.HandleFunc("/", s.http_handler)
    go func() {
        err := s.httpServer.ListenAndServe()
        if err != http.ErrServerClosed {
            l.Error("[listener:goroutine] Unexpected stop on Http Server:", err)
            s.Shutdown()
        } else {
            l.Info("[listener:goroutine] Http Server closed:", err)
        }
    }()
    return fmt.Sprintf("http://%s/", address), nil
}

// http_handler handles the requests coming from the Telemetry API.
// Everytime Telemetry API sends log events, this function will read them from the
// response body
```

```
// and put into a synchronous queue to be dispatched later.
// Logging or printing besides the error cases below is not recommended if you have
// subscribed to
// receive extension logs. Otherwise, logging here will cause Telemetry API to send new
// logs for
// the printed lines which may create an infinite loop.
func (s *TelemetryApiListener) http_handler(w http.ResponseWriter, r *http.Request) {
    body, err := ioutil.ReadAll(r.Body)
    if err != nil {
        l.Error("[listener:http_handler] Error reading body:", err)
        return
    }

    // Parse and put the log messages into the queue
    var slice []interface{}
    _ = json.Unmarshal(body, &slice)

    for _, el := range slice {
        s.LogEventsQueue.Put(el)
    }

    l.Info("[listener:http_handler] logEvents received:", len(slice), " LogEventsQueue
length:", s.LogEventsQueue.Len())
    slice = nil
}
```

## Specifica di un protocollo di destinazione

Quando esegui la sottoscrizione per ricevere la telemetria tramite l'API di telemetria, puoi specificare un protocollo di destinazione in aggiunta all'URI di destinazione:

```
{
  "destination": {
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080"
  }
}
```

Lambda accetta due protocolli per la ricezione della telemetria:

- HTTP (consigliato): Lambda consegna la telemetria a un endpoint HTTP locale (`http://sandbox.localdomain:${PORT}/${PATH}`) come matrice di record in formato JSON. Il

parametro `$PATH` è facoltativo. Lambda supporta solo HTTP, non HTTPS. Lambda fornisce la telemetria tramite richieste POST.

- TCP: Lambda consegna la telemetria a una porta TCP in [formato JSON delimitato da Newline \(NDJSON\)](#).

#### Note

Si consiglia di utilizzare HTTP anziché TCP. Con TCP, la piattaforma Lambda non può riconoscere che la telemetria viene consegnata al livello dell'applicazione. Pertanto, se l'estensione si blocca si potrebbero perdere i dati di telemetria. HTTP non condivide questa limitazione.

Prima di eseguire l'abbonamento per ricevere i dati di telemetria, definisci l'listener HTTP locale o la porta TCP. Durante l'installazione, tenere presente quanto segue:

- Lambda invia la telemetria solo alle destinazioni che si trovano all'interno dell'ambiente di esecuzione.
- Lambda riprova a inviare i dati di telemetria (con backoff) in assenza di un listener o se la richiesta POST subisce un errore. Se riporta un arresto anomalo, l'listener di telemetria continuerà a riceverla dopo che Lambda avrà riavviato l'ambiente di esecuzione.
- Lambda riserva la porta 9001. Non ci sono altre restrizioni o raccomandazioni sul numero di porta.

## Configurazione dell'utilizzo della memoria e del buffering

L'utilizzo della memoria in un ambiente di esecuzione aumenta linearmente con il numero di abbonati. Gli abbonamenti consumano risorse di memoria, perché ognuno apre un nuovo buffer di memoria per archiviare i dati di telemetria. L'utilizzo della memoria buffer contribuisce al consumo di memoria complessivo nell'ambiente di esecuzione.

Quando effettui l'abbonamento per ricevere la telemetria attraverso l'API Telemetry, hai la possibilità di eseguire il buffer dei dati di telemetria e consegnarli agli abbonati in batch. Per ottimizzare l'utilizzo della memoria, puoi specificare una configurazione di buffering:

```
{  
  "buffering": {
```



```

    "maxBytes": 256*1024,
    "maxItems": 1000,
    "timeoutMs": 100
  }
}

```

| Parametro | Description                                                                                | Valori predefiniti e limiti                                              |
|-----------|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| maxBytes  | Il volume massimo di dati di telemetria (in byte) da memorizzare nel buffer della memoria. | Predefinito: 262.144.<br><br>Minimo: 262.144.<br><br>Massimo: 1.048.576. |
| maxItems  | Il numero massimo di eventi da memorizzare nel buffer.                                     | Predefinito: 10.000<br><br>Minimo: 1.000<br><br>Massimo: 10.000.         |
| timeoutMs | Il tempo massimo (in millisecondi) per memorizzare nel buffer un batch.                    | Predefinito: 1.000<br><br>Minimo: 25<br><br>Massimo: 30.000              |

Quando configuri il buffering, tieni presente i seguenti punti:

- Se uno qualsiasi dei flussi di input è chiuso, Lambda svuota i log. Ad esempio, ciò si può verificare quando il runtime subisce un arresto anomalo.
- Ogni abbonato può personalizzare la configurazione di buffering nella richiesta di abbonamento.
- Al momento di determinare la dimensione di buffer per la lettura dei dati, prevedi di ricevere payload di dimensioni pari a  $2 * \text{maxBytes} + \text{metadataBytes}$ , dove maxBytes è un componente della configurazione di buffering. Per valutare la quantità di metadataBytes da considerare, esamina i seguenti metadati. Lambda allega metadati simili a questi a ogni record:

```

{
  "time": "2022-08-20T12:31:32.123Z",
  "type": "function",
  "record": "Hello World"
}

```

```
}
```

- Se il sottoscrittore non è in grado di elaborare la telemetria in ingresso abbastanza rapidamente o se il codice di funzione genera un volume di log molto elevato, Lambda potrebbe eliminare i registri per mantenere limitato l'utilizzo della memoria. Quando ciò si verifica, Lambda invia un evento `platform.logsDropped`.

## Invio di una richiesta di sottoscrizione all'API di telemetria

Un'estensione Lambda può eseguire la sottoscrizione per ricevere i dati di telemetria inviando una richiesta di sottoscrizione all'API di telemetria. La richiesta di sottoscrizione deve contenere informazioni sui tipi di eventi a cui si desidera che l'estensione si iscriva. Inoltre, la richiesta può contenere [informazioni sulla destinazione della consegna](#) e una [configurazione del buffering](#).

Prima di inviare una richiesta di sottoscrizione, è necessario disporre di un ID di estensione (Lambda-Extension-Identifier). Quando [registri l'estensione con l'API delle estensioni](#), ottieni un ID di estensione dalla risposta dell'API.

La registrazione avviene durante la [fase di inizializzazione dell'estensione](#). L'esempio seguente mostra una richiesta HTTP di sottoscrizione a tutti e tre i flussi di telemetria: telemetria della piattaforma, log delle funzioni e log delle estensioni.

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry HTTP/1.1
{
  "schemaVersion": "2025-01-29",
  "types": [
    "platform",
    "function",
    "extension"
  ],
  "buffering": {
    "maxItems": 1000,
    "maxBytes": 256*1024,
    "timeoutMs": 100
  },
  "destination": {
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080"
  }
}
```

Se la richiesta ha esito positivo, il sottoscrittore riceve una risposta do operazione riuscita HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

## Messaggi dell'API di telemetria in entrata

Dopo la sottoscrizione tramite l'API di telemetria, un'estensione inizia automaticamente a ricevere telemetria da Lambda tramite richieste POST. Ogni corpo di richiesta POST contiene una serie di oggetti Event. Ogni Event presenta il seguente schema:

```
{
  time: String,
  type: String,
  record: Object
}
```

- La proprietà `time` definisce il momento in cui la piattaforma Lambda ha generato l'evento. Si tratta di un valore diverso da quando l'evento si è verificato effettivamente. Il valore della stringa di `time` è un timestamp nel formato ISO 8601.
- La proprietà `type` definisce il tipo di evento. La seguente tabella riporta tutti i valori possibili.
- La proprietà `record` definisce un oggetto JSON che contiene i dati di telemetria. Lo schema di questo oggetto JSON dipende dal `type`.

### Ordinamento degli eventi con invocazioni simultanee

Per le istanze gestite [Lambda](#), è possibile eseguire più invocazioni di funzioni contemporaneamente all'interno dello stesso ambiente di esecuzione. In questo caso, l'ordine `platform.start` e `platform.report` gli eventi non sono garantiti tra diverse chiamate simultanee. Le estensioni devono gestire gli eventi provenienti da più invocazioni eseguite in parallelo e non devono assumere un ordinamento sequenziale.

Per attribuire correttamente gli eventi a chiamate specifiche, le estensioni devono utilizzare il `requestId` campo presente in questi eventi della piattaforma. Ogni chiamata ha un ID di richiesta univoco che rimane coerente in tutti gli eventi relativi a tale chiamata, consentendo alle estensioni di correlare correttamente gli eventi anche quando arrivano fuori ordine.

La tabella seguente riassume tutti i tipi di oggetti Event e i collegamenti al [riferimento dello schema Event dell'API di telemetria](#) per ogni tipo di evento.

| Categoria                | Tipo di evento                           | Description                                                                        | Schema dei registri di eventi                                         |
|--------------------------|------------------------------------------|------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| Evento della piattaforma | <code>platform.initStart</code>          | Inizializzazione della funzione avviata.                                           | Schema <a href="#">the section called “Platform.initStart ”</a>       |
| Evento della piattaforma | <code>platform.initRuntimeDone</code>    | Inizializzazione della funzione completata.                                        | Schema <a href="#">the section called “platform.initRuntimeDone ”</a> |
| Evento della piattaforma | <code>platform.initReport</code>         | Un rapporto sull'inizializzazione della funzione.                                  | Schema <a href="#">the section called “Rapporto Platform.init ”</a>   |
| Evento della piattaforma | <code>platform.start</code>              | L'invocazione della funzione è stata avviata.                                      | Schema <a href="#">the section called “platform.start ”</a>           |
| Evento della piattaforma | <code>platform.runtimeDone</code>        | Il runtime ha terminato l'elaborazione di un evento con esito positivo o negativo. | Schema <a href="#">the section called “platform.runtimeDone ”</a>     |
| Evento della piattaforma | <code>platform.report</code>             | Un rapporto sull'invocazione di una funzione.                                      | Schema <a href="#">the section called “platform.report ”</a>          |
| Evento della piattaforma | <code>platform.restoreStart</code>       | Il ripristino del runtime è iniziato.                                              | Schema <a href="#">the section called “Platform.RestoreStart ”</a>    |
| Evento della piattaforma | <code>platform.restoreRuntimeDone</code> | Il ripristino del runtime è stato completato.                                      | Schema <a href="#">the section called “piattafor</a>                  |

| Categoria                | Tipo di evento                 | Description                                                           | Schema dei registri di eventi                                                                                      |
|--------------------------|--------------------------------|-----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| Evento della piattaforma | platform.restoreReport         | Rapporto di ripristino del runtime.                                   | <a href="#">ma. Restore RuntimeDone</a> ”<br>Schema <a href="#">the section called “Rapporto Platform.restore”</a> |
| Evento della piattaforma | platform.telemetrySubscription | L'estensione che ha eseguito la sottoscrizione all'API di telemetria. | Schema <a href="#">the section called “Abbonamento a platform.telemetry”</a>                                       |
| Evento della piattaforma | platform.logsDropped           | Lambda ha eliminato le voci del log.                                  | Schema <a href="#">the section called “Platform.logs è stato eliminato”</a>                                        |
| Log delle funzioni       | function                       | Una riga del log dal codice della funzione.                           | Schema <a href="#">the section called “funzione”</a>                                                               |
| Log di estensioni        | extension                      | Una riga di log dal codice dell'estensione.                           | Schema <a href="#">the section called “Estensione”</a>                                                             |

## Riferimento all'API di telemetria Lambda

Utilizza l'endpoint dell'API di telemetria Lambda per sottoscrivere le estensioni ai flussi di telemetria. È possibile recuperare l'endpoint dell'API di telemetria dalla variabile di ambiente `AWS_LAMBDA_RUNTIME_API`. Per inviare una richiesta API, aggiungi la versione dell'API (2022-07-01/) e `telemetry/`. Esempio:

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Per la definizione della specifica OpenAPI (OAS) della versione delle risposte alla sottoscrizione 2025-01-29, consulta quanto segue:

- HTTP — [telemetry-api-http-schema.zip](#)
- TCP — [.zip telemetry-api-tcp-schema](#)

operazioni API

- [Subscribe](#)

### Subscribe

Per sottoscrivere un flusso di telemetria, un'estensione Lambda può inviare una richiesta API `Subscribe`.

- Percorso – `/telemetry`
- Metodo: `PUT`
- Headers
  - `Content-Type: application/json`
- Parametri del corpo della richiesta
  - `schemaVersion`
    - Obbligatorio: sì
    - Tipo: `String`
    - Valori validi: `"2025-01-29"`, `"2022-12-13"` o `"2022-07-01"`
    - Nota: le istanze gestite Lambda richiedono. `"2025-01-29"` Questa versione è retrocompatibile con le funzioni Lambda (predefinite).

- **destinazione:** le impostazioni di configurazione che definiscono la destinazione dell'evento di telemetria e il protocollo per la consegna degli eventi.
- **Obbligatorio:** sì
- **Tipo:** oggetto

```
{
  "protocol": "HTTP",
  "URI": "http://sandbox.localdomain:8080"
}
```

- **protocollo:** il protocollo utilizzato da Lambda per inviare dati di telemetria.
- **Obbligatorio:** sì
- **Tipo:** String
- **Valori validi:** "HTTP"|"TCP"
- **URI:** l'URI a cui inviare i dati di telemetria.
- **Obbligatorio:** sì
- **Tipo:** String
- Per ulteriori informazioni, consulta [the section called “Specifica di un protocollo di destinazione”](#).
- **tipi:** i tipi di dati di telemetria che desideri siano sottoscritti dall'estensione.
- **Obbligatorio:** sì
- **Tipo:** array di stringhe
- **Valori validi:** "platform"|"function"|"extension"
- **buffering:** le impostazioni di configurazione per il buffering degli eventi.
- **Obbligatorio:** no
- **Tipo:** oggetto

```
{
  "buffering": {
    "maxItems": 1000,
    "maxBytes": 256*1024,
    "timeoutMs": 100
  }
}
```

- Obbligatorio: no
  - Tipo: numero intero
  - Predefinito: 1.000
  - Minimo: 1.000
  - Massimo: 10.000.
- maxBytes: il volume massimo di dati di telemetria (in byte) da memorizzare nel buffer della memoria.
    - Obbligatorio: no
    - Tipo: numero intero
    - Predefinito: 262.144.
    - Minimo: 262.144.
    - Massimo: 1.048.576.
  - timeoutMs – Il tempo massimo (in millisecondi) per il buffer di un batch.
    - Obbligatorio: no
    - Tipo: numero intero
    - Predefinito: 1.000
    - Minimo: 25
    - Massimo: 30.000
  - Per ulteriori informazioni, consulta [the section called “Configurazione dell'utilizzo della memoria e del buffering”](#).

### Esempio di richiesta API Subscribe

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry HTTP/1.1
{
  "schemaVersion": "2025-01-29",
  "types": [
    "platform",
    "function",
    "extension"
  ],
  "buffering": {
    "maxItems": 1000,
    "maxBytes": 256*1024,
    "timeoutMs": 100
  }
}
```



```
  },  
  "destination": {  
    "protocol": "HTTP",  
    "URI": "http://sandbox.localdomain:8080"  
  }  
}
```

Se la richiesta Subscribe ha esito positivo, il sottoscrittore riceve una risposta di operazione riuscita HTTP 200.

```
HTTP/1.1 200 OK  
"OK"
```

Se la richiesta non riesce, l'estensione riceve una risposta di errore. Esempio:

```
HTTP/1.1 400 OK  
{  
  "errorType": "ValidationError",  
  "errorMessage": "URI port is not provided; types should not be empty"  
}
```

Ecco alcuni codici di risposta aggiuntivi che l'estensione può ricevere:

- 200 – Richiesta completata con successo
- 202 – Richiesta accettata. Risposta alla richiesta di sottoscrizione nell'ambiente di test locale
- 400: richiesta non valida
- 500 – Errore servizio

## Riferimento allo schema degli eventi dell'API di telemetria Lambda

Utilizza l'endpoint dell'API di telemetria Lambda per sottoscrivere le estensioni ai flussi di telemetria. È possibile recuperare l'endpoint dell'API di telemetria dalla variabile di ambiente `AWS_LAMBDA_RUNTIME_API`. Per inviare una richiesta API, aggiungi la versione dell'API (2022-07-01/) e `telemetry/`. Ad esempio:

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Per la definizione della specifica OpenAPI (OAS) della versione delle risposte alla sottoscrizione 2025-01-29, consulta quanto segue:

- HTTP: [telemetry-api-http-schema.zip](#)
- TCP: [telemetry-api-tcp-schema.zip](#)

La tabella seguente è un riepilogo di tutti i tipi di oggetti Event supportati dall'API di telemetria.

| Categoria                | Tipo di evento                        | Description                                       | Schema dei registri di eventi                                        |
|--------------------------|---------------------------------------|---------------------------------------------------|----------------------------------------------------------------------|
| Evento della piattaforma | <code>platform.initStart</code>       | Inizializzazione della funzione avviata.          | Schema <a href="#">the section called “Platform.initStart”</a>       |
| Evento della piattaforma | <code>platform.initRuntimeDone</code> | Inizializzazione della funzione completata.       | Schema <a href="#">the section called “platform.initRuntimeDone”</a> |
| Evento della piattaforma | <code>platform.initReport</code>      | Un rapporto sull'inizializzazione della funzione. | Schema <a href="#">the section called “Rapporto Platform.init”</a>   |
| Evento della piattaforma | <code>platform.start</code>           | L'invocazione della funzione è stata avviata.     | Schema <a href="#">the section called “platform.start”</a>           |

| Categoria                | Tipo di evento                 | Description                                                                        | Schema dei registri di eventi                                                |
|--------------------------|--------------------------------|------------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| Evento della piattaforma | platform.runtimeDone           | Il runtime ha terminato l'elaborazione di un evento con esito positivo o negativo. | Schema <a href="#">the section called “platform.runtimeDone”</a>             |
| Evento della piattaforma | platform.report                | Un rapporto sull'invocazione di una funzione.                                      | Schema <a href="#">the section called “platform.report”</a>                  |
| Evento della piattaforma | platform.restoreStart          | Il ripristino del runtime è iniziato.                                              | Schema <a href="#">the section called “Platform.RestoreStart”</a>            |
| Evento della piattaforma | platform.restoreRuntimeDone    | Il ripristino del runtime è stato completato.                                      | Schema <a href="#">the section called “piattaforma.RestoreRuntimeDone”</a>   |
| Evento della piattaforma | platform.restoreReport         | Rapporto di ripristino del runtime.                                                | Schema <a href="#">the section called “Rapporto Platform.restore”</a>        |
| Evento della piattaforma | platform.telemetrySubscription | L'estensione che ha eseguito la sottoscrizione all'API di telemetria.              | Schema <a href="#">the section called “Abbonamento a platform.telemetry”</a> |
| Evento della piattaforma | platform.logsDropped           | Lambda ha eliminato le voci del log.                                               | Schema <a href="#">the section called “Platform.logs è stato eliminato”</a>  |
| Log delle funzioni       | function                       | Una riga del log dal codice della funzione.                                        | Schema <a href="#">the section called “funzione”</a>                         |

| Categoria         | Tipo di evento | Description                                 | Schema dei registri di eventi                            |
|-------------------|----------------|---------------------------------------------|----------------------------------------------------------|
| Log di estensioni | extension      | Una riga di log dal codice dell'estensione. | Schema <a href="#">the section called “Extension e ”</a> |

## Indice

- [Tipi di oggetti Event dell'API di telemetria](#)
  - [Platform.initStart](#)
  - [piattaforma. init RuntimeDone](#)
  - [Rapporto Platform.init](#)
  - [platform.start](#)
  - [platform.runtimeDone](#)
  - [platform.report](#)
  - [Platform.RestoreStart](#)
  - [piattaforma. Restore RuntimeDone](#)
  - [Rapporto Platform.restore](#)
  - [platform.estensione](#)
  - [Abbonamento a platform.telemetry](#)
  - [Platform.logs è stato eliminato](#)
  - [funzione](#)
  - [Estensione](#)
- [Tipi di oggetti condivisi](#)
  - [InitPhase](#)
  - [InitReportMetrics](#)
  - [InitType](#)
  - [ReportMetrics](#)
  - [RestoreReportMetrics](#)
  - [RuntimeDoneMetrics](#)

- [Stato](#)
- [TraceContext](#)
- [TracingType](#)

## Tipi di oggetti Event dell'API di telemetria

Questa sezione descrive in dettaglio i tipi di oggetti Event supportati dall'API di telemetria Lambda. Nelle descrizioni degli eventi, un punto interrogativo (?) indica che l'attributo potrebbe non essere presente nell'oggetto.

### **Platform.initStart**

Un evento `platform.initStart` indica che la fase di inizializzazione della funzione è stata avviata. Un oggetto `platform.initStart` Event ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.initStart
- record: PlatformInitStart
```

L'oggetto `PlatformInitStart` ha i seguenti attributi:

- `functionName`: String
- `functionVersion`: String
- `initializationType`: oggetto [the section called "InitType"](#)
- `instanceId?`: String
- `istanza? MaxMemory` — Integer
- `phase`: oggetto [the section called "InitPhase"](#)
- `runtimeVersion?`: String
- `tempo di esecuzioneVersionArn?` — String

Di seguito è riportato un esempio Event di tipo `platform.initStart`:

```
{
  "time": "2022-10-12T00:00:15.064Z",
  "type": "platform.initStart",
  "record": {
```

```

    "initializationType": "on-demand",
    "phase": "init",
    "runtimeVersion": "nodejs-14.v3",
    "runtimeVersionArn": "arn",
    "functionName": "myFunction",
    "functionVersion": "$LATEST",
    "instanceId": "82561ce0-53dd-47d1-90e0-c8f5e063e62e",
    "instanceMaxMemory": 256
  }
}

```

## piattaforma. init RuntimeDone

Un evento `platform.initRuntimeDone` indica che la fase di inizializzazione della funzione è stata completata. Un oggetto `platform.initRuntimeDone` Event ha la seguente forma:

```

Event: Object
- time: String
- type: String = platform.initRuntimeDone
- record: PlatformInitRuntimeDone

```

L'oggetto `PlatformInitRuntimeDone` ha i seguenti attributi:

- `initializationType`: oggetto [the section called "InitType"](#)
- `phase`: oggetto [the section called "InitPhase"](#)
- `status`: oggetto [the section called "Stato"](#)
- `spans?`: elenco di oggetti [the section called "Intervallo"](#)

Di seguito è riportato un esempio Event di tipo `platform.initRuntimeDone`:

```

{
  "time": "2022-10-12T00:01:15.000Z",
  "type": "platform.initRuntimeDone",
  "record": {
    "initializationType": "on-demand"
    "status": "success",
    "spans": [
      {
        "name": "someTimeSpan",
        "start": "2022-06-02T12:02:33.913Z",

```

```

        "durationMs": 70.5
      }
    ]
  }
}

```

## Rapporto Platform.init

Un evento `platform.initReport` contiene un rapporto generale della fase di inizializzazione della funzione. Un oggetto `platform.initReport` Event ha la seguente forma:

```

Event: Object
- time: String
- type: String = platform.initReport
- record: PlatformInitReport

```

L'oggetto `PlatformInitReport` ha i seguenti attributi:

- `errorType?`: stringa
- `initializationType`: oggetto [the section called "InitType"](#)
- `phase`: oggetto [the section called "InitPhase"](#)
- `metrics`: oggetto [the section called "InitReportMetrics"](#)
- `spans?`: elenco di oggetti [the section called "Intervallo"](#)
- `status`: oggetto [the section called "Stato"](#)

Di seguito è riportato un esempio Event di tipo `platform.initReport`:

```

{
  "time": "2022-10-12T00:01:15.000Z",
  "type": "platform.initReport",
  "record": {
    "initializationType": "on-demand",
    "status": "success",
    "phase": "init",
    "metrics": {
      "durationMs": 125.33
    },
    "spans": [
      {

```

```

        "name": "someTimeSpan",
        "start": "2022-06-02T12:02:33.913Z",
        "durationMs": 90.1
      }
    ]
  }
}

```

## platform.start

Un evento `platform.start` indica che la fase di invocazione della funzione è stata avviata. Un oggetto `platform.start` Event ha la seguente forma:

```

Event: Object
- time: String
- type: String = platform.start
- record: PlatformStart

```

L'oggetto `PlatformStart` ha i seguenti attributi:

- `requestId`: String
- `version?`: String
- `tracing?`: [the section called "TraceContext"](#)

Di seguito è riportato un esempio Event di tipo `platform.start`:

```

{
  "time": "2022-10-12T00:00:15.064Z",
  "type": "platform.start",
  "record": {
    "requestId": "6d68ca91-49c9-448d-89b8-7ca3e6dc66aa",
    "version": "$LATEST",
    "tracing": {
      "spanId": "54565fb41ac79632",
      "type": "X-Amzn-Trace-Id",
      "value":
"Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
    }
  }
}

```



## platform.runtimeDone

Un evento `platform.runtimeDone` indica che la fase di invocazione della funzione è stata completata. Un oggetto `platform.runtimeDone` Event ha la seguente forma:

### Istanze Lambda gestite

L'evento `platform.runtimeDone` non è supportato per le istanze gestite Lambda. Le estensioni in esecuzione su istanze gestite non ricevono questo evento perché le estensioni non possono iscriversi all'evento `INVOKED` su istanze gestite. A causa del modello di esecuzione simultanea in cui è possibile elaborare più invocazioni contemporaneamente, le estensioni non possono eseguire l'elaborazione post-richiamo per le singole chiamate come avviene tradizionalmente sulle funzioni Lambda (predefinite).

Per le istanze gestite, gli intervalli `responseDuration` `responseLatency` e normalmente inclusi sono invece disponibili nell'evento `platform.runtimeDone` `platform.report`. Per informazioni dettagliate, vedi [the section called "platform.report"](#).

```
Event: Object
- time: String
- type: String = platform.runtimeDone
- record: PlatformRuntimeDone
```

L'oggetto `PlatformRuntimeDone` ha i seguenti attributi:

- `errorType?: String`
- `metrics?:` oggetto [the section called "RuntimeDoneMetrics"](#)
- `requestId: String`
- `status:` oggetto [the section called "Stato"](#)
- `spans? :` elenco di oggetti [the section called "Intervallo"](#)
- `tracing?:` oggetto [the section called "TraceContext"](#)

Di seguito è riportato un esempio Event di tipo `platform.runtimeDone`:

```
{
  "time": "2022-10-12T00:01:15.000Z",
  "type": "platform.runtimeDone",
```

```

    "record": {
      "requestId": "6d68ca91-49c9-448d-89b8-7ca3e6dc66aa",
      "status": "success",
      "tracing": {
        "spanId": "54565fb41ac79632",
        "type": "X-Amzn-Trace-Id",
        "value":
"Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
      },
      "spans": [
        {
          "name": "someTimeSpan",
          "start": "2022-08-02T12:01:23:521Z",
          "durationMs": 80.0
        }
      ],
      "metrics": {
        "durationMs": 140.0,
        "producedBytes": 16
      }
    }
  }
}

```

## platform.report

Un evento `platform.report` contiene un rapporto generale della fase di invocazione della funzione. Un oggetto `platform.report Event` ha la seguente forma:

### Istanze Lambda gestite

L'evento `platform.report` per le istanze gestite Lambda ha metriche e intervalli diversi rispetto alle funzioni Lambda (predefinite). Per le istanze gestite:

- **Intervallo:** contiene `responseLatency` e `responseDuration` invece di `extensionOverhead`. L'intervallo `extensionOverhead` non è disponibile perché le estensioni non possono sottoscrivere l'evento `INVOKED` sulle istanze gestite a causa del modello di esecuzione simultanea.
- **Metriche:** include solo `durationMs`. Le seguenti metriche non sono incluse: `billedDurationMs`, `initDurationMs`, `maxMemoryUsedMB`, e `memorySizeMB`. Queste metriche per richiamo non sono applicabili nell'ambiente di esecuzione simultanea. Per le metriche sull'utilizzo delle risorse, utilizza o Lambda Insights. [Monitoraggio delle](#)

istanze gestite Lambda <https://docs.aws.amazon.com/lambda/latest/dg/monitoring-insights.html>

```
Event: Object
- time: String
- type: String = platform.report
- record: PlatformReport
```

L'oggetto PlatformReport ha i seguenti attributi:

- metrics: oggetto [the section called "ReportMetrics"](#)
- requestId: String
- spans?: elenco di oggetti [the section called "Intervallo"](#)
- status: oggetto [the section called "Stato"](#)
- tracing?: oggetto [the section called "TraceContext"](#)

Di seguito è riportato un esempio Event di tipo platform.report:

```
{
  "time": "2022-10-12T00:01:15.000Z",
  "type": "platform.report",
  "record": {
    "metrics": {
      "billedDurationMs": 694,
      "durationMs": 693.92,
      "initDurationMs": 397.68,
      "maxMemoryUsedMB": 84,
      "memorySizeMB": 128
    },
    "requestId": "6d68ca91-49c9-448d-89b8-7ca3e6dc66aa",
  }
}
```

## Platform.RestoreStart

Un evento platform.restoreStart indica che un evento di ripristino dell'ambiente della funzione è stato avviato. In un evento di ripristino dell'ambiente, Lambda crea l'ambiente da uno snapshot

memorizzato nella cache anziché iniziarlo da zero. Per ulteriori informazioni, consulta [SnapStart](#). Un oggetto `platform.restoreStart` Event ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.restoreStart
- record: PlatformRestoreStart
```

L'oggetto `PlatformRestoreStart` ha i seguenti attributi:

- `functionName`: String
- `functionVersion`: String
- `instanceId?`: String
- `istanzaMaxMemory?` — String
- `runtimeVersion?`: String
- `tempo di esecuzioneVersionArn?` — String

Di seguito è riportato un esempio Event di tipo `platform.restoreStart`:

```
{
  "time": "2022-10-12T00:00:15.064Z",
  "type": "platform.restoreStart",
  "record": {
    "runtimeVersion": "nodejs-14.v3",
    "runtimeVersionArn": "arn",
    "functionName": "myFunction",
    "functionVersion": "$LATEST",
    "instanceId": "82561ce0-53dd-47d1-90e0-c8f5e063e62e",
    "instanceMaxMemory": 256
  }
}
```

## piattaforma. Restore RuntimeDone

Un evento `platform.restoreRuntimeDone` indica che un evento di ripristino dell'ambiente della funzione è stato completato. In un evento di ripristino dell'ambiente, Lambda crea l'ambiente da uno snapshot memorizzato nella cache anziché iniziarlo da zero. Per ulteriori informazioni, consulta [SnapStart](#). Un oggetto `platform.restoreRuntimeDone` Event ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.restoreRuntimeDone
- record: PlatformRestoreRuntimeDone
```

L'oggetto PlatformRestoreRuntimeDone ha i seguenti attributi:

- `errorType?: String`
- `spans? :` elenco di oggetti [the section called "Intervallo"](#)
- `status:` oggetto [the section called "Stato"](#)

Di seguito è riportato un esempio Event di tipo `platform.restoreRuntimeDone`:

```
{
  "time": "2022-10-12T00:00:15.064Z",
  "type": "platform.restoreRuntimeDone",
  "record": {
    "status": "success",
    "spans": [
      {
        "name": "someTimeSpan",
        "start": "2022-08-02T12:01:23:521Z",
        "durationMs": 80.0
      }
    ]
  }
}
```

## Rapporto Platform.restore

Un evento `platform.restoreReport` contiene un rapporto generale di un evento di ripristino della funzione. Un oggetto `platform.restoreReport` Event ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.restoreReport
- record: PlatformRestoreReport
```

L'oggetto PlatformRestoreReport ha i seguenti attributi:

- `errorType?`: stringa
- `metrics?`: oggetto [the section called "RestoreReportMetrics"](#)
- `spans?`: elenco di oggetti [the section called "Intervallo"](#)
- `status`: oggetto [the section called "Stato"](#)

Di seguito è riportato un esempio Event di tipo `platform.restoreReport`:

```
{
  "time": "2022-10-12T00:00:15.064Z",
  "type": "platform.restoreReport",
  "record": {
    "status": "success",
    "metrics": {
      "durationMs": 15.19
    },
    "spans": [
      {
        "name": "someTimeSpan",
        "start": "2022-08-02T12:01:23:521Z",
        "durationMs": 30.0
      }
    ]
  }
}
```

## platform.estensione

Un evento `extension` contiene i log del codice dell'estensione. Un oggetto `extension Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = extension
- record: {}
```

L'oggetto `PlatformExtension` ha i seguenti attributi:

- `events`: elenco di `String`
- `name`: `String`
- `state`: `String`

Di seguito è riportato un esempio Event di tipo `platform.extension`:

```
{
  "time": "2022-10-12T00:02:15.000Z",
  "type": "platform.extension",
  "record": {
    "events": [ "INVOKE", "SHUTDOWN" ],
    "name": "my-telemetry-extension",
    "state": "Ready"
  }
}
```

### Abbonamento a `platform.telemetry`

Un evento `platform.telemetrySubscription` contiene informazioni su una sottoscrizione dell'estensione. Un oggetto `platform.telemetrySubscription` Event ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.telemetrySubscription
- record: PlatformTelemetrySubscription
```

L'oggetto `PlatformTelemetrySubscription` ha i seguenti attributi:

- `name: String`
- `state: String`
- `tipi: elenco di String`

Di seguito è riportato un esempio Event di tipo `platform.telemetrySubscription`:

```
{
  "time": "2022-10-12T00:02:35.000Z",
  "type": "platform.telemetrySubscription",
  "record": {
    "name": "my-telemetry-extension",
    "state": "Subscribed",
    "types": [ "platform", "function" ]
  }
}
```

## Platform.logs è stato eliminato

Un evento `platform.logsDropped` contiene informazioni sugli eventi eliminati. Quando una funzione genera log a una velocità troppo elevata per consentire a Lambda di elaborarli, Lambda emette l'evento `platform.logsDropped`. Quando Lambda non è in grado di inviare log a CloudWatch o all'estensione sottoscritta all'API di Telemetry alla velocità con cui la funzione li produce, elimina i log per evitare che l'esecuzione della funzione rallenti. Un oggetto `platform.logsDropped` Event ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.logsDropped
- record: PlatformLogsDropped
```

L'oggetto `PlatformLogsDropped` ha i seguenti attributi:

- `droppedBytes`: Integer
- `droppedRecords`: Integer
- `reason`: String

Di seguito è riportato un esempio Event di tipo `platform.logsDropped`:

```
{
  "time": "2022-10-12T00:02:35.000Z",
  "type": "platform.logsDropped",
  "record": {
    "droppedBytes": 12345,
    "droppedRecords": 123,
    "reason": "Some logs were dropped because the downstream consumer is slower than the logs production rate"
  }
}
```

## funzione

Un evento `function` contiene i log del codice della funzione. Un oggetto `function` Event ha la seguente forma:

```
Event: Object
- time: String
```



```
- type: String = function
- record: {}
```

Il formato del campo `record` dipende dal fatto che i log della funzione siano formattati in testo normale o JSON. Per ulteriori informazioni sulle opzioni di configurazione del formato dei log, consulta la pagina [the section called “Formati di registro”](#)

Di seguito è riportato un esempio di Event di tipo `function` in cui il formato di log è il testo normale:

```
{
  "time": "2022-10-12T00:03:50.000Z",
  "type": "function",
  "record": "[INFO] Hello world, I am a function!"
}
```

Di seguito è riportato un esempio di Event di tipo `function` in cui il formato di log è JSON:

```
{
  "time": "2022-10-12T00:03:50.000Z",
  "type": "function",
  "record": {
    "timestamp": "2022-10-12T00:03:50.000Z",
    "level": "INFO",
    "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189",
    "message": "Hello world, I am a function!"
  }
}
```

#### Note

Se la versione dello schema che stai utilizzando è precedente alla versione 2022-12-13, il `"record"` viene sempre visualizzato come stringa anche quando il formato di registrazione della funzione è configurato come JSON. Per le istanze gestite da Lambda, è necessario utilizzare la versione dello schema. 2025-01-29

## Estensione

Un evento `extension` contiene i log del codice dell'estensione. Un oggetto `extension Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = extension
- record: {}
```

Il formato del campo `record` dipende dal fatto che i log della funzione siano formattati in testo normale o JSON. Per ulteriori informazioni sulle opzioni di configurazione del formato dei log, consulta la pagina [the section called “Formati di registro”](#)

Di seguito è riportato un esempio di Event di tipo `extension` in cui il formato di log è il testo normale:

```
{
  "time": "2022-10-12T00:03:50.000Z",
  "type": "extension",
  "record": "[INFO] Hello world, I am an extension!"
}
```

Di seguito è riportato un esempio di Event di tipo `extension` in cui il formato di log è JSON:

```
{
  "time": "2022-10-12T00:03:50.000Z",
  "type": "extension",
  "record": {
    "timestamp": "2022-10-12T00:03:50.000Z",
    "level": "INFO",
    "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189",
    "message": "Hello world, I am an extension!"
  }
}
```

#### Note

Se la versione dello schema che stai utilizzando è precedente alla versione 2022-12-13, il `"record"` viene sempre visualizzato come stringa anche quando il formato di registrazione della funzione è configurato come JSON. Per le istanze gestite Lambda, è necessario utilizzare la versione dello schema. 2025-01-29

## Tipi di oggetti condivisi

Questa sezione descrive in dettaglio i tipi di oggetti condivisi supportati dall'API di telemetria Lambda.

### InitPhase

Una stringa enum che descrive la fase in cui si verifica l'operazione di inizializzazione. Nella maggior parte dei casi, Lambda esegue il codice di inizializzazione della funzione durante la fase `init`. Tuttavia, in alcuni casi di errore, Lambda può eseguire nuovamente il codice di inizializzazione della funzione durante la fase `invoke`. (Questa è chiamata `init` soppressa.)

- Tipo: `String`
- Valori validi: `init|invoke|snap-start`

### InitReportMetrics

Un oggetto che contiene parametri relativi a una fase di inizializzazione.

- Tipo: `Object`

Un oggetto `InitReportMetrics` ha la seguente forma:

```
InitReportMetrics: Object
- durationMs: Double
```

Di seguito è illustrato un esempio di oggetto `InitReportMetrics` di esempio:

```
{
  "durationMs": 247.88
}
```

### InitType

Una stringa enum che descrive come Lambda ha inizializzato l'ambiente.

- Tipo: `String`
- Valori validi: `on-demand|provisioned-concurrency`

## ReportMetrics

Un oggetto che contiene i parametri di una fase completata.

- Tipo: Object

Un oggetto ReportMetrics ha la seguente forma:

```
ReportMetrics: Object
- billedDurationMs: Integer
- durationMs: Double
- initDurationMs?: Double
- maxMemoryUsedMB: Integer
- memorySizeMB: Integer
- restoreDurationMs?: Double
```

Di seguito è illustrato un esempio di oggetto ReportMetrics di esempio:

```
{
  "billedDurationMs": 694,
  "durationMs": 693.92,
  "initDurationMs": 397.68,
  "maxMemoryUsedMB": 84,
  "memorySizeMB": 128
}
```

## RestoreReportMetrics

Un oggetto che contiene i parametri di una fase di ripristino completata.

- Tipo: Object

Un oggetto RestoreReportMetrics ha la seguente forma:

```
RestoreReportMetrics: Object
- durationMs: Double
```

Di seguito è illustrato un esempio di oggetto RestoreReportMetrics di esempio:

```
{
  "durationMs": 15.19
}
```

```
}
```

## RuntimeDoneMetrics

Un oggetto che contiene parametri relativi a una fase di chiamata.

- Tipo: Object

Un oggetto `RuntimeDoneMetrics` ha la seguente forma:

```
RuntimeDoneMetrics: Object
- durationMs: Double
- producedBytes?: Integer
```

Di seguito è illustrato un esempio di oggetto `RuntimeDoneMetrics` di esempio:

```
{
  "durationMs": 200.0,
  "producedBytes": 15
}
```

## Intervallo

Un oggetto che contiene i dettagli di un intervallo. Un intervallo rappresenta un'unità di lavoro o un'operazione in una traccia. Per ulteriori informazioni sugli intervalli, consulta [Span](#) nella pagina Tracing API del sito Web Docs. OpenTelemetry

Lambda supporta i seguenti intervalli per l'evento `platform.RuntimeDone`:

- L'intervallo `responseLatency` descrive il tempo impiegato dalla funzione Lambda per iniziare a inviare la risposta.
- L'intervallo `responseDuration` descrive il tempo impiegato dalla funzione Lambda per finire di inviare la risposta.
- L'intervallo `runtimeOverhead` descrive il tempo impiegato dal runtime di Lambda per segnalare che è pronto per l'elaborazione del richiamo successivo della funzione. Questo è il tempo impiegato dal runtime per il richiamo dell'API dell'[invocazione successiva](#) per ottenere l'evento successivo dopo aver restituito la risposta della funzione.

Di seguito è illustrato un oggetto di intervallo `responseLatency` di esempio:

```
{
  "name": "responseLatency",
  "start": "2022-08-02T12:01:23.521Z",
  "durationMs": 23.02
}
```

## Stato

Un oggetto che descrive lo stato di una fase di inizializzazione o invocazione. Se lo stato è `failure` o `error`, l'oggetto `Status` contiene anche un campo `errorType` che descrive l'errore.

- Tipo: `Object`
- Valori di stato validi: `success|failure|error|timeout`

## TraceContext

Un oggetto che descrive le proprietà di una traccia.

- Tipo: `Object`

Un oggetto `TraceContext` ha la seguente forma:

```
TraceContext: Object
- spanId?: String
- type: TracingType enum
- value: String
```

Di seguito è illustrato un esempio di oggetto `TraceContext` di esempio:

```
{
  "spanId": "073a49012f3c312e",
  "type": "X-Amzn-Trace-Id",
  "value":
    "Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
}
```

## TracingType

Una stringa enum che descrive il tipo di tracciamento in un oggetto [the section called "TraceContext"](#).

- Tipo: `String`
- Valori validi: `X-Amzn-Trace-Id`

## Conversione degli oggetti evento dell'API di telemetria Lambda in intervalli OpenTelemetry

Lo schema dell'API di AWS Lambda telemetria è semanticamente compatibile con (Otel).

OpenTelemetry Ciò significa che puoi convertire gli oggetti dell'API di AWS Lambda telemetria in intervalli (Otel). Event OpenTelemetry Durante la conversione, non dovresti mappare un singolo oggetto Event a un singolo intervallo OTel.

Dovreste invece presentare tutti e tre gli eventi relativi a una fase del ciclo di vita in un singolo intervallo OTel. Ad esempio, gli eventi `start`, `runtimeDone` e `runtimeReport` rappresentano una singola chiamata di funzione. Presenta tutti e tre questi eventi come un unico intervallo OTel.

Puoi convertire i tuoi eventi usando Span Events o Child Spans (nidificati). Le tabelle di questa pagina descrivono le mappature tra le proprietà dello schema dell'API di telemetria e le proprietà dell'intervallo OTel per entrambi gli approcci. Per ulteriori informazioni su Otel Spans, vedi [Span nella pagina Tracing API del](#) sito Web Docs. OpenTelemetry

### Sections

- [Mappa a intervalli OTel con eventi di intervallo](#)
- [Mappa a intervalli OTel con intervalli secondari](#)

### Mappa a intervalli OTel con eventi di intervallo

Nelle seguenti tabelle, e rappresenta l'evento proveniente dall'origine di telemetria.

#### Mappatura degli eventi \*Start

| OpenTelemetry               | Schema dell'API di telemetria Lambda                                        |
|-----------------------------|-----------------------------------------------------------------------------|
| <code>Span.Name</code>      | La tua estensione genera questo valore in base al campo <code>type</code> . |
| <code>Span.StartTime</code> | Utilizza <code>e.time</code> .                                              |
| <code>Span.EndTime</code>   | N/A, perché l'evento non è ancora completato.                               |
| <code>Span.Kind</code>      | Imposta su <code>Server</code> .                                            |
| <code>Span.Status</code>    | Imposta su <code>Unset</code> .                                             |



| OpenTelemetry                            | Schema dell'API di telemetria Lambda                                                                                    |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>Span.TraceId</code>                | Analizza l' AWS X-Ray intestazione trovata in <code>e.tracing.value</code> , quindi usa il <code>TraceId</code> valore. |
| <code>Span.ParentId</code>               | Analizza l' X-Ray intestazione trovata in <code>e.tracing.value</code> , quindi usa il valore. <code>Parent</code>      |
| <code>Span.SpanId</code>                 | Usa <code>e.tracing.spanId</code> se disponibile. Altrimenti, genera un nuovo <code>SpanId</code> .                     |
| <code>Span.SpanContext.TraceState</code> | N/A per un contesto di X-Ray tracciamento.                                                                              |
| <code>Span.SpanContext.TraceFlags</code> | Analizza l' X-Ray intestazione trovata in <code>e.tracing.value</code> , quindi usa il <code>Sampled</code> valore.     |
| <code>Span.Attributes</code>             | La tua estensione può aggiungere qualsiasi valore personalizzato qui.                                                   |

### Mappatura degli eventi \* RuntimeDone

| OpenTelemetry               | Schema dell'API di telemetria Lambda                                                                                                                   |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Span.Name</code>      | L'estensione genera il valore in base al campo <code>type</code> .                                                                                     |
| <code>Span.StartTime</code> | Usa <code>e.time</code> dall'evento <code>*Start</code> corrispondente.<br><br>In alternativa, utilizzare <code>e.time - e.metrics.durationMs</code> . |
| <code>Span.EndTime</code>   | N/A, perché l'evento non è ancora completato.                                                                                                          |
| <code>Span.Kind</code>      | Imposta su <code>Server</code> .                                                                                                                       |

| OpenTelemetry                            | Schema dell'API di telemetria Lambda                                                                                                                                |
|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Span.Status</code>                 | <p>Se <code>e.status</code> non è uguale a <code>success</code>, imposta su <code>Error</code>.</p> <p>In caso contrario, imposta il valore su <code>Ok</code>.</p> |
| <code>Span.Events[]</code>               | Utilizza <code>e.spans[]</code> .                                                                                                                                   |
| <code>Span.Events[i].Name</code>         | Utilizza <code>e.spans[i].name</code> .                                                                                                                             |
| <code>Span.Events[i].Time</code>         | Utilizza <code>e.spans[i].start</code> .                                                                                                                            |
| <code>Span.TraceId</code>                | Analizza l' AWS X-Ray intestazione trovata in <code>e.tracing.value</code> , quindi usa il <code>TraceId</code> valore.                                             |
| <code>Span.ParentId</code>               | Analizza l' X-Ray intestazione trovata in <code>e.tracing.value</code> , quindi usa il valore <code>Parent</code> .                                                 |
| <code>Span.SpanId</code>                 | Usa lo stesso <code>SpanId</code> dell'evento <code>*Start</code> . Se non disponibile, usa <code>e.tracing.spanId</code> o genera un nuovo <code>SpanId</code> .   |
| <code>Span.SpanContext.TraceState</code> | N/A per un contesto di X-Ray tracciamento.                                                                                                                          |
| <code>Span.SpanContext.TraceFlags</code> | Analizza l' X-Ray intestazione trovata in <code>e.tracing.value</code> , quindi usa il <code>Sampled</code> valore.                                                 |
| <code>Span.Attributes</code>             | La tua estensione può aggiungere qualsiasi valore personalizzato qui.                                                                                               |

## Mappatura degli eventi \*Report

| OpenTelemetry                            | Schema dell'API di telemetria Lambda                                                                                                                              |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Span.Name</code>                   | L'estensione genera il valore in base al campo <code>type</code> .                                                                                                |
| <code>Span.StartTime</code>              | <p>Usa <code>e.time</code> dall'evento <code>*Start</code> corrispondente.</p> <p>In alternativa, utilizzare <code>e.time - e.metrics.durationMs</code>.</p>      |
| <code>Span.EndTime</code>                | Utilizza <code>e.time</code> .                                                                                                                                    |
| <code>Span.Kind</code>                   | Imposta su <code>Server</code> .                                                                                                                                  |
| <code>Span.Status</code>                 | Usa lo stesso valore dell'evento <code>*RuntimeDone</code> .                                                                                                      |
| <code>Span.TraceId</code>                | Analizza l' AWS X-Ray intestazione trovata in <code>e.tracing.value</code> , quindi usa il valore. <code>TraceId</code>                                           |
| <code>Span.ParentId</code>               | Analizza l' X-Ray intestazione trovata in <code>e.tracing.value</code> , quindi usa il valore. <code>Parent</code>                                                |
| <code>Span.SpanId</code>                 | Usa lo stesso <code>SpanId</code> dell'evento <code>*Start</code> . Se non disponibile, usa <code>e.tracing.spanId</code> o genera un nuovo <code>SpanId</code> . |
| <code>Span.SpanContext.TraceState</code> | N/A per un contesto di X-Ray tracciamento.                                                                                                                        |
| <code>Span.SpanContext.TraceFlags</code> | Analizza l' X-Ray intestazione trovata in <code>e.tracing.value</code> , quindi usa il <code>Sampled</code> valore.                                               |
| <code>Span.Attributes</code>             | La tua estensione può aggiungere qualsiasi valore personalizzato qui.                                                                                             |

## Mappa a intervalli OTel con intervalli secondari

La tabella seguente descrive come convertire gli eventi dell'API di telemetria Lambda in intervalli OTel con intervalli secondari (annidati) per gli intervalli `*RuntimeDone`. Per le mappature `*Start` e `*Report`, fai riferimento alle tabelle in [the section called “Mappa a intervalli OTel con eventi di intervallo”](#), poiché sono le stesse per gli intervalli secondari. In questa tabella, `e` rappresenta l'evento proveniente dall'origine di telemetria.

### Mappatura degli eventi `* RuntimeDone`

| OpenTelemetry               | Schema dell'API di telemetria Lambda                                                                                                                                |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Span.Name</code>      | L'estensione genera il valore in base al campo <code>type</code> .                                                                                                  |
| <code>Span.StartTime</code> | <p>Usa <code>e.time</code> dall'evento <code>*Start</code> corrispondente.</p> <p>In alternativa, utilizzare <code>e.time - e.metrics.durationMs</code>.</p>        |
| <code>Span.EndTime</code>   | N/A, perché l'evento non è ancora completato.                                                                                                                       |
| <code>Span.Kind</code>      | Imposta su <code>Server</code> .                                                                                                                                    |
| <code>Span.Status</code>    | <p>Se <code>e.status</code> non è uguale a <code>success</code>, imposta su <code>Error</code>.</p> <p>In caso contrario, imposta il valore su <code>Ok</code>.</p> |
| <code>Span.TraceId</code>   | Analizza l' AWS X-Ray intestazione trovata in <code>e.tracing.value</code> , quindi usa il <code>TraceId</code> valore.                                             |
| <code>Span.ParentId</code>  | Analizza l' X-Ray intestazione trovata in <code>e.tracing.value</code> , quindi usa il valore <code>Parent</code> .                                                 |

| OpenTelemetry                                    | Schema dell'API di telemetria Lambda                                                                                                                              |
|--------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Span.SpanId</code>                         | Usa lo stesso <code>SpanId</code> dell'evento <code>*Start</code> . Se non disponibile, usa <code>e.tracing.spanId</code> o genera un nuovo <code>SpanId</code> . |
| <code>Span.SpanContext.TraceState</code>         | N/A per un contesto di X-Ray tracciamento.                                                                                                                        |
| <code>Span.SpanContext.TraceFlags</code>         | Analizza l' X-Ray intestazione trovata in <code>e.tracing.value</code> , quindi usa il <code>Sampled</code> valore.                                               |
| <code>Span.Attributes</code>                     | La tua estensione può aggiungere qualsiasi valore personalizzato qui.                                                                                             |
| <code>ChildSpan[i].Name</code>                   | Utilizza <code>e.spans[i].name</code> .                                                                                                                           |
| <code>ChildSpan[i].StartTime</code>              | Utilizza <code>e.spans[i].start</code> .                                                                                                                          |
| <code>ChildSpan[i].EndTime</code>                | Utilizza <code>e.spans[i].start + e.spans[i].durations</code> .                                                                                                   |
| <code>ChildSpan[i].Kind</code>                   | Uguale all'elemento padre <code>Span.Kind</code> .                                                                                                                |
| <code>ChildSpan[i].Status</code>                 | Uguale all'elemento padre <code>Span.Status</code> .                                                                                                              |
| <code>ChildSpan[i].TraceId</code>                | Uguale all'elemento padre <code>Span.TraceId</code> .                                                                                                             |
| <code>ChildSpan[i].ParentId</code>               | Usa elemento padre <code>Span.SpanId</code> .                                                                                                                     |
| <code>ChildSpan[i].SpanId</code>                 | Genera un nuovo <code>SpanId</code> .                                                                                                                             |
| <code>ChildSpan[i].SpanContext.TraceState</code> | N/A per un contesto di X-Ray tracciamento.                                                                                                                        |
| <code>ChildSpan[i].SpanContext.TraceFlags</code> | Uguale all'elemento padre <code>Span.SpanContext.TraceFlags</code> .                                                                                              |

## Utilizzo dell'API Logs di Lambda

### ⚠ Important

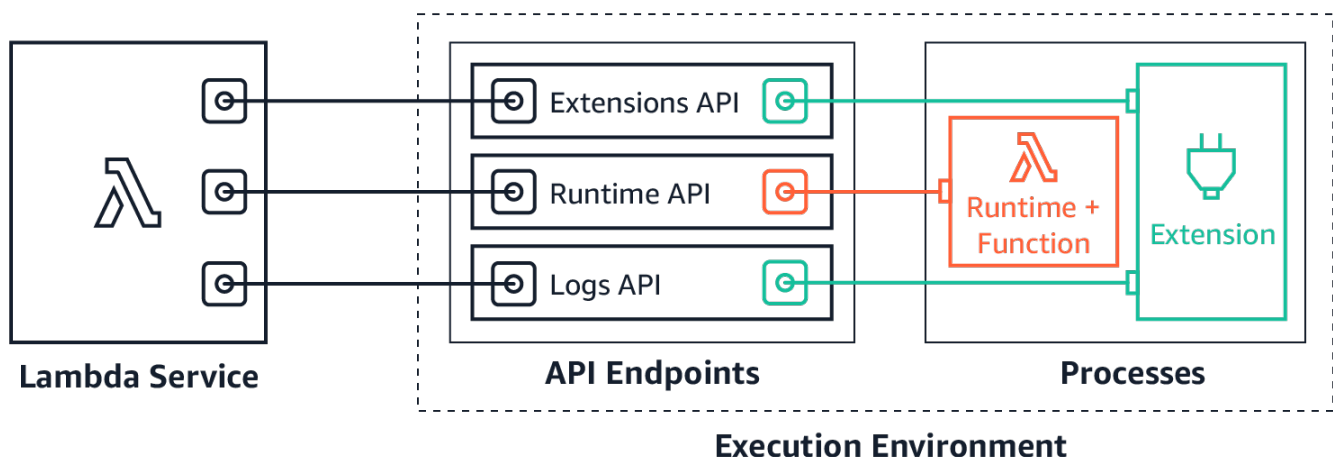
L'API di telemetria Lambda sostituisce l'API Lambda Logs. Sebbene l'API Logs rimanga completamente funzionante, in futuro consigliamo di utilizzare solo l'API di telemetria. Puoi iscrivere la tua estensione a un flusso di telemetria utilizzando l'API di telemetria o l'API Logs. Dopo la sottoscrizione tramite una di queste API, qualsiasi tentativo di sottoscrizione utilizzando l'altra API restituirà un errore.

### ⚠ Le istanze gestite Lambda non supportano l'API Logs

Le istanze gestite Lambda non supportano l'API Logs. Se utilizzi le funzioni Managed Instance, utilizza invece l'API di [telemetria](#). L'API di telemetria offre funzionalità avanzate per la raccolta e l'elaborazione dei dati di telemetria dalle funzioni Lambda.

Lambda acquisisce automaticamente i log di runtime e li trasmette ad Amazon. CloudWatch Questo flusso di registri contiene i log generati dal codice funzione e dalle estensioni e anche i log generati da Lambda nell'ambito del richiamo della funzione.

Le [estensioni Lambda](#) possono utilizzare l'API Logs del runtime di Lambda per eseguire la sottoscrizione ai flussi di log direttamente dall'interno dell'[ambiente di esecuzione](#) Lambda. Lambda trasmette i log all'estensione e l'estensione può quindi elaborare, filtrare e inviare i log a qualsiasi destinazione preferita.



L'API Logs consente alle estensioni di sottoscrivere tre flussi di log diversi:

- Log di funzioni che la funzione Lambda genera e scrive in `stdout` o `stderr`.
- Log di estensione generati dal codice di estensione.
- Log di piattaforma Lambda, che registrano eventi ed errori relativi a richiami ed estensioni.

#### Note

Lambda invia tutti i log a CloudWatch, anche quando un'estensione si iscrive a uno o più flussi di log.

## Argomenti

- [Sottoscrizione ai log di ricezione](#)
- [Utilizzo della memoria](#)
- [Protocolli destinazione](#)
- [Configurazione buffering](#)
- [Esempio di sottoscrizione](#)
- [Codice di esempio per l'API Logs](#)
- [Riferimento dell'API Logs.](#)
- [Messaggi di log](#)

## Sottoscrizione ai log di ricezione

Un'estensione Lambda può sottoscrivere i log di ricezione inviando una richiesta di sottoscrizione all'API Logs.

Per sottoscrivere i log di ricezione, è necessario l'identificatore di estensione (`Lambda-Extension-Id`). Innanzitutto [registrare l'estensione](#) per ricevere l'identificatore dell'estensione. Quindi sottoscrivere l'API Logs durante l'[inizializzazione](#). Al termine della fase di inizializzazione, Lambda non elabora le richieste di sottoscrizione.

 Note

La sottoscrizione Logs API è idempotente. Le richieste di sottoscrizione duplicate non comportano sottoscrizioni duplicate.

## Utilizzo della memoria

L'utilizzo della memoria aumenta linearmente man mano che aumenta il numero di abbonati. Le sottoscrizioni consumano risorse di memoria perché ogni sottoscrizione apre un nuovo buffer di memoria per archiviare i log. Per ottimizzare l'utilizzo della memoria, è possibile regolare la [configurazione di buffering](#). L'utilizzo della memoria buffer conta per il consumo complessivo della memoria nell'ambiente di esecuzione.

## Protocolli destinazione

È possibile scegliere uno dei seguenti protocolli per ricevere i log:

1. HTTP (consigliato) – Lambda consegna i log a un endpoint HTTP locale (`http://sandbox.localdomain:${PORT}/${PATH}`) come matrice di record in formato JSON. Il parametro `$PATH` è facoltativo. Si noti che è supportato solo HTTP, non HTTPS. Puoi scegliere di ricevere i log tramite PUT o POST.
2. TCP – Lambda consegna i log a una porta TCP in [formato JSON delimitato da Newline \(NDJSON\)](#).

Si consiglia di utilizzare HTTP anziché TCP. Con TCP, la piattaforma Lambda non può riconoscere che i log vengono consegnati al livello dell'applicazione. Pertanto, se l'estensione si blocca si potrebbero perdere i log. HTTP non condivide questa limitazione.

Si consiglia inoltre di impostare il listener HTTP locale o la porta TCP prima di sottoscrivere i log di ricezione. Durante l'installazione, tenere presente quanto segue:

- Lambda invia i log solo alle destinazioni che si trovano all'interno dell'ambiente di esecuzione.
- Lambda ritenta il tentativo di inviare i log (con backoff) se non c'è listener o se la richiesta POST o PUT genera errori. Se il sottoscrittore del log si blocca, continuerà a ricevere i log dopo che Lambda riavvia l'ambiente di esecuzione.
- Lambda riserva la porta 9001. Non ci sono altre restrizioni o raccomandazioni sul numero di porta.



## Configurazione buffering

Lambda può tamponare i log e consegnarli al sottoscrittore. È possibile configurare questo comportamento nella richiesta di sottoscrizione specificando i seguenti campi facoltativi. Si noti che Lambda utilizza il valore predefinito per qualsiasi campo non specificato.

- `timeoutMs` – Il tempo massimo (in millisecondi) per il buffer di un batch. Valore predefinito: 1.000  
Minimo: 25. Massimo: 30.000
- `maxBytes` – La dimensione massima (in byte) dei registri al buffer in memoria. Valore predefinito: 262.144. Minimo: 262.144. Massimo: 1.048.576.
- `maxItems` – Il numero massimo di eventi da memorizzare nel buffer. Valore predefinito: 10.000  
Minimo: 1.000. Massimo: 10.000.

Durante la configurazione del buffering, prendere nota dei seguenti punti:

- Lambda svuota i log se uno qualsiasi dei flussi di input è chiuso, ad esempio, se il runtime si arresta in modo anomalo.
- Ogni sottoscrittore può specificare una configurazione di buffering diversa durante la richiesta di sottoscrizione.
- Considerare la dimensione del buffer necessaria per leggere i dati. Aspettarsi di ricevere payload grandi come  $2 * \text{maxBytes} + \text{metadata}$ , dove `maxBytes` è configurato nella richiesta di sottoscrizione. Ad esempio, Lambda aggiunge i byte di metadati seguenti a ciascun record:

```
{
  "time": "2020-08-20T12:31:32.123Z",
  "type": "function",
  "record": "Hello World"
}
```

- Se il server di sottoscrizione non è in grado di elaborare i log in ingresso abbastanza rapidamente, Lambda potrebbe eliminare i log per mantenere limitato l'utilizzo della memoria. Per indicare il numero di record eliminati, Lambda invia un log `platform.logsDropped`. Per ulteriori informazioni, consulta [the section called “Lambda: non vengono visualizzati tutti i log della mia funzione”](#).

## Esempio di sottoscrizione

L'esempio seguente mostra una richiesta di sottoscrizione ai log della piattaforma e delle funzioni.

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs HTTP/1.1
{ "schemaVersion": "2020-08-15",
  "types": [
    "platform",
    "function"
  ],
  "buffering": {
    "maxItems": 1000,
    "maxBytes": 262144,
    "timeoutMs": 100
  },
  "destination": {
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080/lambda_logs"
  }
}
```

Se la richiesta ha esito positivo, il sottoscrittore riceve una risposta riuscita HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

## Codice di esempio per l'API Logs

Per un codice di esempio che mostra come inviare i log a una destinazione personalizzata, consulta [Using AWS Lambda extensions to send log to custom destination](#) sul AWS Compute Blog.

Per esempi di codice Python e Go che mostrano come sviluppare un'estensione Lambda di base e sottoscrivere l'API Logs, consulta [AWS Lambda Extensions nel AWS repository](#) Samples. GitHub Per ulteriori informazioni sulla creazione di un'estensione Lambda, consultare [the section called “API estensioni”](#).

## Riferimento dell'API Logs.

È possibile recuperare l'endpoint API Logs dalla `AWS_LAMBDA_RUNTIME_API` variabile di ambiente. Per inviare una richiesta API, utilizzare il prefisso `2020-08-15/` prima del percorso API. Ad esempio:

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs
```

La specifica OpenAPI per la richiesta di sottoscrizione a Log API, versione 2020-08-15, è disponibile qui: [logs-api-request.zip](#)

## Subscribe

Per sottoscrivere uno o più flussi di log disponibili nell'ambiente di esecuzione di Lambda, le estensioni inviano una richiesta API di sottoscrizione.

Percorso – /logs

Method – PUT

Parametri corpo

`destination` - Consultare [the section called “Protocolli destinazione”](#). Campo obbligatorio: sì Tipo: stringhe.

`buffering` - Consultare [the section called “Configurazione buffering”](#). Campo obbligatorio: no. Tipo: stringhe.

`types` – Un array dei tipi di log da ricevere. Campo obbligatorio: sì Tipo: array di stringhe Valori validi: «piattaforma», «funzione», «estensione».

`schemaVersion` – Obbligatorio: no. Valore di default: "2020-08-15". Impostare su "2021-03-18" per l'estensione in modo da ricevere messaggi [platform.runtimeDone](#).

Parametri di risposta

Le specifiche OpenAPI per le risposte di sottoscrizione, versione 2020-08-15, sono disponibili per i protocolli HTTP e TCP:

- HTTP: [logs-api-http-response.zip](#)
- TCP: [logs-api-tcp-response.zip](#)

Codice di risposta

- 200 – Richiesta completata con successo
- 202 – Richiesta accettata. Risposta ad una richiesta di sottoscrizione durante il test locale.
- 4XX – Richiesta non valida

- 500 – Errore servizio

Se la richiesta ha esito positivo, il sottoscrittore riceve una risposta riuscita HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Se la richiesta fallisce, il sottoscrittore riceve una risposta di errore. Ad esempio:

```
HTTP/1.1 400 OK
{
  "errorType": "Logs.ValidationError",
  "errorMessage": "URI port is not provided; types should not be empty"
}
```

## Messaggi di log

L'API Logs consente alle estensioni di sottoscrivere tre flussi di log diversi:

- Funzione – Log che la funzione Lambda genera e scrive in `stdout` o `stderr`.
- Estensione – Log generati dal codice di estensione.
- Piattaforma – Log generati dalla piattaforma runtime, che registrano eventi ed errori relativi a richiami ed estensioni.

## Argomenti

- [Log delle funzioni](#)
- [Log di estensioni](#)
- [Log di piattaforma](#)

## Log delle funzioni

La funzione Lambda e le estensioni interne generano log di funzioni e li scrivono in `stdout` o `stderr`.

L'esempio seguente mostra il formato di un messaggio di registro delle funzioni. {«time»: «2020-08-20 », «type»: T12:31:32.123Z «function», «record»: «ERRORE riscontrato. Stack trace:\n\my-function (line 10)\n" }

## Log di estensioni

Le estensioni possono generare log di estensioni. Il formato del log è uguale a quello di un log di funzioni.

## Log di piattaforma

Lambda genera messaggi di log per eventi della piattaforma come `platform.start`, `platform.end` e `platform.fault`.

Facoltativamente, è possibile sottoscrivere la versione 2021-03-18 dello schema dell'API Log, che include il messaggio di log `platform.runtimeDone`.

## Esempio di messaggi di log di piattaforma

Nell'esempio seguente vengono illustrati i log di inizio e di fine della piattaforma. Questi log indicano l'ora di inizio e l'ora di fine chiamata per la chiamata specificata da `requestId`.

```
{
  "time": "2020-08-20T12:31:32.123Z",
  "type": "platform.start",
  "record": {"requestId": "6f7f0961f83442118a7af6fe80b88d56"}
}
{
  "time": "2020-08-20T12:31:32.123Z",
  "type": "platform.end",
  "record": {"requestId": "6f7f0961f83442118a7af6fe80b88d56"}
}
```

Il messaggio di `RuntimeDone` log `platform.init` mostra lo stato della `Runtime init` sottofase, che fa parte della fase del ciclo di vita di `Init`. [???](#) In caso di riuscita di `Runtime init`, il runtime invia una richiesta API `/next` di runtime (per i tipi di inizializzazione `on-demand` e `provisioned-concurrency`) o `restore/next` (per il tipo di inizializzazione `snap-start`). L'esempio seguente mostra un messaggio di log `platform.init` riuscito per il tipo di inizializzazione. `RuntimeDone snap-start`

```
{
  "time": "2022-07-17T18:41:57.083Z",
  "type": "platform.initRuntimeDone",
  "record": {
    "initializationType": "snap-start",
    "status": "success"
  }
}
```

```
}  
}
```

Il messaggio di log `platform.initReport` mostra quanto è durata la fase `Init` e quanti millisecondi sono stati fatturati durante questa fase. Quando il tipo di inizializzazione è `provisioned-concurrency`, Lambda invia questo messaggio durante la chiamata. Quando il tipo di inizializzazione è `snap-start`, Lambda invia questo messaggio dopo aver ripristinato lo snapshot. L'esempio seguente mostra un messaggio di log `platform.initReport` riuscito per il tipo di inizializzazione `snap-start`.

```
{  
  "time": "2022-07-17T18:41:57.083Z",  
  "type": "platform.initReport",  
  "record": {  
    "initializationType": "snap-start",  
    "metrics": {  
      "durationMs": 731.79,  
      "billedDurationMs": 732  
    }  
  }  
}
```

Il log dei report della piattaforma include parametri sulla chiamata specificata da `requestId`. Il campo `initDurationMs` è incluso nel log solo se la chiamata include un avvio a freddo. Se la AWS X-Ray traccia è attiva, il registro include i metadati. X-Ray Nell'esempio seguente viene illustrato un log di rapporto della piattaforma per una chiamata che includeva un avvio a freddo.

```
{  
  "time": "2020-08-20T12:31:32.123Z",  
  "type": "platform.report",  
  "record": { "requestId": "6f7f0961f83442118a7af6fe80b88d56",  
    "metrics": { "durationMs": 101.51,  
      "billedDurationMs": 300,  
      "memorySizeMB": 512,  
      "maxMemoryUsedMB": 33,  
      "initDurationMs": 116.67  
    }  
  }  
}
```

Il log della piattaforma acquisisce errori di runtime o dell'ambiente di esecuzione. L'esempio seguente mostra un messaggio di log di errore della piattaforma.

```
{
  "time": "2020-08-20T12:31:32.123Z",
  "type": "platform.fault",
  "record": "RequestId: d783b35e-a91d-4251-af17-035953428a2c Process exited before
completing request"
}
```

### Note

AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. Uno degli output dei log interessati da questa modifica è il campo "record" del log degli errori della piattaforma. Gli esempi seguenti mostrano campi "record" nei formati vecchi e nuovi. Il nuovo stile del log degli errori contiene un messaggio più conciso. Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regioni, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia nel nuovo formato.

### Example record del log degli errori della piattaforma (vecchio stile)

```
"record": "RequestId: ... \tError: Runtime exited with error: exit status
255\nRuntime.ExitError"
```

### Example record del log degli errori della piattaforma (nuovo stile)

```
"record": "RequestId: ... Status: error \tErrorType: Runtime.ExitError"
```

Lambda genera un log di estensioni della piattaforma quando un'estensione si registra con l'API delle estensioni. L'esempio seguente mostra un messaggio dell'estensione della piattaforma.

```
{
  "time": "2020-08-20T12:31:32.123Z",
  "type": "platform.extension",
  "record": { "name": "Foo.bar",
    "state": "Ready",
    "events": ["INVOKE", "SHUTDOWN"]
  }
}
```

```
}  
}
```

Lambda genera un log di sottoscrizione al log della piattaforma quando un'estensione sottoscrive l'API Log. Nell'esempio seguente viene illustrato un messaggio di sottoscrizione log.

```
{  
  "time": "2020-08-20T12:31:32.123Z",  
  "type": "platform.logsSubscription",  
  "record": {"name": "Foo.bar",  
    "state": "Subscribed",  
    "types": ["function", "platform"]},  
}
```

Lambda genera un log eliminato dei log di piattaforma quando un'estensione non è in grado di elaborare il numero di log che sta ricevendo. Nell'esempio seguente viene mostrato unplatform.logsDropped messaggio di log.

```
{  
  "time": "2020-08-20T12:31:32.123Z",  
  "type": "platform.logsDropped",  
  "record": {"reason": "Consumer seems to have fallen behind as it has not  
acknowledged receipt of logs.",  
    "droppedRecords": 123,  
    "droppedBytes": 12345  
  },  
}
```

Il messaggio di log platform.restoreStart mostra l'ora di inizio della fase Restore (solo per il tipo di inizializzazione snap-start). Esempio:

```
{  
  "time": "2022-07-17T18:43:44.782Z",  
  "type": "platform.restoreStart",  
  "record": {}  
}
```

Il messaggio di log platform.restoreReport mostra quanto è durata la fase Restore e quanti millisecondi sono stati fatturati durante questa fase (solo per il tipo di inizializzazione snap-start). Esempio:



```
{
  "time": "2022-07-17T18:43:45.936Z",
  "type": "platform.restoreReport",
  "record": {
    "metrics": {
      "durationMs": 70.87,
      "billedDurationMs": 13
    }
  }
}
```

## Messaggi RuntimeDone della piattaforma

Se si imposta la versione dello schema su "2021-03-18" nella richiesta di sottoscrizione, Lambda invia un messaggio `platform.runtimeDone` dopo il completamento della chiamata della funzione o con un errore. L'estensione può utilizzare questo messaggio per arrestare tutta la raccolta di dati di telemetria per questa chiamata di funzione.

La specifica OpenAPI per il tipo di evento Log nella versione dello schema 2021-03-18 è disponibile qui: [schema-2021-03-18.zip](#)

Lambda genera il messaggio di log `platform.runtimeDone` quando il runtime invia una richiesta API di runtime Next o Error. Il log `platform.runtimeDone` informa i consumatori dell'API Log che la chiamata di funzione è stata completata. Le estensioni possono utilizzare queste informazioni per decidere quando inviare tutti i dati di telemetria raccolti durante tale chiamata.

## Esempi

Lambda invia il messaggio `platform.runtimeDone` dopo che il runtime invia la richiesta NEXT al completamento della chiamata della funzione. Negli esempi seguenti vengono illustrati i messaggi relativi a ciascuno dei valori di stato: riuscita, errore e timeout.

### Example Esempio di messaggio di riuscita

```
{
  "time": "2021-02-04T20:00:05.123Z",
  "type": "platform.runtimeDone",
  "record": {
    "requestId": "6f7f0961f83442118a7af6fe80b88",
    "status": "success"
  }
}
```

```
}  
}
```

### Example Esempio di messaggio di errore

```
{  
  "time": "2021-02-04T20:00:05.123Z",  
  "type": "platform.runtimeDone",  
  "record": {  
    "requestId": "6f7f0961f83442118a7af6fe80b88",  
    "status": "failure"  
  }  
}
```

### Example Esempio di messaggio di timeout

```
{  
  "time": "2021-02-04T20:00:05.123Z",  
  "type": "platform.runtimeDone",  
  "record": {  
    "requestId": "6f7f0961f83442118a7af6fe80b88",  
    "status": "timeout"  
  }  
}
```

### Example Esempio di RuntimeDone messaggio platform.restore (solo tipo di inizializzazione snap-start)

Il messaggio di RuntimeDone registro platform.restore indica se la fase ha avuto successo o meno. Restore Lambda genera questo messaggio quando il runtime invia una richiesta API di runtime restore/next. Ci sono tre stati possibili: riuscito, errore e timeout. L'esempio seguente mostra un messaggio di log RuntimeDone platform.restore riuscito.

```
{  
  "time": "2022-07-17T18:43:45.936Z",  
  "type": "platform.restoreRuntimeDone",  
  "record": {  
    "status": "success"  
  }  
}
```

# Risoluzione dei problemi in Lambda

Negli argomenti seguenti vengono forniti suggerimenti per la risoluzione dei problemi relativi a errori e problemi che potrebbero verificarsi durante l'utilizzo della console, degli strumenti o dell'API Lambda. Se scopri un problema che non è elencato qui di seguito, puoi utilizzare il pulsante Feedback in questa pagina per segnalarlo.

Per ulteriori suggerimenti sulla risoluzione dei problemi e per risposte a domande comuni relative al supporto, visitare il [Knowledge Center di AWS](#).

Per ulteriori informazioni sul debug e la risoluzione dei problemi delle applicazioni Lambda, consulta [Debug](#) in Serverless Land.

## Argomenti

- [Risolvi i problemi di configurazione in Lambda](#)
- [Risoluzione dei problemi relativi alle implementazioni in Lambda](#)
- [Risoluzione dei problemi di invocazione in Lambda](#)
- [Risoluzione dei problemi di esecuzione in Lambda](#)
- [Risolvi i problemi di mappatura delle sorgenti degli eventi in Lambda](#)
- [Risoluzione dei problemi di rete in Lambda](#)

## Risolvi i problemi di configurazione in Lambda

Le impostazioni di configurazione delle funzioni possono avere un impatto sulle prestazioni e sul comportamento complessivi della funzione Lambda. Queste potrebbero non causare errori di funzione effettivi, ma possono causare timeout e risultati imprevisti.

I seguenti argomenti forniscono consigli per la risoluzione dei problemi più comuni che potresti riscontrare relativi alle impostazioni di configurazione delle funzioni Lambda.

## Argomenti

- [Configurazioni della memoria](#)
- [CPU-bound configurazioni](#)
- [Timeout](#)
- [Perdita di memoria tra invocazioni](#)

- [Risultati asincroni restituiti a una invocazione successiva](#)

## Configurazioni della memoria

È possibile configurare una funzione Lambda per utilizzare tra 128 MB e 10.240 MB di memoria. Per impostazione predefinita, a qualsiasi funzione creata nella console viene assegnata la quantità di memoria minima. Molte funzioni Lambda sono performanti con questa impostazione minima. Tuttavia, se stai importando librerie di codice di grandi dimensioni o completando attività che richiedono molta memoria, 128 MB non sono sufficienti.

Se le funzioni vengono eseguite molto più lentamente del previsto, il primo passo è aumentare l'impostazione della memoria. Per le funzioni legate alla memoria, questo risolve il problema e potrebbe migliorare le prestazioni della funzione.

## CPU-bound configurazioni

Per le operazioni ad alta intensità di calcolo, se le prestazioni della funzione sono inferiori al previsto, ciò potrebbe essere dovuto al fatto che la funzione è in uso. CPU-bound In questo caso, la capacità di calcolo della funzione non può tenere il passo con il lavoro.

Sebbene Lambda non consenta di modificare direttamente la configurazione della CPU, la CPU viene controllata indirettamente tramite le impostazioni della memoria. Il servizio Lambda alloca proporzionalmente più CPU virtuale man mano che si alloca più memoria. Con 1,8 GB di memoria, una funzione Lambda ha un'intera vCPU allocata e, al di sopra di questo livello, ha accesso a più di un core vCPU. Con 10.240 MB, sono disponibili 6 vCPU. In altre parole, è possibile migliorare le prestazioni aumentando l'allocazione di memoria, anche se la funzione non utilizza tutta la memoria.

## Timeout

[I timeout](#) per le funzioni Lambda possono essere impostati tra 1 e 900 secondi (15 minuti). Per impostazione predefinita, la console Lambda lo imposta su 3 secondi. Il valore di timeout è una valvola di sicurezza che garantisce che le funzioni non vengano eseguite all'infinito. Una volta raggiunto il valore di timeout, Lambda interrompe l'invocazione della funzione.

Se un valore di timeout è impostato vicino alla durata media di una funzione, aumenta il rischio che la funzione scada inaspettatamente. La durata di una funzione può variare in base alla quantità di trasferimento ed elaborazione dei dati e alla latenza dei servizi con cui interagisce la funzione. Le cause più comuni di timeout includono:

- Quando si scaricano dati da bucket S3 o altri archivi di dati, il download è più grande o richiede più tempo della media.
- Una funzione invia una richiesta a un altro servizio, che impiega più tempo a rispondere.
- I parametri forniti a una funzione richiedono una maggiore complessità computazionale della funzione, il che fa sì che l'invocazione richieda più tempo.

Quando testate l'applicazione, assicuratevi che i test riflettano accuratamente le dimensioni e la quantità di dati e i valori dei parametri realistici. È importante sottolineare che utilizzate i set di dati al limite superiore di ciò che è ragionevolmente previsto per il carico di lavoro.

Inoltre, implementa i limiti massimi del carico di lavoro laddove possibile. In questo esempio, l'applicazione potrebbe utilizzare un limite di dimensione massima per ogni tipo di file. È quindi possibile testare le prestazioni dell'applicazione per una gamma di dimensioni di file previste, fino ai limiti massimi inclusi.

## Perdita di memoria tra invocazioni

Le variabili e gli oggetti globali memorizzati nella fase INIT di una invocazione Lambda mantengono il loro stato tra le invocazioni a caldo. Vengono ripristinati completamente solo quando l'ambiente di esecuzione viene eseguito per la prima volta (operazione nota anche come «avvio a freddo»). Tutte le variabili memorizzate nell'handler vengono distrutte quando l'handler viene terminato. È consigliabile utilizzare la fase INIT per configurare connessioni al database, caricare librerie, creare cache e caricare risorse immutabili.

Quando utilizzi librerie di terze parti per più chiamate nello stesso ambiente di esecuzione, controlla la loro documentazione per verificarne l'utilizzo in un ambiente di elaborazione senza server. Alcune librerie di connessione e registrazione al database potrebbero salvare i risultati delle chiamate intermedie e altri dati. Ciò fa sì che l'utilizzo della memoria di queste librerie aumenti con le successive invocazioni a caldo. In tal caso, potresti scoprire che la funzione Lambda esaurisce la memoria, anche se il tuo codice personalizzato sta eliminando le variabili correttamente.

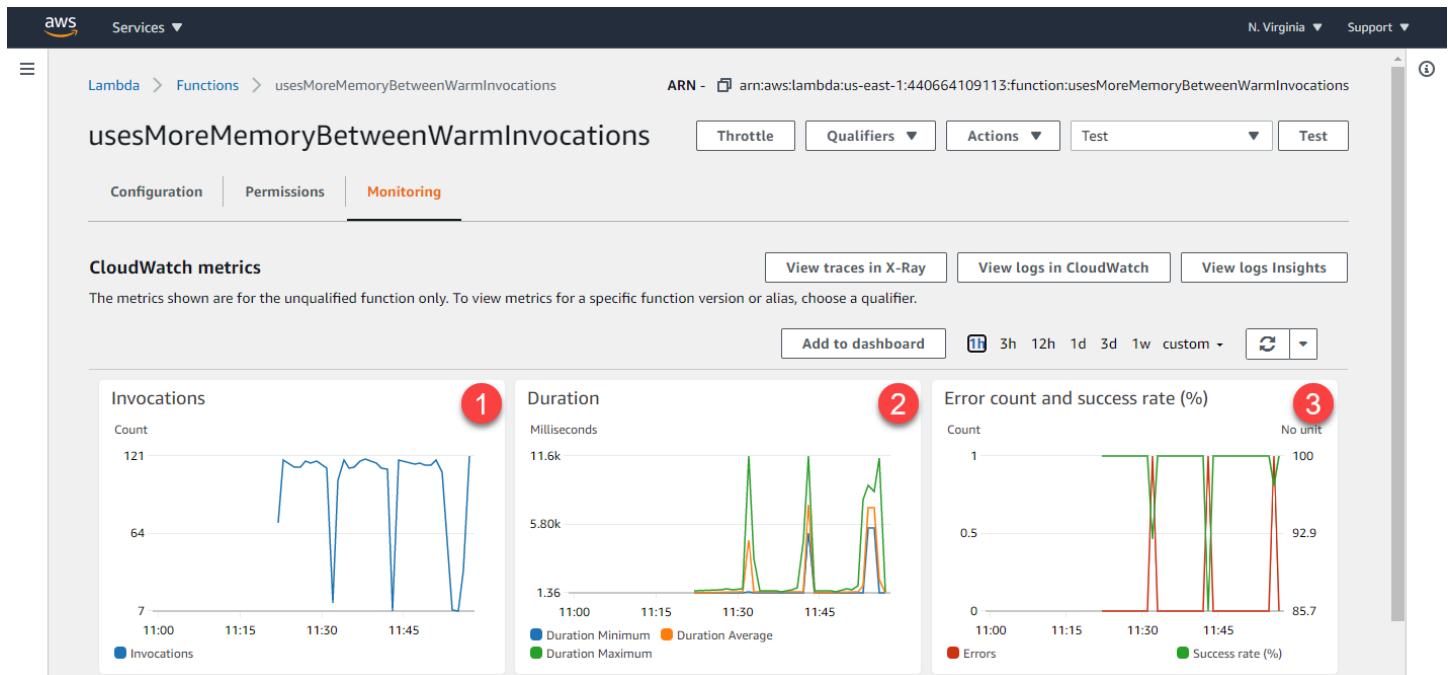
Questo problema riguarda le invocazioni che si verificano in ambienti di esecuzione a caldo. Ad esempio, il seguente codice crea una perdita di memoria tra le invocazioni. La funzione Lambda consuma memoria aggiuntiva ad ogni invocazione aumentando le dimensioni di un array globale:

```
let a = []

exports.handler = async (event) => {
```

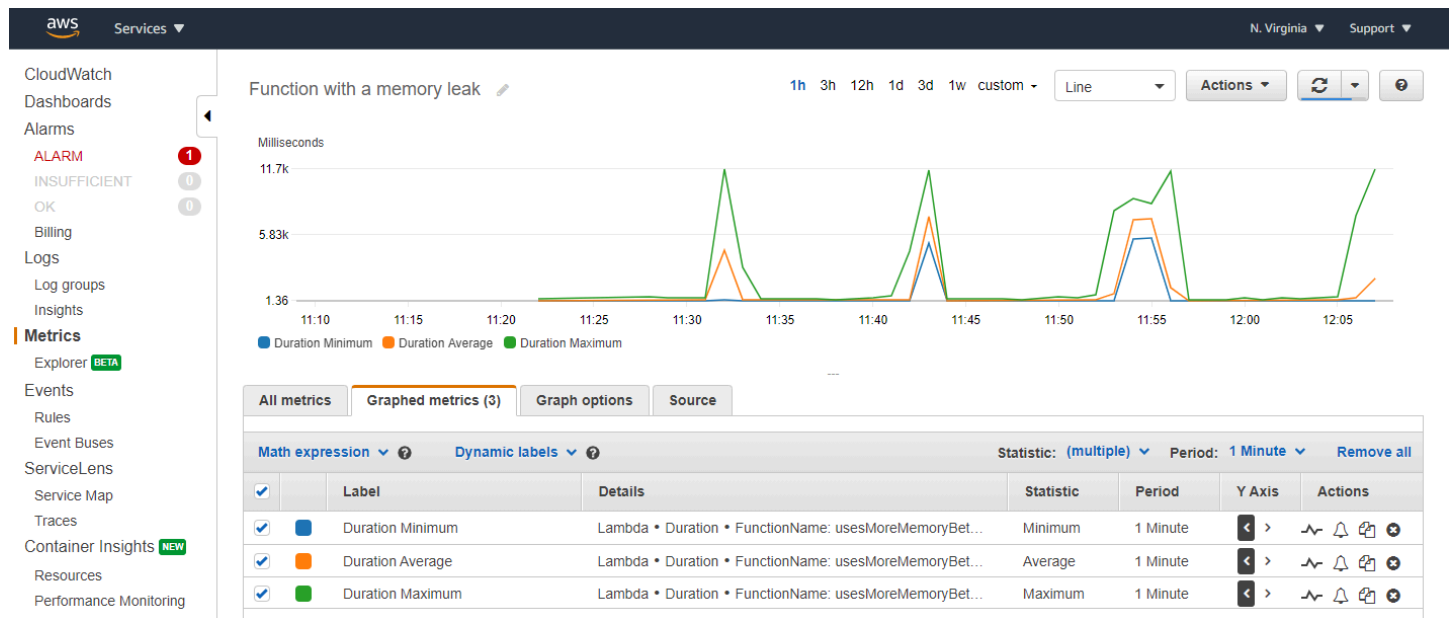
```
a.push(Array(100000).fill(1))
}
```

Configurata con 128 MB di memoria, dopo aver richiamato questa funzione 1.000 volte, la scheda Monitoraggio della funzione Lambda mostra le tipiche modifiche nelle invocazioni, nella durata e nel conteggio degli errori quando si verifica una perdita di memoria:



1. Richiami: una velocità di transazione costante viene interrotta periodicamente poiché le chiamate richiedono più tempo per essere completate. Durante lo stato stazionario, la perdita di memoria non sta consumando tutta la memoria allocata alla funzione. Man mano che le prestazioni peggiorano, il sistema operativo utilizza la memoria locale per adattarla alla crescente quantità di memoria richiesta dalla funzione, il che comporta un minor numero di transazioni completate.
2. Durata: prima che la funzione esaurisca la memoria, termina le chiamate a una velocità costante di millisecondi a due cifre. Man mano che si verifica la paginazione, la durata aumenta di un ordine di grandezza.
3. Numero di errori: quando la perdita di memoria supera la memoria allocata, alla fine la funzione si blocca a causa del superamento del timeout del calcolo o dell'ambiente di esecuzione che interrompe la funzione.

Dopo l'errore, Lambda riavvia l'ambiente di esecuzione, il che spiega perché tutti e tre i grafici mostrano un ritorno allo stato originale. L'espansione delle CloudWatch metriche relative alla durata fornisce maggiori dettagli per le statistiche sulla durata minima, massima e media:



Per trovare gli errori generati nelle 1000 chiamate, puoi utilizzare il linguaggio di interrogazione Insights. CloudWatch La seguente query esclude i log informativi per riportare solo gli errori:

```
fields @timestamp, @message
| sort @timestamp desc
| filter @message not like 'EXTENSION'
| filter @message not like 'Lambda Insights'
| filter @message not like 'INFO'
| filter @message not like 'REPORT'
| filter @message not like 'END'
| filter @message not like 'START'
```

Se eseguita sul gruppo di log per questa funzione, ciò dimostra che i timeout erano responsabili degli errori periodici:

The screenshot shows the AWS CloudWatch Logs Insights interface. The left sidebar contains navigation options like CloudWatch, Dashboards, Alarms, and Insights. The main panel displays a query for the log group `/aws/lambda/usesMoreMemoryBetweenWarmInvocations`. The query is as follows:

```
1 fields @timestamp, @message
2 | sort @timestamp desc
3 | filter @message not like 'EXTENSION'
4 | filter @message not like 'Lambda Insights'
5 | filter @message not like 'INFO'
6 | filter @message not like 'REPORT'
7 | filter @message not like 'END'
8 | filter @message not like 'START'
9 | limit 20
```

The results are visualized as a histogram and a table. The histogram shows four records. The table below shows the details of these records:

| # | @timestamp               | @message                                                                |
|---|--------------------------|-------------------------------------------------------------------------|
| 1 | 2020-10-14T08:07:46.361Z | 1917d63d-ccf5-4547-987a-1fecb469447f Task timed out after 11.65 seconds |
| 2 | 2020-10-14T07:56:39.579Z | 1a00b31d-86cb-42e6-b916-eb57c3d3b69a Task timed out after 11.45 seconds |
| 3 | 2020-10-14T07:44:00.652Z | 43644f33-8dec-4cea-87c5-a92a8c2da8cf Task timed out after 11.56 seconds |
| 4 | 2020-10-14T07:33:05.929Z | abab510c-92a3-4b69-8be7-a62b23876418 Task timed out after 11.61 seconds |

## Risultati asincroni restituiti a una invocazione successiva

Per il codice di funzione che utilizza pattern asincroni, è possibile che i risultati di callback di una chiamata vengano restituiti in una chiamata futura. Questo esempio utilizza Node.js, ma la stessa logica può essere applicata ad altri runtime che utilizzano pattern asincroni. La funzione utilizza la tradizionale sintassi di callback in JavaScript. Richiama una funzione asincrona con un contatore incrementale che tiene traccia del numero di invocazioni:

```
let seqId = 0

exports.handler = async (event, context) => {
  console.log(`Starting: sequence Id=${++seqId}`)
  doWork(seqId, function(id) {
    console.log(`Work done: sequence Id=${id}`)
  })
}

function doWork(id, callback) {
  setTimeout(() => callback(id), 3000)
```



```
}
```

Quando vengono richiamati più volte in successione, i risultati dei callback si verificano nelle invocazioni successive:

The screenshot displays the AWS Lambda console interface for a function named 'delayedAsyncReturn'. The 'Function code' tab is active, showing a JavaScript file named 'index.js'. The code defines an asynchronous handler that logs the start of a sequence, calls a 'doWork' function with a callback, and then logs the completion of the sequence. The 'doWork' function uses 'setTimeout' to delay the callback. The 'Execution Result' tab shows the function's execution details, including the request ID, function logs, and execution metrics. The logs indicate that the function was executed successfully, with the sequence ID increasing from 1 to 3 across three separate invocations. The execution time was 1.54 ms, and the memory used was 64 MB.

```
1 let seqId = 0
2
3 exports.handler = async (event) => {
4   console.log(`Starting: sequence Id=${++seqId}`)
5   doWork(seqId, function(id) {
6     console.log(`Work done: sequence Id=${id}`)
7   })
8 }
9
10 function doWork(id, callback) {
11   setTimeout(() => callback(id), 3000)
12 }
```

Execution Result: Status: Succeeded, Max Memory Used: 64 MB, Time: 1.54 ms

Response: null

Request ID: 6afdf887-424a-4ec6-b622-3ffc07eebb64

Function logs:

| Timestamp                | RequestID                            | Message                  | Level |
|--------------------------|--------------------------------------|--------------------------|-------|
| 2020-10-13T19:22:38.586Z | 6afdf887-424a-4ec6-b622-3ffc07eebb64 | Starting: sequence Id=1  | INFO  |
| 2020-10-13T19:22:38.587Z | 6afdf887-424a-4ec6-b622-3ffc07eebb64 | Work done: sequence Id=1 | INFO  |
| 2020-10-13T19:22:38.587Z | 6afdf887-424a-4ec6-b622-3ffc07eebb64 | Starting: sequence Id=2  | INFO  |
| 2020-10-13T19:22:38.587Z | 6afdf887-424a-4ec6-b622-3ffc07eebb64 | Work done: sequence Id=2 | INFO  |
| 2020-10-13T19:22:38.587Z | 6afdf887-424a-4ec6-b622-3ffc07eebb64 | Starting: sequence Id=3  | INFO  |
| 2020-10-13T19:22:38.587Z | 6afdf887-424a-4ec6-b622-3ffc07eebb64 | Work done: sequence Id=3 | INFO  |

END RequestId: 6afdf887-424a-4ec6-b622-3ffc07eebb64

REPORT RequestId: 6afdf887-424a-4ec6-b622-3ffc07eebb64 Duration: 1.54 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 64 MB

1. Il codice richiama la funzione `doWork`, fornendo una funzione di callback come ultimo parametro.
2. Il completamento della funzione `doWork` richiede un certo periodo di tempo prima di richiamare il callback.
3. La registrazione della funzione indica che l'invocazione termina prima che la funzione termini l'esecuzione. `doWork` Inoltre, dopo l'avvio di un'iterazione, vengono elaborati i callback delle iterazioni precedenti, come mostrato nei log.

In JavaScript, i callback asincroni vengono gestiti con un ciclo di eventi. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop> Altri runtime utilizzano meccanismi diversi per gestire la simultaneità. Quando l'ambiente di esecuzione della funzione termina, Lambda blocca l'ambiente fino alla chiamata successiva.

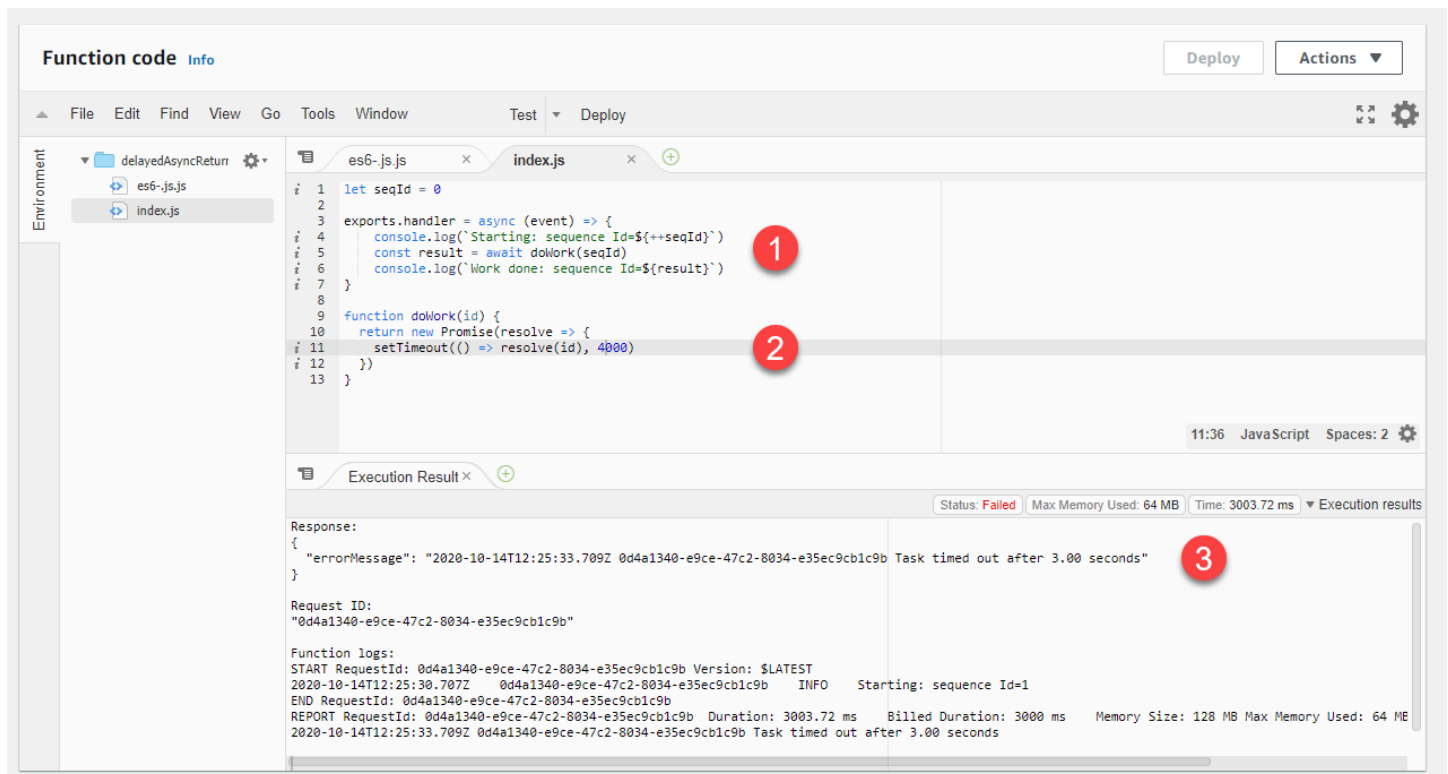
Dopo la ripresa, JavaScript continua l'elaborazione del ciclo di eventi, che in questo caso include un callback asincrono da una chiamata precedente. Senza questo contesto, può sembrare che la funzione esegua codice senza motivo e restituisca dati arbitrari. In effetti, è davvero un artefatto del modo in cui la simultaneità di runtime e gli ambienti di esecuzione interagiscono.

Ciò crea la possibilità che i dati privati di una invocazione precedente vengano visualizzati in una invocazione successiva. Esistono due modi per prevenire o rilevare questo comportamento. Innanzitutto, JavaScript fornisce le parole chiave [async e await](#) per semplificare lo sviluppo asincrono e inoltre forzare l'esecuzione del codice in attesa del completamento di una chiamata asincrona. La funzione precedente può essere riscritta utilizzando questo approccio nel modo seguente:

```
let seqId = 0
exports.handler = async (event) => {
  console.log(`Starting: sequence Id=${++seqId}`)
  const result = await doWork(seqId)
  console.log(`Work done: sequence Id=${result}`)
}

function doWork(id) {
  return new Promise(resolve => {
    setTimeout(() => resolve(id), 4000)
  })
}
```

L'utilizzo di questa sintassi impedisce all'handler di uscire prima che la funzione asincrona sia terminata. In questo caso, se il callback richiede più tempo del timeout della funzione Lambda, la funzione genera un errore, invece di restituire il risultato del callback in una chiamata successiva:



1. Il codice richiama la funzione `doWork` asincrona utilizzando la parola chiave `await` nell'handler.
2. Il completamento della funzione `doWork` richiede un certo periodo di tempo prima di risolvere la promessa.
3. La funzione scade perché `doWork` impiega più tempo del limite di timeout consentito e il risultato del callback non viene restituito in una invocazione successiva.

Di solito, è necessario accertarsi che tutti i processi in background e le callback nel codice vengano completate prima che il codice sia terminato. Se ciò non è possibile nel tuo caso d'uso, puoi utilizzare un identificatore per assicurarti che il callback appartenga all'invocazione corrente. A tale scopo, è possibile utilizzare gli `aws.RequestId` forniti dall'oggetto `context`. Passando questo valore al callback asincrono, è possibile confrontare il valore passato con il valore corrente per rilevare se il callback ha avuto origine da un'altra invocazione:

```

let currentContext

exports.handler = async (event, context) => {
  console.log(`Starting: request id=${context.awsRequestId}`)
  currentContext = context

  doWork(context.awsRequestId, function(id) {

```

```

    if (id !== currentContext.awsRequestId) {
        console.info(`This callback is from another invocation.`)
    }
})

}

function doWork(id, callback) {
    setTimeout(() => callback(id), 3000)
}

```

The screenshot displays the AWS Lambda console interface. At the top, there are tabs for 'Function code' and 'Info', along with 'Deploy' and 'Actions' buttons. Below the tabs is a menu bar with 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', 'Window', 'Test', and 'Deploy'. The main area shows the 'index.js' file with the following code:

```

1 let currentContext
2
3 exports.handler = async (event, context) => {
4     console.log(`Starting: request id=${context.awsRequestId}`)
5     currentContext = context
6
7     doWork(context.awsRequestId, function(id) {
8         if (id !== currentContext.awsRequestId) {
9             console.info(`This callback is from another invocation.`)
10        }
11    })
12 }
13
14 function doWork(id, callback) {
15     setTimeout(() => callback(id), 3000)
16 }
17

```

Two red circles with numbers 1 and 2 are overlaid on the code. Circle 1 points to the `currentContext` variable declaration on line 1. Circle 2 points to the `currentContext` variable assignment on line 5. Below the code editor, the 'Execution Result' tab is selected, showing the status 'Succeeded', 'Max Memory Used: 65 MB', and 'Time: 1.28 ms'. The 'Response' is 'null'. The 'Request ID' is 'aa137379-9c11-4e8d-b45b-895930ecca46'. The 'Function logs' section shows the following output:

```

START RequestId: aa137379-9c11-4e8d-b45b-895930ecca46 Version: $LATEST
2020-10-14T12:50:46.765Z aa137379-9c11-4e8d-b45b-895930ecca46 INFO Starting: request id=aa137379-9c11-4e8d-b45b-895930ecca46
2020-10-14T12:50:46.766Z aa137379-9c11-4e8d-b45b-895930ecca46 INFO This callback is from another invocation.
END RequestId: aa137379-9c11-4e8d-b45b-895930ecca46
REPORT RequestId: aa137379-9c11-4e8d-b45b-895930ecca46 Duration: 1.28 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 65 MB

```

1. L'handler della funzione Lambda accetta il parametro `context`, che fornisce l'accesso a un ID di richiesta di invocazione univoco.
2. `awsRequestId` viene passato alla funzione `doWork`. Nel callback, l'ID viene confrontato con `awsRequestId` dell'invocazione corrente. Se questi valori sono diversi, il codice può agire di conseguenza.

## Risoluzione dei problemi relativi alle implementazioni in Lambda

Quando si aggiorna la funzione, Lambda distribuisce la modifica avviando nuove istanze della funzione con il codice o le impostazioni aggiornati. Gli errori di distribuzione impediscono l'utilizzo della nuova versione e possono derivare da problemi relativi al pacchetto di distribuzione, al codice, alle autorizzazioni o agli strumenti.

Quando distribuisce gli aggiornamenti alla tua funzione direttamente con l'API Lambda o con un client come il AWS CLI, puoi vedere gli errori di Lambda direttamente nell'output. Se utilizzi servizi come AWS CloudFormation, o AWS CodeDeploy AWS CodePipeline, cerca la risposta di Lambda nei log o nel flusso di eventi di quel servizio.

Negli argomenti seguenti vengono forniti suggerimenti per la risoluzione dei problemi relativi a errori e problemi che potrebbero verificarsi durante l'utilizzo della console, degli strumenti o dell'API Lambda. Se scopri un problema che non è elencato qui di seguito, puoi utilizzare il pulsante Feedback in questa pagina per segnalarlo.

Per ulteriori suggerimenti sulla risoluzione dei problemi e per risposte a domande comuni relative al supporto, visitare il [Knowledge Center di AWS](#).

Per ulteriori informazioni sul debug e la risoluzione dei problemi delle applicazioni Lambda, consulta [Debug](#) in Serverless Land.

### Argomenti

- [Generale: l'autorizzazione è negata o non è possibile caricare tale file](#)
- [Generale: si verifica un errore quando si chiama il UpdateFunctionCode](#)
- [Amazon S3: codice di errore. PermanentRedirect](#)
- [Generale: impossibile trovare, impossibile caricare, impossibile importare, classe non trovata, file o directory non trovati](#)
- [Generale: handler di metodi non definito](#)
- [Generale: è stato superato il limite di memorizzazione del codice Lambda](#)
- [Lambda: conversione dei livelli non riuscita](#)
- [Lambda: o InvalidParameterValueException RequestEntityTooLargeException](#)
- [Lambda: InvalidParameterValueException](#)
- [Lambda: quote di simultaneità e memoria](#)
- [Lambda: configurazione dell'alias non valida per la concorrenza assegnata](#)

Generale: l'autorizzazione è negata o non è possibile caricare tale file

Errore: EACCES: autorizzazione negata, apri var/task '/index.js '

Errore: cannot load such file -- function (impossibile caricare tale file - funzione)

Errore: [Errno 13] Autorizzazione negata: '/function.py 'var/task

Il runtime Lambda necessita dell'autorizzazione per leggere i file nel pacchetto di distribuzione.

Nella notazione ottale delle autorizzazioni di Linux, Lambda richiede 644 autorizzazioni per i file non eseguibili (rw-r--r--) e 755 autorizzazioni (rwxr-xr-x) per le directory e i file eseguibili.

In Linux e macOS, utilizza il comando `chmod` per modificare le autorizzazioni file su file e directory nel pacchetto di implementazione. Ad esempio, per assegnare a un file non eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 644 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta [Set, View, Change, or Remove Permissions on an Object](#) nella documentazione di Microsoft Windows.

#### Note

Se non concedi a Lambda le autorizzazioni necessarie per accedere alle directory del pacchetto di implementazione, Lambda imposta le autorizzazioni per tali directory su 755 (rwxr-xr-x).

Generale: si verifica un errore quando si chiama il `UpdateFunctionCode`

Errore: si è verificato un errore (`RequestEntityTooLargeException`) durante la chiamata dell'`UpdateFunctionCode` operazione

Quando carichi un pacchetto di distribuzione o un archivio di livelli direttamente in Lambda, la dimensione del file ZIP è limitata a 50 MB. Per caricare un file di dimensioni maggiori, archivalo in Amazon S3 e utilizza i parametri `S3Bucket` e `S3Key`.

#### Note

Quando carichi un file direttamente con AWS SDK o altro AWS CLI, il file ZIP binario viene convertito in base64, il che ne aumenta le dimensioni di circa il 30%. Per consentire questa

operazione e la dimensione di altri parametri nella richiesta, il limite effettivo della dimensione della richiesta Lambda applicabile è maggiore. Per questo motivo, il limite di 50 MB è approssimativo.

## Amazon S3: codice di errore. PermanentRedirect

Errore: si è verificato un errore durante GetObject. Codice di errore S3: PermanentRedirect.

Messaggio di errore S3: il bucket si trova in questa regione: us-east-2. Utilizza questa regione per riprovare la richiesta

Quando carichi il pacchetto di distribuzione di una funzione da un bucket Amazon S3, il bucket deve trovarsi nella stessa regione della funzione. Questo problema può verificarsi quando si specifica un oggetto Amazon S3 in una chiamata o si [UpdateFunctionCode](#) utilizzano i comandi package and deploy nella CLI AWS CLI or. AWS SAM Crea un bucket artefatto di distribuzione per ogni regione in cui sviluppi applicazioni.

Generale: impossibile trovare, impossibile caricare, impossibile importare, classe non trovata, file o directory non trovati

Errore: impossibile trovare il modulo "function"

Errore: cannot load such file -- function (impossibile caricare tale file - funzione)

Errore: impossibile importare il modulo "function"

Errore: classe non trovata: Function.Handler

Errore: fork/exec /var/task/function: nessun file o directory di questo tipo

Errore: impossibile caricare il tipo 'Function.Handler' dall'insieme 'Funzione'.

Il nome del file o della classe nella configurazione dell'handler della funzione non corrisponde al codice. Per ulteriori informazioni, consulta la sezione seguente.

Generale: handler di metodi non definito

Errore: index.handler non è definito o non esportato

Errore: Handler "handler" mancante sul modulo "function"

Errore: metodo «handler» non definito per #< > LambdaHandler:0x000055b76cceb98

Errore: Nessun metodo pubblico denominato HandleRequest con firma del metodo appropriata trovato sulla classe function.Handler

Errore: impossibile trovare il metodo 'handleRequest' nel tipo " dall'assembly 'Function' Function.Handler

Il nome del metodo dell'handler nella configurazione dell'handler della funzione non corrisponde al codice. Ogni runtime definisce una convenzione di denominazione per i gestori, ad esempio. *filename.methodname*. L'handler è il metodo nel codice della funzione che il runtime esegue quando viene invocata la funzione.

Per alcune lingue, Lambda fornisce una libreria con un'interfaccia che prevede un metodo handler per avere un nome specifico. Per informazioni dettagliate sulla denominazione dell'handler per ogni lingua, consulta i seguenti argomenti.

- [Creazione di funzioni Lambda con Node.js](#)
- [Compilazione di funzioni Lambda con Python](#)
- [Compilazione di funzioni Lambda con Ruby](#)
- [Compilazione di funzioni Lambda con Java](#)
- [Compilazione di funzioni Lambda con Go](#)
- [Compilazione di funzioni Lambda con C#](#)
- [Creazione di funzioni Lambda con PowerShell](#)

Generale: è stato superato il limite di memorizzazione del codice Lambda

Errore: limite di archiviazione del codice superato.

Lambda memorizza una copia del codice funzione in un bucket S3 interno privato del tuo account. A ogni AWS account sono assegnati 300 GB di spazio di archiviazione del codice (decompresso) in ogni regione. Lo spazio di archiviazione del codice include lo spazio di archiviazione totale utilizzato sia dalle funzioni che dai livelli Lambda. Se raggiungi la quota, ne riceverai uno `CodeStorageExceededException` quando tenti di implementare nuove funzioni.

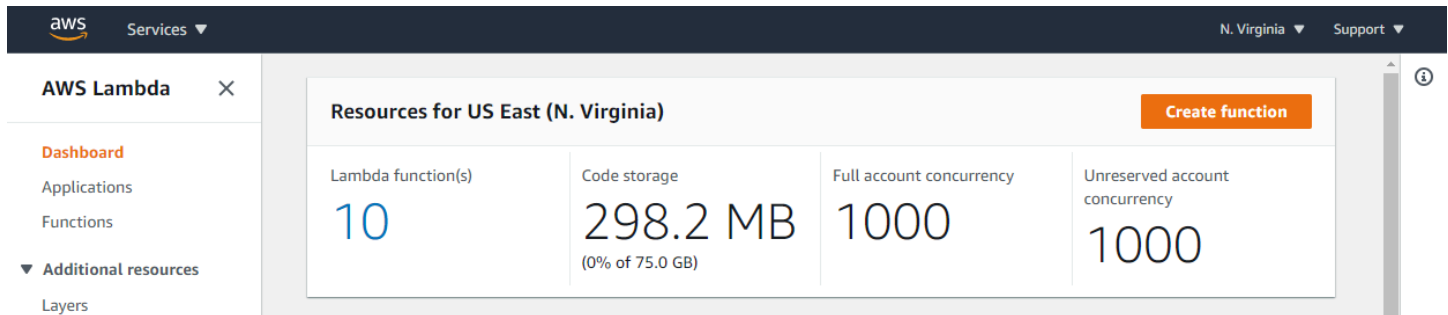
Per risolvere questo problema, puoi ripulire le vecchie versioni delle funzioni, rimuovere il codice inutilizzato o utilizzare i livelli Lambda per deduplicare il codice comune condiviso tra le funzioni.



Inoltre, è buona norma [utilizzare AWS account separati per carichi di lavoro separati per facilitare la gestione delle quote](#) di archiviazione.

In alternativa, puoi configurare le funzioni e i livelli per utilizzare l'archiviazione di codice S3 [autogestita](#). Con lo storage di codice S3 autogestito, Lambda fa riferimento al codice direttamente dal bucket S3, il che non viene conteggiato nella quota di archiviazione.

Puoi visualizzare l'utilizzo totale dello spazio di archiviazione nella console Lambda, nel sottomenu Dashboard:



## Lambda: conversione dei livelli non riuscita

Errore: conversione dei livelli Lambda non riuscita. Per consigli sulla risoluzione di questo problema, consulta la pagina Risoluzione dei problemi di implementazione in Lambda nella Guida per l'utente di Lambda.

Quando si configura una funzione Lambda con un livello, Lambda unisce il livello con il codice della funzione. Se questo processo non viene completato, Lambda restituisce questo errore. Se lo fa, procedere come indicato di seguito:

- Eliminare tutti i file inutilizzati dal livello
- Eliminare tutti i collegamenti simbolici nel tuo livello
- Rinominare tutti i file che hanno lo stesso nome di una directory in uno qualsiasi dei livelli della funzione

## Lambda: InvalidParameterValueException RequestEntityTooLargeException

Errore InvalidParameterValueException: Lambda non è riuscita a configurare le variabili di ambiente perché le variabili di ambiente fornite hanno superato il limite di 4 KB. Stringa misurata: {"A1": "u... SFeY5cyPiPn7AtnX5BsM

**Errore RequestEntityTooLargeException:** la richiesta deve essere inferiore a 5120 byte per l'operazione UpdateFunctionConfiguration

La dimensione massima dell'oggetto variabili memorizzato nella configurazione della funzione non deve superare 4096 byte. Sono inclusi nomi chiave, valori, virgolette, virgole e parentesi. Anche la dimensione totale del corpo della richiesta HTTP è limitata.

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "Runtime": "nodejs24.x",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Environment": {
    "Variables": {
      "BUCKET": "amzn-s3-demo-bucket",
      "KEY": "file.txt"
    }
  },
  ...
}
```

In questo esempio, l'oggetto è di 39 caratteri e quando archiviato occupa 39 byte (senza spazi) come stringa `{"BUCKET": "amzn-s3-demo-bucket", "KEY": "file.txt"}`. I caratteri ASCII standard nei valori delle variabili di ambiente utilizzano un byte ciascuno. I caratteri ASCII e Unicode estesi possono utilizzare tra 2 e 4 byte per carattere.

## Lambda: InvalidParameterValueException

**Errore InvalidParameterValueException:** Lambda non è riuscita a configurare le variabili di ambiente perché le variabili di ambiente fornite contengono chiavi riservate che attualmente non sono supportate per la modifica.

Lambda riserva alcune chiavi di variabili di ambiente per uso interno. Ad esempio, `AWS_REGION` viene utilizzata dal runtime per determinare la regione corrente e non può essere sovrascritta. Altre variabili, come `PATH`, sono utilizzate dal runtime ma possono essere estese nella configurazione della funzione. Per un elenco completo, consultare [Variabili di ambiente di runtime definite](#).

## Lambda: quote di simultaneità e memoria

**Errore: ConcurrentExecutions** la funzione specificata riduce l'account al di `UnreservedConcurrentExecution` sotto del valore minimo

Errore: il valore `MemorySize` " non soddisfa il vincolo: il membro deve avere un valore inferiore o uguale a 3008

Questi errori si verificano quando superi le [quote](#) di simultaneità o memoria per il tuo account. AWS I nuovi account hanno ridotto la concorrenza e le quote di memoria. Per risolvere gli errori relativi alla simultaneità, puoi [richiedere un aumento della quota](#). Non è possibile richiedere un aumento della quota.

- Simultaneità: se provi a creare una funzione utilizzando la simultaneità riservata o con provisioning o se la tua richiesta di simultaneità per funzione ([PutFunctionConcurrency](#)) supera la quota di simultaneità del tuo account potresti ricevere un errore.
- Memoria: se la quantità di memoria allocata per la funzione supera la quota di memoria dell'account si verificano degli errori.

## Lambda: configurazione dell'alias non valida per la concorrenza assegnata

Errore: configurazione dell'alias non valida per la concorrenza assegnata

Questo errore si verifica quando si tenta di aggiornare il codice o la configurazione di una funzione Lambda mentre un alias con concorrenza assegnata punta a una versione che presenta problemi. Lambda preinizializza gli ambienti di esecuzione per la concorrenza su cui è stato eseguito il provisioning e, se questi ambienti non possono essere inizializzati correttamente a causa di errori di codice, vincoli di risorse o stack e alias interessati, la distribuzione ha esito negativo. Se riscontri questo problema, procedi nel seguente modo:

1. Ripristina l'alias: aggiorna temporaneamente l'alias in modo che punti alla versione funzionante in precedenza.

```
aws lambda update-alias \  
  --function-name <function-name> \  
  --name <alias-name> \  
  --function-version <known-good-version>
```

2. Correggi il codice di inizializzazione Lambda: assicurati che il codice di inizializzazione eseguito all'esterno del gestore non contenga eccezioni non rilevate e inizializza i client e le connessioni.
3. Reimplementa la sicurezza: distribuisci codice fisso e pubblica una nuova versione. Quindi, aggiorna l'alias in modo che punti alla versione fissa. Facoltativamente, riabilita la concorrenza [assegnata, se necessario](#).

Se lo usi AWS CloudFormation, aggiorna la definizione dello stack `FunctionVersion: !GetAtt version.Version` in modo che l'alias punti alla versione funzionante:

```
alias:
  Type: AWS::Lambda::Alias
  Properties:
    FunctionName: !Ref function
    FunctionVersion: !GetAtt version.Version
    Name: BLUE
    ProvisionedConcurrencyConfig:
    ProvisionedConcurrentExecutions: 1
```

## Risoluzione dei problemi di invocazione in Lambda

Quando si richiama una funzione Lambda, Lambda convalida la richiesta e verifica la capacità di ridimensionamento prima di inviare l'evento alla funzione o, per la invocazione asincrona, alla coda eventi. Gli errori di invocazione possono essere causati da problemi relativi ai parametri di richiesta, alla struttura degli eventi, alle impostazioni delle funzioni, alle autorizzazioni utente, alle autorizzazioni delle risorse o alle restrizioni.

Se invochi direttamente la funzione, visualizzi eventuali errori di invocazione nella risposta da Lambda. Se si richiama la funzione in modo asincrono, con un mapping di origine eventi o tramite un altro servizio, è possibile che vengano riscontrati errori nei log, nella coda DLQ o in una destinazione in caso di errore. Le opzioni di gestione degli errori e il comportamento dei tentativi variano a seconda di come si richiama la funzione e del tipo di errore.

Per un elenco di tipi di errore che l'operazione `Invoke` può restituire, consulta [Invoke \(Invoca\)](#).

### Argomenti

- [Lambda: la funzione scade durante la fase Init \(\) Sandbox.Timeout](#)
- [IAM: lambda: non autorizzato InvokeFunction](#)
- [Lambda: impossibile trovare bootstrap \(\) valido Runtime.InvalidEntrypoint](#)
- [Lambda: l'operazione non può essere eseguita ResourceConflictException](#)
- [Lambda: la funzione è bloccata in sospeso](#)
- [Lambda: una funzione sta usando tutta la simultaneità](#)
- [Generale: impossibile richiamare la funzione con altri account o servizi](#)
- [Generale: il richiamo della funzione è in loop](#)

- [Lambda: routing alias con simultaneità fornita](#)
- [Lambda: avvii a freddo con simultaneità fornita](#)
- [Lambda: avvii a freddo con nuove versioni](#)
- [Lambda: Node.js uscita inaspettata in runtime \(\) Runtime.NodejsExit](#)
- [EFS: la funzione non è in grado di montare il file system EFS](#)
- [EFS: la funzione non è in grado di connettersi al file system EFS](#)
- [EFS: la funzione non è in grado di montare il file system EFS a causa del timeout](#)
- [File S3: la funzione non può montare il file system S3](#)
- [File S3: la funzione non può connettersi al file system S3](#)
- [File S3: la funzione non può montare il file system S3 a causa del timeout](#)
- [Lambda: Lambda ha rilevato un processo IO che stava impiegando troppo tempo](#)
- [Contenitore: CodeArtifactUserException errori](#)
- [Contenitore: InvalidEntrypoint errori](#)

Lambda: la funzione scade durante la fase Init () Sandbox.Timeout

Errore: Timeout dell'attività dopo 3 secondi

Quando la fase [Init](#) scade, Lambda inizializza nuovamente l'ambiente di esecuzione rieseguendo la fase `Init` all'arrivo della successiva richiesta di invocazione. Questa operazione è chiamata [inizializzazione soppressa](#). Tuttavia, se la funzione è configurata con una [durata di timeout](#) breve (in genere circa 3 secondi), l'init soppresso potrebbe non essere completato durante il periodo di timeout assegnato, causando un nuovo timeout della fase `Init`. In alternativa, l'init soppresso viene completato ma non lascia abbastanza tempo per il completamento della fase [Invoke](#), causando il timeout della fase `Invoke`.

Per ridurre gli errori di timeout, utilizza una delle seguenti strategie:

- Aumenta la durata del timeout della funzione: estendi il [timeout](#) per dare più tempo per il completamento delle fasi `Init` e `Invoke`.
- Aumenta l'allocazione della memoria della funzione: una maggiore [quantità di memoria](#) significa anche una maggiore allocazione proporzionale della CPU, che può velocizzare entrambe le fasi `Init` e `Invoke`.
- Ottimizza il codice di inizializzazione della funzione: riduci il tempo necessario per l'inizializzazione per garantire che le fasi `Init` e `Invoke` possano essere completate entro il timeout configurato.

## IAM: lambda: non autorizzato InvokeFunction

Errore: l'utente: arn:aws:iam: :123456789012: non user/developer è autorizzato a eseguire: lambda: on resource: my-function InvokeFunction

L'utente, o il ruolo che assumi, deve disporre dell'autorizzazione per invocare una funzione. Questo requisito si applica anche alle funzioni Lambda e ad altre risorse di calcolo che richiamano funzioni. Aggiungi la policy AWS gestita al tuo utente o aggiungi `AWSLambdaRole` una policy personalizzata che consenta l'azione sulla funzione di destinazione. `lambda:InvokeFunction`

### Note

Il nome dell'azione IAM (`lambda:InvokeFunction`) si riferisce all'operazione dell'API di Lambda `Invoke`.

Per ulteriori informazioni, consulta [Gestione delle autorizzazioni in AWS Lambda](#).

## Lambda: impossibile trovare bootstrap () valido Runtime.InvalidEntrypoint

Errore: impossibile trovare bootstrap validi: [//bootstrap/var/task] opt/bootstrap

Questo errore si verifica in genere quando la root del pacchetto di implementazione non contiene un file eseguibile denominato `bootstrap`. Ad esempio, se desideri implementare una funzione `provided.al2023` con un file `.zip`, il file `bootstrap` deve trovarsi nella root del file `.zip`, non in una directory.

## Lambda: l'operazione non può essere eseguita ResourceConflictException

Errore `ResourceConflictException`: l'operazione non può essere eseguita in questo momento. La funzione è attualmente nello stato seguente: Pending (In sospeso)

Quando si connette una funzione a un VPC al momento della creazione, la funzione entra in uno stato `Pending` mentre Lambda crea le interfacce di rete elastica. Durante questo periodo, non è possibile richiamare o modificare la funzione. Se si connette la funzione a un VPC dopo la creazione, è possibile richiamarla mentre l'aggiornamento è in sospeso, ma non è possibile modificarne il codice o la configurazione.

Per ulteriori informazioni, consulta [Stati funzione Lambda](#).

## Lambda: la funzione è bloccata in sospeso

Errore: una funzione è bloccata nello stato *Pending* per diversi minuti.

Se una funzione è bloccata nello stato *Pending* per più di sei minuti, invoca una delle seguenti operazioni API per sbloccarla.

- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)
- [PublishVersion](#)

Lambda annulla l'operazione in sospeso e imposta la funzione allo stato *Failed*. Puoi tentare, quindi, un altro aggiornamento.

## Lambda: una funzione sta usando tutta la simultaneità

Problema: una funzione utilizza tutta la simultaneità disponibile, causando la limitazione di altre funzioni.

Per dividere la concorrenza disponibile del tuo AWS account in una AWS regione in pool, utilizza la concorrenza [???](#) riservata. La simultaneità riservata garantisce che una funzione possa sempre scalare alla relativa simultaneità assegnata e che non scalerà oltre la simultaneità assegnata.

## Generale: impossibile richiamare la funzione con altri account o servizi

Problema: è possibile invocare la funzione direttamente, ma questa non viene eseguita quando un altro servizio o account la invoca.

Puoi concedere [ad altri servizi](#) e account l'autorizzazione a invocare una funzione nella [policy basata sulle risorse](#) della funzione. Se l'invoker si trova in un altro account, tale utente deve anche disporre dell'[autorizzazione per invocare le funzioni](#).

## Generale: il richiamo della funzione è in loop

Problema: la funzione viene richiamata continuamente in un loop.

Ciò si verifica in genere quando la funzione gestisce le risorse nello stesso AWS servizio che la attiva. Ad esempio, è possibile creare una funzione che memorizza un oggetto in un bucket Amazon Simple Storage Service (Amazon S3) configurato con una [notifica che richiama nuovamente la funzione](#). Per interrompere l'esecuzione della funzione, riduci la [simultaneità](#) a zero, il che limita tutte le invocazioni future.

Quindi identificare il percorso del codice o l'errore di configurazione che ha causato la chiamata ricorsiva. Lambda rileva e interrompe automaticamente i loop ricorsivi per alcuni servizi e SDK. AWS Per ulteriori informazioni, consulta [the section called “Rilevamento di un ciclo ricorsivo”](#).

## Lambda: routing alias con simultaneità fornita

Problema: invocazioni spillover di concorrenza con provisioning durante il routing degli alias.

Lambda utilizza un modello probabilistico semplice per distribuire il traffico tra le due versioni delle funzioni. A livelli di traffico bassi, è possibile che si verifichi una variazione elevata tra la percentuale di traffico configurata e quella effettiva in ciascuna versione. Se la tua funzione utilizza la concorrenza con provisioning, puoi evitare [invocazioni spillover](#) configurando un numero maggiore di istanze di concorrenza sottoposte a provisioning durante il periodo in cui il routing degli alias è attivo.

## Lambda: avvii a freddo con simultaneità fornita

Problema: si verificano avvii a freddo dopo che è stata abilitata la simultaneità fornita.

Quando il numero di esecuzioni simultanee su una funzione è minore o uguale al [livello configurato di simultaneità fornita](#), non dovrebbero verificarsi avvii a freddo. Per confermare se la simultaneità fornita funziona normalmente, effettuare le seguenti operazioni:

- [Verificare che la simultaneità fornita sia abilitata](#) sulla versione o sull'alias della funzione.

### Note

La simultaneità fornita non è configurabile nella [versione non pubblicata della funzione](#) (\$LATEST).

- Assicurarsi che i trigger richiamino la versione o l'alias della funzione corretti. Ad esempio, se si utilizza Amazon API Gateway, verificare che API Gateway richiami la versione o l'alias della funzione con la simultaneità fornita, non \$LATEST. Per confermare che viene utilizzata la concorrenza predisposta, puoi controllare la metrica di Amazon. [ProvisionedConcurrencyInvocations CloudWatch](#) Un valore diverso da zero indica che la funzione sta elaborando le chiamate in ambienti di esecuzione inizializzati.
- Determina se la concorrenza delle tue funzioni supera il livello configurato di concorrenza assegnato controllando la metrica. [ProvisionedConcurrencySpilloverInvocations CloudWatch](#) Un valore diverso da zero indica che tutta la simultaneità fornita è in uso e che si è verificata una chiamata con un avvio a freddo.



- Verifica la [frequenza di chiamata](#) (richieste al secondo). Le funzioni con simultaneità fornita hanno un tasso massimo di 10 richieste al secondo per ogni simultaneità fornita. Ad esempio, una funzione configurata con 100 simultaneità fornita può gestire 1.000 richieste al secondo. Se la frequenza di chiamata supera le 1.000 richieste al secondo, è possibile che si verifichino alcuni avvii a freddo.

## Lambda: avvii a freddo con nuove versioni

Problema: si verificano avvii a freddo durante la distribuzione di nuove versioni della funzione.

Quando si aggiorna un alias di funzione, Lambda sposta automaticamente la simultaneità fornita alla nuova versione in base ai pesi configurati sull'alias.

Errore `KMSDisabledException`: Lambda non è riuscita a decrittografare le variabili di ambiente perché la chiave KMS utilizzata è disabilitata. Controlla le impostazioni delle chiavi KMS della funzione.

Questo errore può verificarsi se la chiave AWS Key Management Service (AWS KMS) è disabilitata o se la concessione che consente a Lambda di utilizzare la chiave viene revocata. Se l'autorizzazione manca, configurare la funzione per utilizzare una chiave diversa. Quindi, riassegnare la chiave personalizzata per ricreare l'autorizzazione.

## Lambda: Node.js uscita inaspettata in runtime () Runtime.NodejsExit

Problema: il client runtime Lambda ha rilevato un codice di Node.js uscita imprevisto.

Questo errore si verifica quando la funzione viene chiusa prima che tutte le Promises siano state risolte, ad esempio a causa di un bug nel codice. Può verificarsi anche quando Node.js rileva una situazione di stallo che impedisce la risoluzione delle promesse. Questo errore riguarda solo i gestori in stile asincrono, non i gestori in stile callback.

Node.js Runtime interessati: 18 e versioni successive.

Per risolvere questo problema:

1. Controlla il codice della funzione per verificare la presenza di promesse non risolte nei gestori asincroni.
2. Assicurati che tutte le promesse siano risolte correttamente (risolte o rifiutate) prima che la funzione venga completata.
3. Controlla il tuo codice per individuare eventuali condizioni di gara nelle operazioni asincrone.

Per ulteriori informazioni sui codici di Node.js uscita e sull'interruzione del processo, consulta la documentazione. [Node.js](#)

## EFS: la funzione non è in grado di montare il file system EFS

Errore `EFSMountFailureException`: la funzione non è riuscita a montare il file system EFS con il punto di accesso `arn:aws:elasticfilesystem:us-east-2:123456789012:access-015cxmplb72b405fd.point/fsap`

La richiesta di montaggio al file system della funzione è stata rifiutata. Verificare le autorizzazioni della funzione e verificare che il file system e il punto di accesso esistano e siano pronti per l'uso.

## EFS: la funzione non è in grado di connettersi al file system EFS

Errore `EFSMountConnectivityException`: la funzione non è riuscita a connettersi al file system Amazon EFS con il punto di accesso `arn:aws:elasticfilesystem:us-east-2:123456789012:access-015cxmplb72b405fd.point/fsap`. Controlla la configurazione di rete e riprova.

La funzione non è in grado di stabilire una connessione al file system della funzione con il protocollo NFS (porta TCP 2049). Controllare la [configurazione del gruppo di sicurezza e del routing](#) per le sottoreti del VPC.

Se riscontri questi errori dopo aver aggiornato le impostazioni di configurazione VPC della funzione, prova a smontare e rimontare il file system.

## EFS: la funzione non è in grado di montare il file system EFS a causa del timeout

Errore `EFSMountTimeoutException`: la funzione non è riuscita a montare il file system EFS con access point `{arn:aws:elasticfilesystem:us-east-2:123456789012:access-015cxmplb72b405fd}` a causa del timeout di montaggio. `point/fsap`

La funzione è stata in grado di connettersi al file system della funzione, ma l'operazione di montaggio è scaduta. Riprovare dopo un po' di tempo e considerare di limitare la [simultaneità](#) della funzione per ridurre il carico sul file system.

## File S3: la funzione non può montare il file system S3

Errore `S3FilesMountFailureException`: la funzione non è riuscita a montare il file system Amazon S3 con il punto di accesso `arn:aws:s3files:us-east-2:123456789012:access-123456789abcde.point/fsap`

La richiesta di montaggio al file system della funzione è stata rifiutata. Verificare le autorizzazioni della funzione e verificare che il file system e il punto di accesso esistano e siano pronti per l'uso.

## File S3: la funzione non può connettersi al file system S3

Errore `S3FilesMountConnectivityException`: la funzione non è riuscita a connettersi al file system Amazon S3 con il punto di accesso `arn:aws:s3files:us-east-2:123456789012:access-123456789abcde`. point/fsap Controlla la configurazione di rete e riprova.

La funzione non è in grado di stabilire una connessione al file system della funzione con il protocollo NFS (porta TCP 2049). Controllare la [configurazione del gruppo di sicurezza e del routing](#) per le sottoreti del VPC.

Se riscontri questi errori dopo aver aggiornato le impostazioni di configurazione VPC della funzione, prova a smontare e rimontare il file system.

## File S3: la funzione non può montare il file system S3 a causa del timeout

Errore `S3FilesMountTimeoutException`: la funzione non è riuscita a montare il file system S3 con il punto di accesso `{arn:aws:s3files:us-east-2:123456789012:access-123456789abcde}` a causa del timeout di montaggio. point/fsap

La funzione è stata in grado di connettersi al file system della funzione, ma l'operazione di montaggio è scaduta. Riprovare dopo un po' di tempo e considerare di limitare la [simultaneità](#) della funzione per ridurre il carico sul file system.

## Lambda: Lambda ha rilevato un processo IO che stava impiegando troppo tempo

`EFSIOException`: questa istanza della funzione è stata arrestata perché Lambda ha rilevato un processo I/O che richiede troppo tempo.

Si è verificato il timeout di una chiamata precedente e Lambda non è stata in grado di terminare l'handler della funzione. Questo problema può verificarsi quando un file system collegato esaurisce i crediti di burst e il throughput di base è insufficiente. Per aumentare il throughput, è possibile aumentare le dimensioni del file system o utilizzare il throughput assegnato.

## Contenitore: `CodeArtifactUserException` errori

Errore: messaggio `CodeArtifactUserPendingException` di errore

L'ottimizzazione CodeArtifact è in sospeso. La funzione passa allo [stato attivo](#) quando Lambda completa l'ottimizzazione. Codice di risposta HTTP: 409.

Errore: messaggio CodeArtifactUserDeletedException di errore

CodeArtifact È programmato per essere eliminato. Codice di risposta HTTP: 409.

Errore: messaggio CodeArtifactUserFailedException di errore

Lambda non è riuscito a ottimizzare il codice. Correggere il codice e caricarlo di nuovo. Codice di risposta HTTP: 409.

## Contenitore: InvalidEntrypoint errori

Errore: Runtime.ExitError o «errorType»: "" Runtime.InvalidEntrypoint

Verificare che l'elemento ENTRYPOINT dell'immagine container includa il percorso assoluto come posizione. Verificare inoltre che l'immagine non contenga un collegamento simbolico come ENTRYPOINT.

Errore: stai utilizzando un CloudFormation modello e il tuo contenitore ENTRYPOINT viene sovrascritto con un valore nullo o vuoto.

Rivedi la [ImageConfig](#) risorsa nel CloudFormation modello. Se si dichiara una ImageConfig risorsa nel modello, è necessario fornire valori non vuoti per tutte e tre le proprietà.

## Risoluzione dei problemi di esecuzione in Lambda

Quando il runtime Lambda esegue il codice della funzione, l'evento potrebbe essere elaborato su un'istanza della funzione che ha elaborato eventi per un determinato periodo o potrebbe richiedere l'inizializzazione di una nuova istanza. Gli errori possono verificarsi durante l'inizializzazione della funzione, quando il codice dell'handler elabora l'evento o quando la funzione restituisce (o non riesce a restituire) una risposta.

Gli errori di esecuzione delle funzioni possono essere causati da problemi relativi al codice, alla configurazione delle funzioni, alle risorse downstream o alle autorizzazioni. Se si invoca direttamente la funzione, si ricevono errori di funzione nella risposta da Lambda. Se si richiama la funzione in modo asincrono, con un mapping di origine eventi o tramite un altro servizio, è possibile che vengano riscontrati errori nei log, nella coda DLQ o in una destinazione in caso di errore. Le opzioni di gestione degli errori e il comportamento dei tentativi variano a seconda di come si richiama la funzione e del tipo di errore.

Quando il codice della funzione o il runtime Lambda restituiscono un errore, il codice di stato nella risposta da Lambda è 200 OK. La presenza di un errore nella risposta è indicata da un'intestazione denominata `X-Amz-Function-Error`. I codici di stato serie 400 e 500 sono riservati agli [errori di chiamata](#).

## Argomenti

- [Lambda: debug remoto con Visual Studio Code](#)
- [Lambda: l'esecuzione richiede troppo tempo](#)
- [Lambda: payload per eventi imprevisti](#)
- [Lambda: payload di dimensioni inaspettatamente elevate](#)
- [Lambda: errori di codifica e decodifica JSON](#)
- [Lambda: i log o le tracce non vengono visualizzati](#)
- [Lambda: non vengono visualizzati tutti i log della mia funzione](#)
- [Lambda: la funzione restituisce prima del termine dell'esecuzione](#)
- [Lambda: esecuzione di una versione o di un alias di una funzione non intenzionale](#)
- [Lambda: rilevamento di loop infiniti](#)
- [Generale: indisponibilità del servizio downstream](#)
- [AWS SDK: versioni e aggiornamenti](#)
- [Python: caricamento librerie errato](#)
- [Java: la funzione impiega più tempo per elaborare gli eventi dopo l'aggiornamento a Java 17 da Java 11](#)
- [Kafka: problemi relativi alla gestione degli errori e ai nuovi tentativi di configurazione](#)

## Lambda: debug remoto con Visual Studio Code

**Problema:** difficoltà nella risoluzione dei problemi relativi al comportamento complesso delle funzioni Lambda nell'ambiente reale AWS

Lambda fornisce una funzionalità di debug remoto tramite AWS Toolkit for Visual Studio Code. Per la configurazione e le istruzioni generali, consulta [Esecuzione da remoto del debug delle funzioni Lambda con Visual Studio Code](#).

Per istruzioni dettagliate sulla risoluzione dei problemi, sui casi d'uso avanzati e sulla disponibilità regionale, consulta le funzioni di debug [remoto di Lambda nella Guida per l'utente](#). AWS Toolkit for Visual Studio Code

## Lambda: l'esecuzione richiede troppo tempo

Problema: l'esecuzione della funzione richiede troppo tempo.

Se il codice richiede molto più tempo per essere eseguito in Lambda rispetto al computer locale, potrebbe essere limitato dalla memoria o dalla potenza di elaborazione disponibile per la funzione.

[Configurare la funzione con memoria aggiuntiva](#) per aumentare sia la memoria sia la CPU.

## Lambda: payload per eventi imprevisti

Problema: errori di funzione legati a un JSON non valido o a una convalida dei dati inadeguata.

Tutte le funzioni Lambda ricevono un payload di eventi nel primo parametro dell'handler. Il payload dell'evento è una struttura JSON che può contenere array ed elementi annidati.

Un JSON non valido può verificarsi se fornito da servizi upstream che non utilizzano un processo affidabile per il controllo delle strutture JSON. Ciò si verifica quando i servizi concatenano stringhe di testo o incorporano input dell'utente che non sono stati eliminati. JSON viene inoltre spesso serializzato per il passaggio da un servizio all'altro. Analizza sempre le strutture JSON sia come producer che come consumer di JSON per assicurarti che la struttura sia valida.

Analogamente, il mancato controllo degli intervalli di valori nel payload dell'evento può causare errori. Nell'esempio seguente viene illustrata una funzione che calcola le ritenute d'acconto:

```
exports.handler = async (event) => {  
  let pct = event.taxPct  
  let salary = event.salary  
  
  // Calculate % of paycheck for taxes  
  return (salary * pct)  
}
```

Questa funzione utilizza lo stipendio e l'aliquota fiscale del payload dell'evento per eseguire il calcolo. Tuttavia, il codice non riesce a verificare se gli attributi sono presenti. Inoltre, non riesce a controllare i tipi di dati o a garantire i limiti, ad esempio garantendo che la percentuale fiscale sia compresa tra 0 e 1. Di conseguenza, i valori al di fuori di questi limiti producono risultati privi di senso. Un tipo errato o un attributo mancante causa un errore di runtime.

Crea test per assicurarti che la tua funzione gestisca dimensioni di payload maggiori. La dimensione massima per il payload di un evento Lambda è 1 MB. A seconda del contenuto, payload più grandi

potrebbero significare più elementi passati alla funzione o più dati binari incorporati in un attributo JSON. In entrambi i casi, ciò può comportare una maggiore elaborazione per una funzione Lambda.

Payload più grandi possono anche causare dei timeout. Ad esempio, una funzione Lambda elabora un record ogni 100 ms e ha un timeout di 3 secondi. L'elaborazione ha esito positivo per 0-29 elementi nel payload. Tuttavia, se il payload contiene più di 30 elementi, la funzione scade e genera un errore. Per evitare ciò, assicurati che siano impostati dei timeout per gestire il tempo di elaborazione aggiuntivo per il numero massimo di elementi previsto.

## Lambda: payload di dimensioni inaspettatamente elevate

**Problema:** le funzioni stanno scadendo o causano errori a causa di payload di grandi dimensioni.

Payload più grandi possono causare timeout ed errori. Consigliamo di creare test per garantire che la funzione gestisca i payload massimi previsti e che il timeout della funzione sia impostato correttamente.

Inoltre, alcuni payload di eventi possono contenere puntatori ad altre risorse. Ad esempio, una funzione Lambda con 128 MB di memoria potrebbe eseguire l'elaborazione delle immagini su un file JPG archiviato come oggetto in S3. La funzione funziona come previsto con file di immagine più piccoli.

Tuttavia, quando viene fornito un file JPG più grande come input, la funzione Lambda genera un errore dovuto all'esaurimento della memoria. Per evitare ciò, i test case dovrebbero includere esempi relativi ai limiti superiori delle dimensioni dei dati previste. Il codice dovrebbe inoltre convalidare le dimensioni del payload.

## Lambda: errori di codifica e decodifica JSON

**Problema:** NoSuchKey eccezione durante l'analisi degli input JSON.

Verifica di elaborare correttamente gli attributi JSON. Ad esempio, per gli eventi generati da S3, l'attributo `s3.object.key` contiene un nome chiave dell'oggetto con codifica URL. Molte funzioni elaborano questo attributo come testo per caricare l'oggetto S3 di riferimento:

### Example

```
const originalText = await s3.getObject({
  Bucket: event.Records[0].s3.bucket.name,
  Key: event.Records[0].s3.object.key
}).promise()
```

Questo codice funziona con il nome della chiave `james.jpg` ma genera un errore `NoSuchKey` quando il nome è `james beswick.jpg`. Poiché la codifica URL converte gli spazi e gli altri caratteri in un nome chiave, è necessario assicurarsi che le funzioni decodifichino le chiavi prima di utilizzare questi dati:

### Example

```
const originalText = await s3.getObject({
  Bucket: event.Records[0].s3.bucket.name,
  Key: decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "))
}).promise()
```

## Lambda: i log o le tracce non vengono visualizzati

Problema: i log non vengono visualizzati nei CloudWatch log.

Problema: le tracce non vengono visualizzate in AWS X-Ray.

La tua funzione richiede l'autorizzazione per chiamare CloudWatch Logs e X-Ray. Aggiornare il [ruolo di esecuzione](#) per concedere a esso l'autorizzazione. Aggiungere le policy gestite seguenti per abilitare i registri e l'analisi.

- `AWSLambdaBasicExecutionRole`
- `AWSXRayDaemonWriteAccess`

Quando aggiungi le autorizzazioni alla funzione, aggiorni anche il relativo codice o la configurazione. Questa operazione forza l'arresto e la sostituzione delle istanze della funzione in esecuzione che hanno credenziali obsolete.

### Note

Potrebbero essere necessari da 5 a 10 minuti prima che i log vengano visualizzati dopo la chiamata di una funzione.

## Lambda: non vengono visualizzati tutti i log della mia funzione

Problema: i log delle funzioni non sono presenti nei CloudWatch log, anche se le mie autorizzazioni sono corrette



Se Account AWS raggiungi i limiti della quota di [CloudWatch Logs](#), limita la CloudWatch registrazione delle funzioni. In questo caso, alcuni dei log generati dalle tue funzioni potrebbero non essere visualizzati in Logs. CloudWatch

Se la tua funzione genera i log a una velocità troppo elevata perché Lambda possa elaborarli, ciò può anche far sì che gli output dei log non vengano visualizzati nei Logs. CloudWatch Quando Lambda non è in grado di inviare i log CloudWatch alla velocità con cui la funzione li produce, elimina i log per evitare che l'esecuzione della funzione rallenti. Aspettati di osservare costantemente i log eliminati quando il throughput dei log supera i 2 MB/s per un singolo flusso di log.

Se la tua funzione è configurata per utilizzare log in formato [JSON](#), Lambda tenta di inviare un [logsDropped](#) evento a CloudWatch Logs quando elimina i log. Tuttavia, quando CloudWatch limita la registrazione della funzione, questo evento potrebbe non raggiungere CloudWatch Logs, quindi non vedrai sempre un record quando Lambda elimina i log.

Per verificare se hai Account AWS raggiunto i limiti della quota di CloudWatch Logs, procedi come segue:

1. Apri la [console Service Quotas](#).
2. Nel pannello di navigazione, scegli Servizi AWS (servizi AWS ).
3. Dall'elenco dei AWS servizi, cerca Amazon CloudWatch Logs.
4. Nell'elenco Service Quotas, scegli le quote `CreateLogGroup throttle limit in transactions per second`, `CreateLogStream throttle limit in transactions per second` e `PutLogEvents throttle limit in transactions per second` per visualizzarne l'utilizzo.

Puoi anche impostare CloudWatch allarmi per avvisarti quando l'utilizzo del tuo account supera il limite specificato per queste quote. Per ulteriori informazioni, consulta [Creare un CloudWatch allarme basato su una soglia](#) statica.

Se i limiti di quota predefiniti per CloudWatch i log non sono sufficienti per il tuo caso d'uso, puoi [richiedere un aumento della quota](#).

**Lambda:** la funzione restituisce prima del termine dell'esecuzione

**Problema:** (Node.js) La funzione viene restituita prima che il codice termini l'esecuzione

Molte librerie, incluso l' AWS SDK, funzionano in modo asincrono. Quando si effettua una chiamata di rete o si esegue un'altra operazione che richiede l'attesa di una risposta, le librerie restituiscono un oggetto chiamato promessa che tiene traccia dell'avanzamento dell'operazione in background.

Per attendere che la promessa si risolva in una risposta, utilizzare la parola chiave `await`. Questo blocca l'esecuzione del codice dell'handler fino a quando la promessa non viene risolta in un oggetto che contiene la risposta. Se non è necessario utilizzare i dati della risposta nel codice, è possibile restituire la promessa direttamente al runtime.

Alcune librerie non restituiscono promesse ma possono essere racchiuse in codice che lo fa. Per ulteriori informazioni, consulta [Definire il gestore di funzioni Lambda in Node.js](#).

## Lambda: esecuzione di una versione o di un alias di una funzione non intenzionale

Problema: versione o alias della funzione non richiamato

Quando si pubblicano nuove funzioni Lambda nella console o utilizzando AWS SAM, la versione più recente del codice è rappresentata da `$LATEST`. Per impostazione predefinita, le invocazioni che non specificano una versione o un alias hanno come target automaticamente la `$LATEST` versione del codice della funzione.

Se utilizzi versioni o alias di funzioni specifici, oltre a queste si tratta di versioni pubblicate immutabili di una funzione. `$LATEST` Quando risolvete i problemi di queste funzioni, verificate innanzitutto che il chiamante abbia richiamato la versione o l'alias desiderato. Puoi farlo controllando i log delle funzioni. La versione della funzione richiamata è sempre mostrata nella riga del registro `START`:

|                               |                                                                         |
|-------------------------------|-------------------------------------------------------------------------|
| 2020-11-13T09:53:21.179-05:00 | START RequestId: a37400cb-f3bc-441b-8592-dcb9fa552995 Version: 1        |
| 2020-11-13T08:14:27.834-05:00 | START RequestId: f4943cb9-58a1-472a-af81-5801b6f0eb8d Version: \$LATEST |

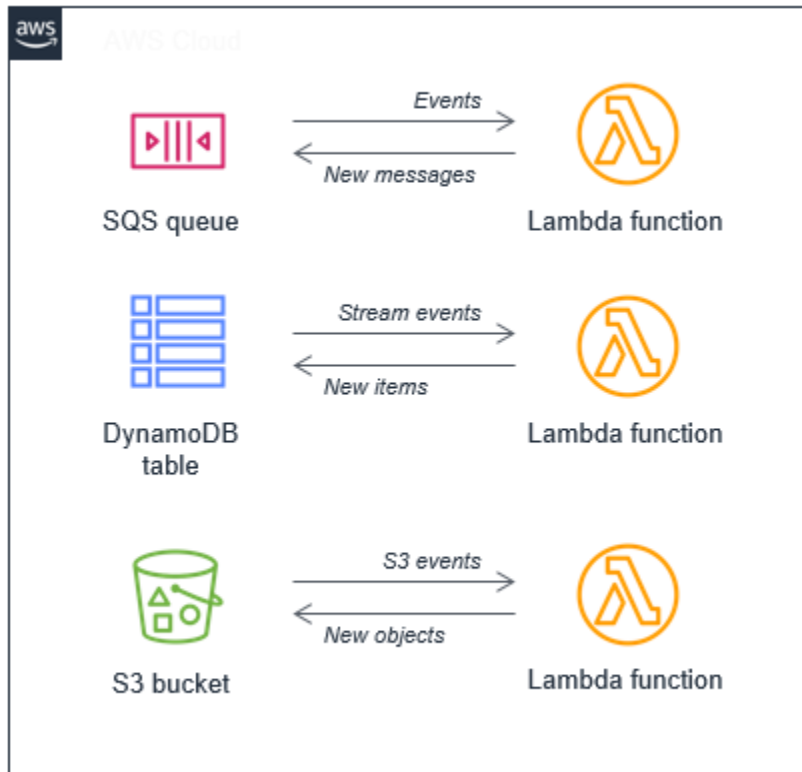
## Lambda: rilevamento di loop infiniti

Problema: modelli di loop infiniti relativi alle funzioni Lambda

Esistono due tipi di cicli infiniti nelle funzioni Lambda. Il primo è all'interno della funzione stessa, causato da un ciclo che non esce mai. L'invocazione termina solo quando la funzione scade. È possibile identificarli monitorando i timeout e quindi correggendo il comportamento di loop.

Il secondo tipo di loop è tra le funzioni Lambda e altre risorse. AWS Si verificano quando un evento proveniente da una risorsa come un bucket S3 richiama una funzione Lambda, che quindi interagisce

con la stessa risorsa di origine per attivare un altro evento. Questo richiama nuovamente la funzione, che crea un'altra interazione con lo stesso bucket S3 e così via. Questi tipi di loop possono essere causati da diverse fonti di AWS eventi, tra cui code Amazon SQS e tabelle DynamoDB. Puoi utilizzare il rilevamento [ricorsivo dei loop per identificare questi pattern](#).



Puoi evitare questi cicli assicurandoti che le funzioni Lambda scrivano su risorse diverse dalla risorsa che la consuma. Se devi pubblicare nuovamente i dati sulla risorsa che consuma, assicurati che i nuovi dati non attivino lo stesso evento. In alternativa, utilizza il filtro [degli eventi](#). Ad esempio, ecco due soluzioni proposte per i loop infiniti con risorse S3 e DynamoDB:

- Se rispondi allo stesso bucket S3, utilizza un prefisso o suffisso diverso da quello del trigger dell'evento.
- Se scrivi elementi nella stessa tabella DynamoDB, includi un attributo in base al quale una funzione Lambda che utilizza può filtrare. Se Lambda trova l'attributo, non genererà un'altra chiamata.

**Generale: indisponibilità del servizio downstream**

**Problema:** i servizi downstream su cui si basa la funzione Lambda non sono disponibili

Per le funzioni Lambda che richiamano endpoint di terze parti o altre risorse a valle, assicurati che siano in grado di gestire gli errori e i timeout del servizio. Queste risorse a valle possono avere tempi di risposta variabili o diventare non disponibili a causa di interruzioni del servizio. A seconda dell'implementazione, questi errori a valle potrebbero apparire come timeout o eccezioni di Lambda se la risposta all'errore del servizio non viene gestita all'interno del codice funzione.

Ogni volta che una funzione dipende da un servizio a valle, ad esempio una chiamata API, implementa una logica appropriata di gestione degli errori e riprova. Per i servizi critici, la funzione Lambda dovrebbe pubblicare metriche o registri su CloudWatch. Ad esempio, se un'API di pagamento di terze parti non è disponibile, la funzione Lambda può registrare queste informazioni. Puoi quindi impostare CloudWatch allarmi per inviare notifiche relative a questi errori.

Poiché Lambda è scalabile rapidamente, i servizi downstream non serverless potrebbero avere difficoltà a gestire i picchi di traffico. Esistono tre approcci comuni per gestire questo problema:

- **Memorizzazione nella cache:** valuta la possibilità di memorizzare nella cache il risultato dei valori restituiti da servizi di terze parti se non cambiano frequentemente. Puoi memorizzare questi valori nella variabile globale della tua funzione o di un altro servizio. Ad esempio, i risultati di una query sull'elenco di prodotti da un'istanza Amazon RDS potrebbero essere salvati per un periodo di tempo all'interno della funzione per evitare query ridondanti.
- **Accodamento:** durante il salvataggio o l'aggiornamento dei dati, aggiungi una coda Amazon SQS tra la funzione Lambda e la risorsa. La coda mantiene i dati in modo duraturo mentre il servizio a valle elabora i messaggi.
- **Proxy:** laddove in genere vengono utilizzate connessioni di lunga durata, ad esempio per le istanze Amazon RDS, utilizza un livello proxy per raggruppare e riutilizzare tali connessioni. Per i database relazionali, [Amazon RDS Proxy](#) è un servizio progettato per contribuire a migliorare la scalabilità e la resilienza delle applicazioni. Lambda-based

## AWS SDK: versioni e aggiornamenti

Problema: l'AWS SDK incluso nel runtime non è la versione più recente

Problema: l'AWS SDK incluso nel runtime si aggiorna automaticamente

I runtime per i linguaggi interpretati includono una versione dell'AWS SDK. Lambda aggiorna periodicamente questi runtime per utilizzare la versione SDK più recente. Per trovare la versione dell'SDK inclusa nel tuo runtime, consulta le seguenti sezioni:

- [Versioni SDK incluse nel runtime \(\) Node.js](#)

- [Versioni SDK incluse nel runtime \(Python\)](#)
- [Versioni SDK incluse nel runtime \(Ruby\)](#)

Per utilizzare una versione più recente dell' AWS SDK o per bloccare le funzioni su una versione specifica, puoi raggruppare la libreria con il tuo codice funzione o [creare un layer Lambda](#). Per informazioni dettagliate sulla creazione di un pacchetto di distribuzione con dipendenze, vedere i seguenti argomenti:

## Node.js

[Distribuisce funzioni Node.js Lambda con archivi di file .zip](#)

## Python

[Utilizzo di archivi di file .zip per le funzioni Lambda in Python](#)

## Ruby

[Distribuire le funzioni Ruby Lambda con gli archivi di file .zip](#)

## Java

[Distribuisce funzioni Lambda per Java con archivi di file .zip o JAR](#)

## Go

[Distribuisce funzioni Lambda per Go con gli archivi di file .zip](#)

## C#

[Crea e implementa le funzioni Lambda C# con gli archivi di file .zip](#)

## PowerShell

[Distribuzione delle funzioni Lambda di PowerShell con gli archivi di file .zip](#)

## Python: caricamento librerie errato

Problema: (Python) alcune librerie non vengono caricate correttamente dal pacchetto di distribuzione

Le librerie con moduli di estensione scritti in C o C ++ devono essere compilate in un ambiente con la stessa architettura del processore di Lambda (Amazon Linux). Per ulteriori informazioni, consulta [Utilizzo di archivi di file .zip per le funzioni Lambda in Python](#).

## Java: la funzione impiega più tempo per elaborare gli eventi dopo l'aggiornamento a Java 17 da Java 11

**Problema:** (Java) la funzione impiega più tempo per elaborare gli eventi dopo l'aggiornamento a Java 17 da Java 11

Ottimizza il compilatore utilizzando il parametro `JAVA_TOOL_OPTIONS`. I runtime Lambda per Java 17 e versioni successive modificano le opzioni predefinite del compilatore. La modifica migliora i tempi di avvio a freddo per le funzioni di breve durata, ma il comportamento precedente è più adatto alle funzioni ad alta intensità di calcolo e di lunga durata. Imposta `JAVA_TOOL_OPTIONS` su `-XX:-TieredCompilation` per ripristinare il comportamento di Java 11. Per ulteriori informazioni sul parametro `JAVA_TOOL_OPTIONS`, vedi [the section called “Comprendere la variabile di ambiente JAVA\\_TOOL\\_OPTIONS”](#).

## Kafka: problemi relativi alla gestione degli errori e ai nuovi tentativi di configurazione

**Problema:** la mappatura dell'origine degli eventi di Kafka non consente di configurare le impostazioni relative ai nuovi tentativi o le destinazioni in caso di errore

Le configurazioni dei nuovi tentativi di Kafka e le destinazioni in caso di errore sono disponibili solo per le mappature delle sorgenti di eventi con la modalità provisioned abilitata. Assicurati di aver effettuato la configurazione prima di tentare di impostare le configurazioni dei nuovi `MinimumPollers` tentativi. `ProvisionedPollerConfig`

Errori di configurazione comuni:

- Tentativi infiniti con bisect batch: non è possibile abilitare `BisectBatchOnFunctionError` quando `MaximumRetryAttempts` è impostato su `-1` (infinito). Imposta un limite di tentativi finito o disabilita il batch bisect.
- Ricorsione dello stesso argomento: l'argomento di destinazione di Kafka in caso di errore non può essere uguale a nessuno degli argomenti di origine. Scegli un nome diverso per l'argomento che hai inserito nella lettera morta.
- Formato di destinazione Kafka non valido: utilizza il `kafka://<topic-name>` formato quando specifichi un argomento Kafka come destinazione in caso di errore.
- kafka: problemi di WriteData autorizzazione: assicurati che il tuo ruolo di esecuzione disponga delle autorizzazioni per l'argomento di destinazione. `kafka-cluster:WriteData` L'argomento non esiste, le eccezioni di timeout o i problemi di limitazione delle API di scrittura potrebbero richiedere l'aumento dei limiti dell'account.

# Risolvi i problemi di mappatura delle sorgenti degli eventi in Lambda

I problemi in Lambda relativi alla mappatura [dell'origine di un evento](#) possono essere più complessi perché implicano il debug su più servizi. Inoltre, il comportamento della fonte dell'evento può differire in base all'origine esatta dell'evento utilizzata. Questa sezione elenca i problemi più comuni che riguardano le mappature delle sorgenti degli eventi e fornisce indicazioni su come identificarli e risolverli.

## Note

Questa sezione utilizza una fonte di eventi Amazon SQS a scopo illustrativo, ma i principi si applicano ad altre mappature di sorgenti di eventi che mettono in coda i messaggi per le funzioni Lambda.

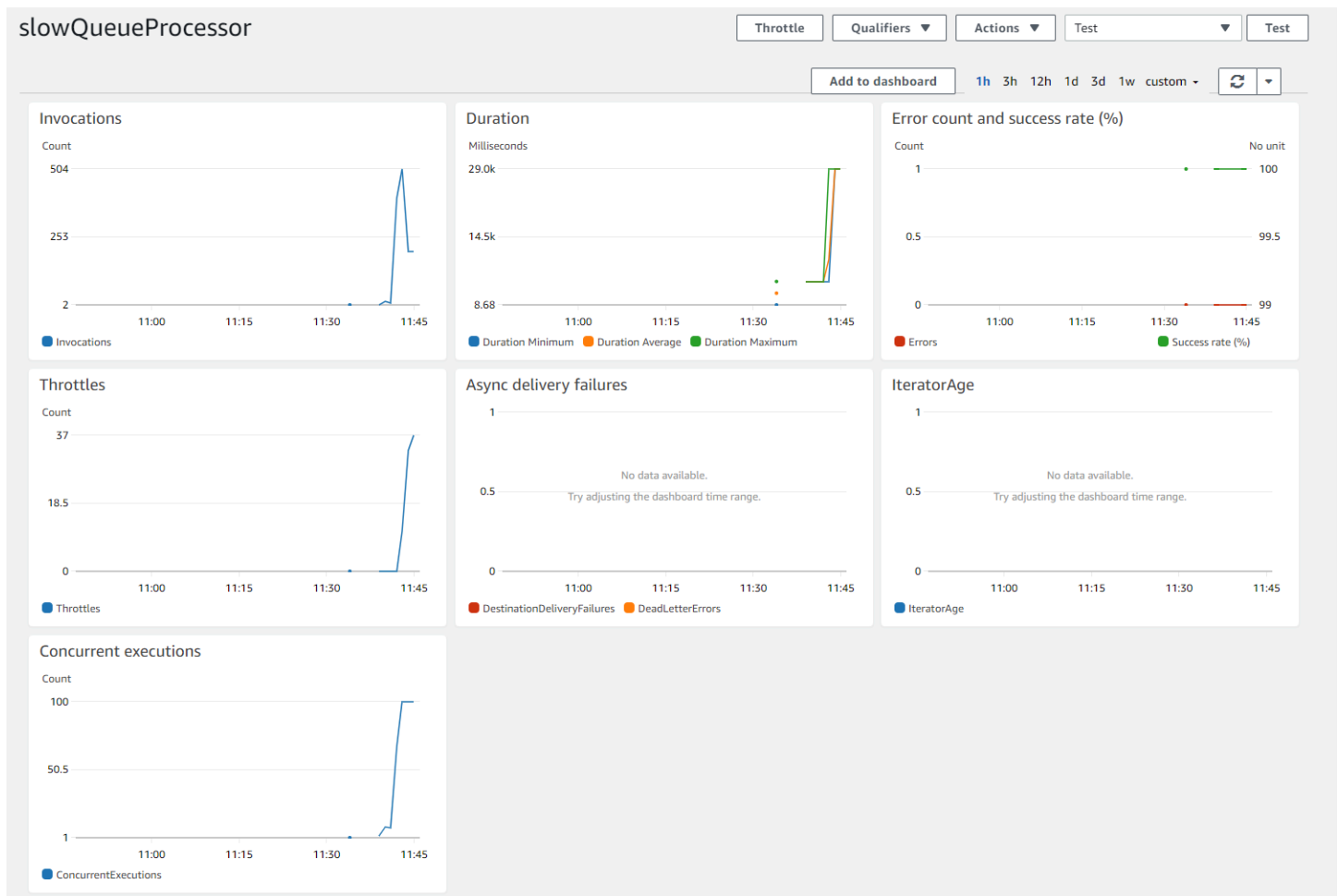
## Identificazione e gestione della limitazione

In Lambda, la limitazione si verifica quando si raggiunge il limite di concorrenza della funzione o dell'account. Considera l'esempio seguente, in cui è presente una funzione Lambda che legge i messaggi da una coda Amazon SQS. Questa funzione Lambda simula invocazioni di 30 secondi e ha una dimensione del batch di 1. Ciò significa che la funzione elabora solo 1 messaggio ogni 30 secondi:

```
const doWork = (ms) => new Promise(resolve => setTimeout(resolve, ms))

exports.handler = async (event) => {
  await doWork(30000)
}
```

Con un tempo di chiamata così lungo, i messaggi iniziano ad arrivare in coda più rapidamente di quanto vengano elaborati. Se la concorrenza non riservata del tuo account è pari a 100, Lambda è in grado di scalare fino a 100 esecuzioni simultanee, dopodiché si verifica una limitazione. Puoi vedere questo schema nelle metriche della funzione: CloudWatch



CloudWatch le metriche relative alla funzione non mostrano errori, ma il grafico delle esecuzioni simultanee mostra che è stata raggiunta la concorrenza massima di 100. Di conseguenza, il grafico Throttles mostra la limitazione in atto.

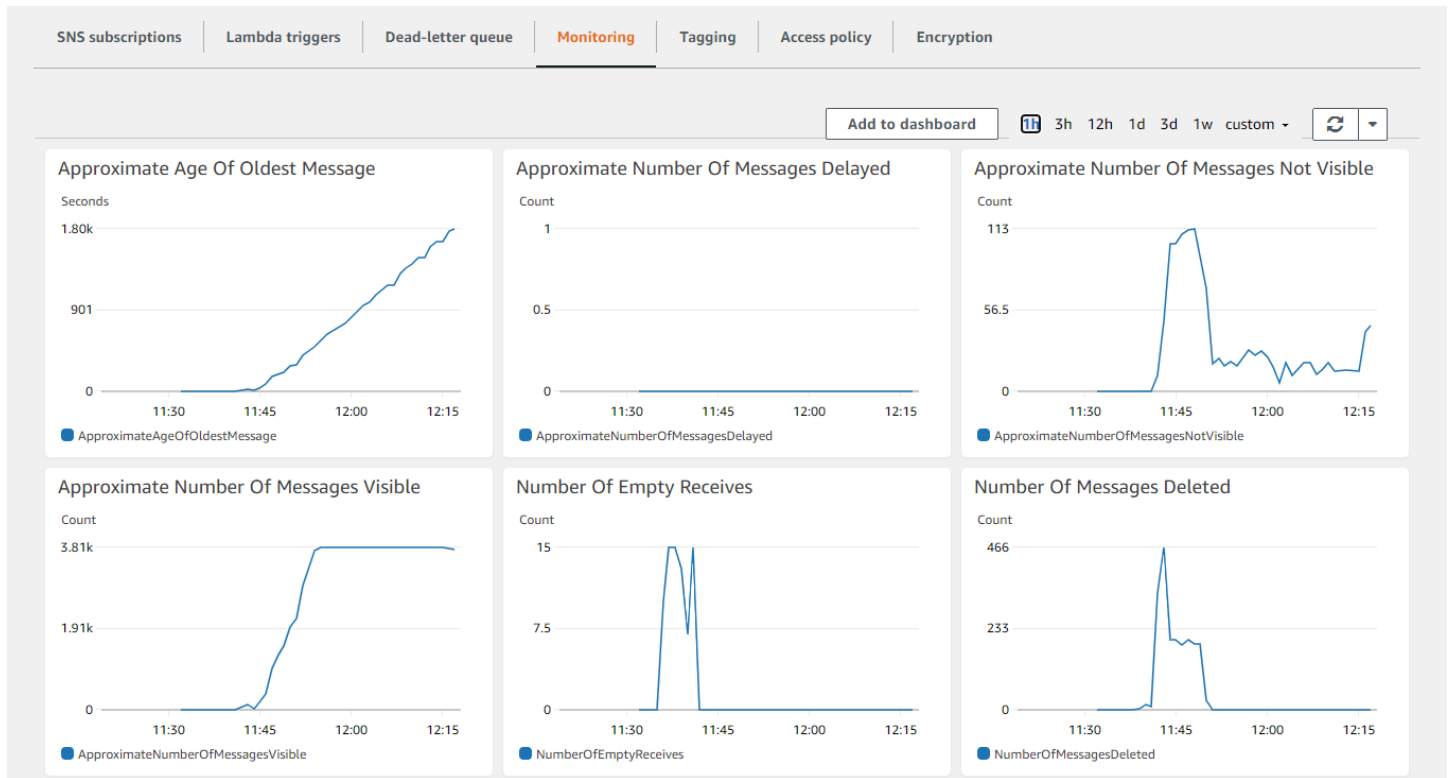
È possibile rilevare la limitazione tramite CloudWatch allarmi e impostare una sveglia ogni volta che la metrica di limitazione per una funzione è maggiore di 0. Dopo aver identificato il problema della limitazione, sono disponibili alcune opzioni per la risoluzione:

- Richiedi un aumento della concorrenza all' AWS assistenza in questa regione.
- Identificare i problemi di prestazioni della funzione per migliorare la velocità di elaborazione e di conseguenza il throughput.
- Aumentate la dimensione del batch della funzione, in modo che a ogni chiamata vengano elaborati più messaggi.

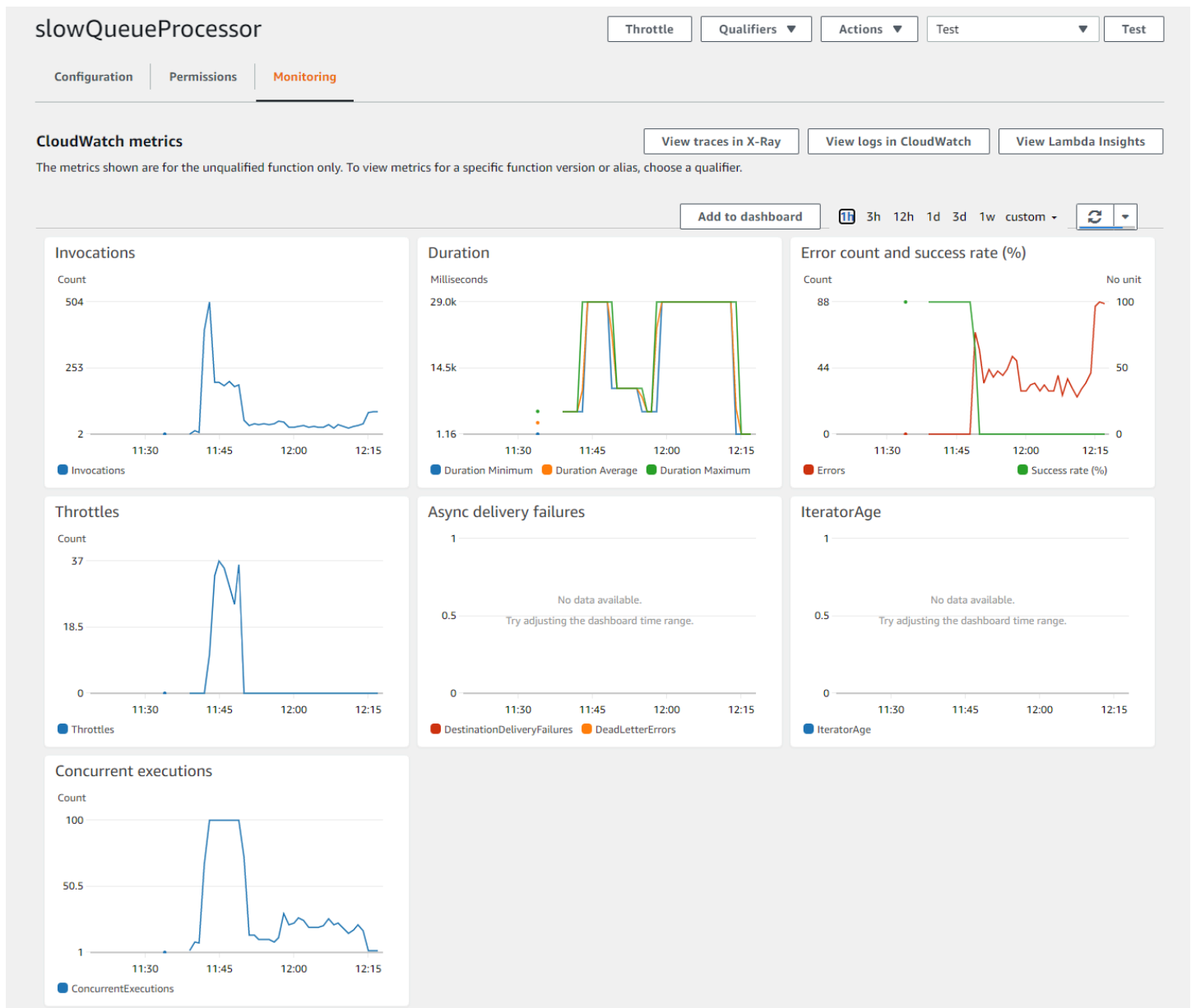


## Errori nella funzione di elaborazione

Se la funzione di elaborazione genera errori, Lambda restituisce i messaggi alla coda SQS. Lambda impedisce il ridimensionamento della funzione per prevenire errori su larga scala. Le seguenti metriche SQS CloudWatch indicano un problema con l'elaborazione delle code:

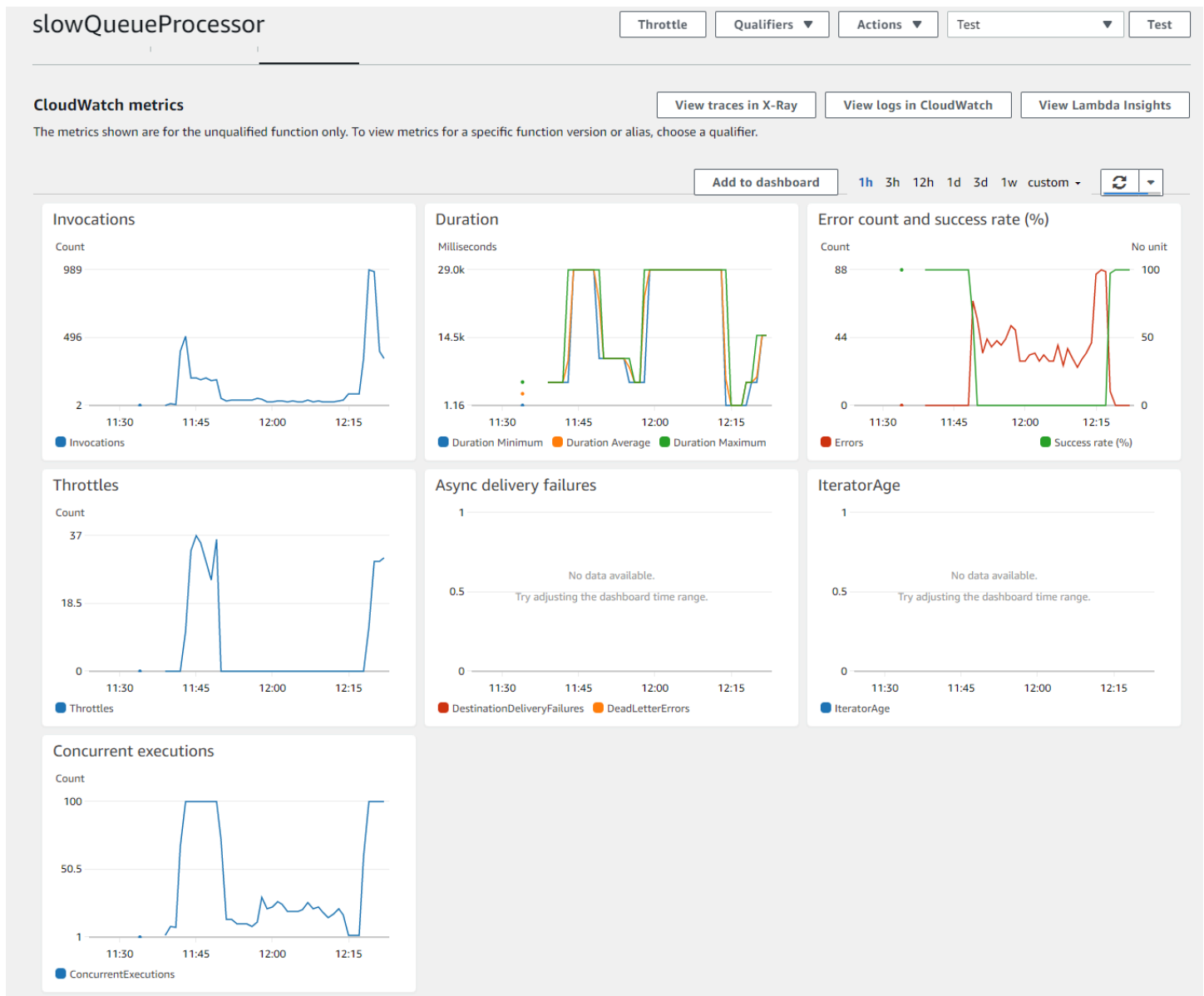


In particolare, aumentano sia l'età del messaggio più vecchio che il numero di messaggi visibili, mentre nessun messaggio viene eliminato. La coda continua a crescere ma i messaggi non vengono elaborati. Le CloudWatch metriche per la funzione Lambda di elaborazione indicano anche che c'è un problema:



Il parametro numero errori è diverso da zero ed è in aumento, mentre le esecuzioni simultanee si sono ridotte e la limitazione è stata interrotta. Ciò dimostra che Lambda ha smesso di aumentare la scalabilità della funzione a causa di errori. I CloudWatch log della funzione forniscono dettagli sul tipo di errore.

È possibile risolvere questo problema identificando la funzione che causa l'errore, quindi trovando e risolvendo l'errore. Dopo aver corretto l'errore e distribuito il nuovo codice funzione, le CloudWatch metriche dovrebbero mostrare il ripristino dell'elaborazione:



Qui, la metrica del conteggio degli errori scende a zero e la metrica del tasso di successo torna al 100%. Lambda inizia nuovamente a scalare la funzione, come mostrato nel grafico Esecuzioni simultanee.

## Identificazione e gestione della contropressione

Se un produttore di eventi genera costantemente messaggi per una coda SQS più velocemente di quanto una funzione Lambda possa elaborarli, si verifica una contropressione. In questo caso, il monitoraggio SQS dovrebbe mostrare che l'età del messaggio più vecchio cresce linearmente, insieme al numero approssimativo di messaggi visibili. È possibile rilevare la contropressione nelle code utilizzando gli allarmi. CloudWatch

I passaggi per risolvere la contropressione dipendono dal carico di lavoro. Se l'obiettivo principale è aumentare la capacità di elaborazione e il throughput tramite la funzione Lambda, hai alcune opzioni:

- Richiedi all'assistenza un aumento della concorrenza nella regione specifica. AWS
- Aumentate la dimensione del batch della funzione, in modo che a ogni chiamata vengano elaborati più messaggi.

## Risoluzione dei problemi di rete in Lambda

Per impostazione predefinita, Lambda esegue le funzioni in un Virtual Private Cloud (VPC) interno con connettività ai servizi AWS e a Internet. Per accedere alle risorse di rete locali, è possibile [configurare la funzione in modo che si connetta a un VPC nel proprio account](#). Quando si utilizza questa funzionalità, è possibile gestire l'accesso a Internet della funzione e la connettività di rete con risorse Amazon Virtual Private Cloud (Amazon VPC).

Gli errori di connettività di rete possono derivare da problemi relativi alla configurazione del routing del VPC, alle regole del gruppo di sicurezza, alle autorizzazioni dei ruoli AWS Identity and Access Management (IAM) o alla traduzione degli indirizzi di rete (NAT) del tuo VPC o dalla disponibilità di risorse come indirizzi IP o interfacce di rete. A seconda del problema, potrebbe verificarsi un errore o un timeout specifico se una richiesta non riesce a raggiungere la sua destinazione.

### Argomenti

- [VPC: la funzione perde l'accesso a Internet o scade](#)
- [VPC: la connessione TCP o UDP fallisce a intermittenza](#)
- [VPC: è necessario accedere alla funzione Servizi AWS senza usare internet](#)
- [VPC: raggiunto il limite dell'interfaccia di rete elastica](#)
- [EC2: interfaccia di rete elastica con tipo "lambda"](#)
- [DNS: impossibile connettersi agli host con UNKNOWNHOSTEXCEPTION](#)

### VPC: la funzione perde l'accesso a Internet o scade

Problema: la funzione Lambda perde l'accesso a Internet dopo la connessione a un VPC.

Errore: Errore: collegare ETIMEDUT 176.32.98.189:443

Errore: Errore: timeout dell'attività dopo 10,00 secondi

Errore: tempo di lettura scaduto ReadTimeoutError. (timeout di lettura = 15)

Quando si collega una funzione a un VPC, tutte le richieste in uscita passano attraverso il VPC. Per connettersi a Internet, configurare il VPC per inviare il traffico in uscita dalla sottorete della funzione a un gateway NAT in una sottorete pubblica. Per ulteriori informazioni e le configurazioni VPC di esempio, vedere [the section called “Accesso Internet per funzioni del VPC”](#).

Se alcune delle tue connessioni TCP stanno scadendo, verifica [the section called “VPC: la connessione TCP o UDP fallisce a intermittenza”](#) se la tua sottorete utilizza una lista di controllo degli accessi alla rete (NACL). Altrimenti, è probabile che ciò sia dovuto alla frammentazione dei pacchetti. Le funzioni Lambda non sono in grado di gestire le richieste TCP frammentate in entrata, poiché Lambda non supporta la frammentazione IP per TCP o ICMP.

## VPC: la connessione TCP o UDP fallisce a intermittenza

### Note

Questo problema si verifica solo se la sottorete utilizza un elenco di controllo degli accessi alla [rete \(ACL\)](#). Gli ACL di rete non sono necessari per consentire a Lambda di connettersi alle sottoreti.

Problema: Lambda perde in modo intermittente la connessione alle sottoreti VPC, per le quali hai configurato un elenco di controllo degli accessi alla rete (ACL).

Per le funzioni VPC-enabled Lambda, AWS crea [ENI hyperplane](#) nell'account del cliente e utilizza porte temporanee 1024 per connettere Lambda al 65535 VPC del cliente. Se utilizzi gli ACL di rete nella sottorete di destinazione, devi consentire l'intervallo di porte sia per TCP che per UDP. 1024 65535 Se non si consente questo intervallo completo di porte, si possono verificare errori di connessione intermittenti.

## VPC: è necessario accedere alla funzione Servizi AWS senza usare internet

Problema: è necessario accedere alla funzione Lambda Servizi AWS senza utilizzare Internet.

Per connettere una funzione Servizi AWS da una sottorete privata senza accesso a Internet, utilizza gli endpoint VPC.

## VPC: raggiunto il limite dell'interfaccia di rete elastica

Errore `ENILimitReachedException`: è stato raggiunto il limite dell'interfaccia di rete elastica per il VPC della funzione.

Quando si connette una funzione Lambda a un VPC, Lambda crea un'interfaccia di rete elastica per ogni combinazione di sottorete e gruppo di sicurezza collegato alla funzione. La quota di servizio predefinita è 250 interfacce di rete per VPC. Per richiedere un aumento di una quota, è possibile utilizzare la [console Service Quotas](#).

## EC2: interfaccia di rete elastica con tipo "lambda"

Codice di errore: `Client.OperationNotPermitted`

Messaggio di errore: il gruppo di sicurezza non può essere modificato per questo tipo di interfaccia

Si riceve questo errore se si tenta di modificare un'interfaccia di rete elastica (ENI) gestita da Lambda. `ModifyNetworkInterfaceAttribute` non è incluso nell'API Lambda per le operazioni di aggiornamento su interfacce di rete elastiche create da Lambda.

## DNS: impossibile connettersi agli host con `UNKNOWNHOSTEXCEPTION`

Messaggio di errore: `UNKNOWNHOSTEXCEPTION`

Le funzioni Lambda supportano un massimo di 20 connessioni TCP simultanee per la risoluzione DNS. La tua funzione potrebbe esaurire tale limite. Le richieste DNS più comuni vengono eseguite tramite UDP. Se la tua funzione consiste solo nel creare connessioni DNS UDP, è poco probabile che questo sia il tuo problema. Questo errore viene generalmente generato a causa di una configurazione errata o di un'infrastruttura degradata, quindi prima di esaminare a fondo il traffico DNS, verifica che l'infrastruttura DNS sia configurata correttamente e integra e che la funzione Lambda si riferisca a un host specificato nel DNS.

Se diagnostichi il problema come correlato al numero massimo di connessioni TCP, tieni presente che non puoi richiedere un aumento di questo limite. Se la tua funzione Lambda ritorna al DNS TCP a causa di payload DNS di grandi dimensioni, verifica che la tua soluzione utilizzi librerie che supportano EDNS. Per ulteriori informazioni su EDNS, consulta [lo standard RFC 6891](#). Se i payload DNS superano costantemente le dimensioni massime EDNS, la soluzione potrebbe comunque esaurire il limite DNS TCP.

# Le migliori pratiche per lavorare con AWS Lambda funzioni

Di seguito sono indicate le best practice per l'utilizzo di AWS Lambda.

## Argomenti

- [Codice della funzione](#)
- [Configurazione delle funzioni](#)
- [Scalabilità delle funzioni](#)
- [Parametri e allarmi](#)
- [Utilizzo di flussi](#)
- [Best practice di sicurezza](#)

## Codice della funzione

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedi [Riutilizzo delle connessioni con Keep-Alive in Node.js](#).

Utilizzare [le variabili di ambiente](#) per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare

un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su 0 per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionali alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha una dipendenza su queste API non pubbliche. Consulta il [riferimento all'API](#) per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta [Come posso rendere idempotente la mia funzione Lambda?](#).

#### Note

Puoi usare Powertools per rendere le funzioni idempotenti. AWS Lambda Per ulteriori informazioni, consulta:

- [Python - Utilità di idempotenza](#)
- [TypeScript - Utilità per l'idempotenza](#)
- [Java - Utilità di idempotenza](#)
- [.NET - Utilità di idempotenza](#)

Per le best practice relative al codice specifico di un linguaggio, consulta le seguenti sezioni:

- [Procedure consigliate per il codice per le funzioni Lambda Node.js](#)
- [Procedure consigliate per il codice per le funzioni Lambda TypeScript](#)
- [Best practice di codice per le funzioni Lambda con Python Lambda](#)
- [Best practice di codice per le funzioni Lambda con Ruby](#)
- [Best practice di codice per funzioni Lambda in Java](#)
- [Procedure consigliate di codice per le funzioni Go Lambda](#)
- [Best practice di codice per le funzioni Lambda con C#](#)
- [Best practice di codice per funzioni Lambda in Rust](#)



## Configurazione delle funzioni

La verifica delle prestazioni della funzione Lambda è una fase fondamentale per garantire la scelta della configurazione delle dimensioni di memoria ottimali. Ogni aumento delle dimensioni di memoria determina un aumento equivalente della CPU disponibile per la funzione. L'utilizzo della memoria per la funzione viene determinato per richiamo e può essere visualizzato in Amazon. [CloudWatch](#) Ad ogni richiamo viene REPORT : inserita una voce, come mostrato di seguito:

```
REPORT RequestId: 3604209a-e9a3-11e6-939a-754dd98c7be3 Duration: 12.34 ms Billed
Duration: 100 ms Memory Size: 128 MB Max Memory Used: 18 MB
```

Se si analizza il campo `Max Memory Used` :, è possibile determinare se per la funzione è necessaria una maggiore quantità di memoria o se ne è stata riservata una quantità eccessiva.

Per trovare la giusta configurazione di memoria per le tue funzioni, ti consigliamo di utilizzare il progetto open source AWS Lambda Power Tuning. Per ulteriori informazioni, consulta [AWS Lambda Power Tuning on GitHub](#).

Per ottimizzare le prestazioni delle funzioni, consigliamo inoltre di distribuire librerie che possono utilizzare Advanced Vector Extensions 2 (AVX2). In questo modo, è possibile elaborare carichi di lavoro impegnativi, tra cui inferenza di machine learning, elaborazione multimediale, calcolo ad alte prestazioni (HPC), simulazioni scientifiche e modellazione finanziaria. Per ulteriori informazioni, consultate [Creazione di funzioni più veloci AWS Lambda con AVX2](#).

Eseguire il test della funzione Lambda per determinare un valore di timeout ottimale. È importante analizzare la durata di esecuzione della funzione in modo da poter determinare meglio eventuali problemi con un servizio di dipendenza che potrebbero aumentare la concorrenza della funzione oltre quanto previsto. Ciò è particolarmente importante quando la funzione Lambda effettua chiamate di rete a risorse che potrebbero non gestire la scalabilità di Lambda. Per ulteriori informazioni sul test di carico dell'applicazione, consulta [Test del carico distribuito su AWS](#).

Utilizzare le autorizzazioni più restrittive per le impostazioni delle policy IAM. È importante comprendere le risorse e le operazioni necessarie alla funzione Lambda e limitare il ruolo di esecuzione in base a tali autorizzazioni. Per ulteriori informazioni, consulta [Gestione delle autorizzazioni in AWS Lambda](#).

Acquisire familiarità con [Quote di Lambda](#). Le dimensioni del payload, i descrittori di file e lo spazio / tmp sono aspetti spesso sottovalutati quando si determinano i limiti delle risorse del runtime.

Eliminare le funzioni Lambda non più utilizzate. In questo modo le funzioni non utilizzate non vengono conteggiate inutilmente ai fini del calcolo del limite delle dimensioni del pacchetto di distribuzione.

Se si sta usando Amazon Simple Queue Service come origine eventi, verificare che il valore del tempo di invocazione stimato della funzione non superi il valore [Visibility Timeout \(Timeout visibilità\)](#) della coda. Questo si applica a [CreateFunction](#) e [UpdateFunctionConfiguration](#).

- In caso di `CreateFunction`, AWS Lambda fallisce il processo di creazione della funzione.
- In caso di `UpdateFunctionConfiguration`, potrebbe causare invocazioni duplicate della funzione.

## Scalabilità delle funzioni

Informati sui vincoli di throughput a monte e a valle. Sebbene le funzioni Lambda si adattino perfettamente in base al carico, le dipendenze upstream e downstream potrebbero non avere le stesse capacità di throughput. Se devi limitare il valore massimo di scalabilità della tua funzione, puoi [configurare la simultaneità riservata](#) sulla funzione.

Limitazione all'acceleratore incorporata. Se la tua funzione sincrona subisce una limitazione a causa del traffico che supera la velocità di scalabilità di Lambda, puoi utilizzare le seguenti strategie per migliorare la tolleranza alla limitazione:

- Utilizza [timeout, nuovi tentativi e backoff con jitter](#). L'implementazione di queste strategie semplifica le chiamate ripetute e aiuta a garantire che Lambda possa scalare in pochi secondi per ridurre al minimo le limitazioni da parte dell'utente finale.
- Utilizza la [simultaneità fornita](#). La simultaneità fornita è il numero di ambienti di esecuzione pre-inizializzati che Lambda assegna alla tua funzione. Lambda gestisce le richieste in arrivo utilizzando la simultaneità fornita, se disponibile. Lambda può anche scalare la funzione oltre l'impostazione di simultaneità fornita, se necessario. La configurazione della concorrenza predisposta comporta costi aggiuntivi per l'account. AWS

## Parametri e allarmi

Usa [Utilizzo delle CloudWatch metriche con Lambda](#) and [CloudWatch Alarms](#) invece di creare o aggiornare una metrica dall'interno del codice della funzione Lambda. In questo modo è molto più efficiente eseguire il controllo dello stato delle funzioni Lambda, perché è possibile rilevare tempestivamente i problemi nel processo di sviluppo. Ad esempio, puoi configurare un allarme

in base alla durata prevista dell'invocazione della funzione Lambda per risolvere eventuali colli di bottiglia o latenze attribuibili al codice della funzione.

Emetti metriche personalizzate in modo asincrono utilizzando Embedded Metric Format (EMF). Invece di effettuare chiamate API sincrone a, usa EMF per emettere metriche attraverso i log della CloudWatch tua funzione. Questo approccio riduce la latenza e migliora le prestazioni.

L'utilità Metrics in Powertools for AWS Lambda gestisce automaticamente la formattazione EMF. Per ulteriori informazioni, consultate le utilità [Python](#), [TypeScript](#), Java o [.NET](#) Metrics nella documentazione di Powertools. AWS Lambda Per informazioni sull'utilizzo di EMF per generare log in formato metrico, consulta [Publishing logs with the embedded metric format nella Amazon User Guide](#). CloudWatch

Usa la registrazione JSON strutturata per una migliore osservabilità. La registrazione strutturata semplifica la ricerca, il filtraggio e l'analisi dei log delle funzioni. Prendi in considerazione l'utilizzo dell'utilità Logger di Powertools per formattare automaticamente i log AWS Lambda in JSON. Per ulteriori informazioni, consulta le utilità [Python](#), [TypeScript](#), Java o [.NET](#) Logger nella documentazione di Powertools. AWS Lambda

Usa la tua libreria di registrazione e [AWS Lambda Metrics and Dimensions](#) per rilevare gli errori dell'app (come ERR, ERROR e WARNING).

Utilizza [AWS Cost Anomaly Detection](#) per rilevare attività insolite sul tuo account. Cost Anomaly Detection utilizza il machine learning per monitorare continuamente i costi e l'utilizzo riducendo al minimo i falsi positivi. Cost Anomaly Detection utilizza i dati provenienti da AWS Cost Explorer, con un ritardo fino a 24 ore.

Di conseguenza, possono essere necessarie fino a 24 ore per rilevare un'anomalia dopo l'utilizzo. Per iniziare a utilizzare Cost Anomaly Detection, è necessario [registrarsi per Cost Explorer](#). Quindi, [accedi a Cost Anomaly Detection](#).

## Utilizzo di flussi

Eseguire il test con dimensioni di batch e record diverse in modo che la frequenza di polling di ogni origine eventi sia regolata in base alla velocità con cui la funzione è in grado di completare l'attività. Il [CreateEventSourceMapping](#) BatchSize parametro controlla il numero massimo di record che possono essere inviati alla funzione con ogni richiamo. Una dimensione di batch maggiore può spesso assorbire in modo più efficiente il costo associato all'invocazione in un set di record più grande, aumentando in tal modo il throughput.

Per impostazione predefinita, Lambda richiama la funzione non appena i record sono disponibili. Se il batch che Lambda legge dall'origine eventi contiene un solo record, Lambda invia solo un record alla funzione. Per evitare di richiamare la funzione con pochi record è possibile, configurando un periodo di batch, chiedere all'origine eventi di memorizzare nel buffer i registri per un massimo di 5 minuti. Prima di richiamare la funzione, Lambda continua a leggere i registri dall'origine eventi fino a quando non ha raccolto un batch completo, fino alla scadenza del periodo di batch o fino a quando il batch non ha raggiunto il limite del payload di 6 MB. Per ulteriori informazioni, consulta [Comportamento di batching](#).

#### Warning

Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per saperne di più, vedi [Come posso rendere idempotente](#) la mia funzione Lambda nel Knowledge Center. AWS

Abilita la risposta batch parziale per l'elaborazione dei flussi. Durante l'elaborazione di batch di record da flussi come Kinesis o DynamoDB Streams, abilita la risposta batch parziale per consentire a Lambda di riprovare solo i record non riusciti anziché l'intero batch. Ciò migliora l'efficienza di elaborazione e riduce le rielaborazioni non necessarie. Facoltativamente, è possibile utilizzare l'utilità Batch di Powertools per AWS Lambda semplificare i modelli di elaborazione in batch.

#### Note

È possibile utilizzare Powertools for AWS Lambda per l'elaborazione in batch. Per ulteriori informazioni, consulta:

- [Python - Elaborazione in batch](#)
- [TypeScript - Elaborazione in batch](#)
- [Java - Elaborazione in batch](#)
- [.NET - Elaborazione in batch](#)

Aumentare il throughput di elaborazione del flusso Kinesis tramite l'aggiunta di shard. Un flusso Kinesis è costituito da uno o più shard. La velocità con cui Lambda è in grado di leggere i dati da Kinesis si dimensiona linearmente con il numero di shard.

L'aumento del numero di shard aumenta direttamente il numero massimo di chiamate simultanee di funzioni Lambda e può aumentare la velocità di elaborazione dello stream processing di Kinesis. Per ulteriori informazioni sulle relazioni tra shard e invocazioni di funzioni, consulta [the section called “Flussi di polling e batching”](#). Se si aumenta il numero di shard in un flusso Kinesis, verificare di avere scelto una chiave di partizione opportuna (consultare l'argomento relativo alle [chiavi di partizione](#)) per i dati, in modo che i record correlati appartengano agli stessi shard e che i dati siano distribuiti in modo uniforme.

Usa [Amazon CloudWatch](#) on per determinare se il tuo stream IteratorAge Kinesis è in fase di elaborazione. Ad esempio, configura un CloudWatch allarme con un'impostazione massima su 30000 (30 secondi).

## Best practice di sicurezza

Monitora il tuo utilizzo AWS Lambda in relazione alle migliori pratiche di sicurezza utilizzando AWS Security Hub CSPM. AWS Security Hub CSPM utilizza i controlli di sicurezza per valutare le configurazioni delle risorse e gli standard di sicurezza per aiutarti a rispettare vari framework di conformità. Per ulteriori informazioni sull'utilizzo di Security Hub CSPM per valutare le risorse Lambda, consulta i controlli nella Guida per l'utente. [AWS Lambda](#) AWS Security Hub CSPM

Monitora i log delle attività di rete Lambda utilizzando Amazon Lambda Protection. GuardDuty GuardDuty La protezione Lambda ti aiuta a identificare potenziali minacce alla sicurezza quando le funzioni Lambda vengono richiamate nel tuo Account AWS. Ad esempio, se una delle tue funzioni richiede un indirizzo IP associato ad attività relative alle criptovalute, GuardDuty monitora i log delle attività di rete generati quando viene richiamata una funzione Lambda. Per saperne di più, consulta la protezione [Lambda](#) nella Amazon GuardDuty User Guide.

## Applicazioni di esempio Lambda

L' GitHub archivio di questa guida include applicazioni di esempio che dimostrano l'uso di vari linguaggi e AWS servizi. Ogni applicazione di esempio include script per semplificare l'implementazione, la pulizia e il supporto delle risorse.

## Node.js

### Esempi di applicazioni Lambda in Node.js

- [blank-nodejs](#) — Una Node.js funzione che mostra l'uso della registrazione, delle variabili di ambiente, del AWS X-Ray tracciamento, dei livelli, dei test unitari e dell'SDK. AWS
- [nodejs-apig](#) – Una funzione con un endpoint API pubblico che elabora un evento proveniente da API Gateway e restituisce una risposta HTTP.

## Python

### Applicazioni Lambda di esempio in Python

- [blank-python](#) — Una funzione Python che mostra l'uso della registrazione, delle variabili di ambiente, del AWS X-Ray tracciamento, dei livelli, dei test unitari e dell'SDK. AWS

## Ruby

### Applicazioni Lambda di esempio in Ruby

- [blank-ruby](#) — [Una funzione Ruby](#) che mostra l'uso della registrazione, delle variabili di ambiente, del tracciamento, dei livelli, dei test unitari e dell'SDK. AWS X-Ray AWS
- [Esempi di codice Ruby per AWS Lambda](#) — Esempi di codice scritti in Ruby che dimostrano come interagire con Lambda. AWS

## Java

### Applicazioni Lambda di esempio in Java

- [example-java](#): una funzione Java che dimostra come utilizzare Lambda per elaborare gli ordini. Questa funzione illustra come definire e deserializzare un oggetto evento di input personalizzato, utilizzare l'SDK e registrare l'output. AWS
- [java-basic](#): una raccolta di funzioni Java minimali con unit test e configurazione della registrazione dei log delle variabili.
- [java-events](#): una raccolta di funzioni Java che contengono codice skeleton per la gestione degli eventi di vari servizi, ad esempio Gateway Amazon API, Amazon SQS e Amazon Kinesis. Queste funzioni utilizzano la versione più recente della libreria [aws-lambda-java-events](#) (3.0.0 e versioni successive). Questi esempi non richiedono l'SDK come dipendenza. AWS

- [s3-java](#) – Una funzione Java che elabora gli eventi di notifica da Amazon S3 e utilizza la Java Class Library (JCL) per creare anteprime dai file di immagine caricati.
- [layer-java](#): una funzione Java che illustra come utilizzare un livello Lambda per impacchettare dipendenze separate dal codice della funzione principale.

Esecuzione dei framework Java più diffusi su Lambda

- [spring-cloud-function-samples](#) — Un esempio di Spring che mostra come utilizzare il [framework Spring Cloud Function](#) per creare funzioni Lambda. AWS
- [Demo dell'applicazione Spring Boot senza server](#): un esempio che mostra come configurare una tipica applicazione Spring Boot in un runtime Java gestito con e senza SnapStart, o come immagine nativa GraalVM con un runtime personalizzato.
- [Demo dell'applicazione Serverless Micronaut](#): un esempio che mostra come utilizzare Micronaut in un runtime Java gestito con e senza SnapStart, o come immagine nativa GraalVM con un runtime personalizzato. [Scopri Micronaut/Lambda di più nelle guide.](#)
- [Demo dell'applicazione Quarkus senza server](#): un esempio che mostra come utilizzare Quarkus in un runtime Java gestito con e senza SnapStart, o come immagine nativa GraalVM con un runtime personalizzato. [Scopri Quarkus/Lambda di più nella guida e nella guida. Quarkus/SnapStart](#)

## Go

Lambda fornisce le seguenti applicazioni di esempio per il runtime di Go:

Applicazioni Lambda di esempio in Go

- [go-al2](#): una funzione hello world che restituisce l'indirizzo IP pubblico. Questa app utilizza il runtime personalizzato `provided.al2`.
- [blank-go](#): una funzione Go che mostra l'uso delle librerie Go di Lambda, la registrazione, le variabili di ambiente e l'SDK. AWS Questa app utilizza il runtime `go1.x`.

## C#

Applicazioni Lambda di esempio in C#

- [blank-csharp](#) – Una funzione C# che mostra l'uso di librerie .NET di Lambda, logging, variabili di ambiente, tracciamento AWS X-Ray, unit test e SDK AWS.

- [blank-csharp-with-layer](#): funzione C# che utilizza la CLI di .NET per creare un livello che raggruppa le dipendenze della funzione.
- [ec2-spot](#) – Una funzione che gestisce le richieste di istanze spot in Amazon EC2.

## PowerShell

Lambda fornisce le seguenti applicazioni di esempio per: PowerShell

- [blank-powershell](#) — Una PowerShell funzione che mostra l'uso della registrazione, delle variabili di ambiente e dell'SDK. AWS

Per distribuire un'applicazione di esempio, seguire le istruzioni contenute nel file README.

## Strumenti di sviluppo per Lambda

Hai accesso a una varietà di strumenti che aumentano la produttività e la facilità d'uso durante l'intero ciclo di vita dello sviluppo. Questa sezione fornisce informazioni sugli strumenti che aiutano molti clienti Lambda a progettare, sviluppare e gestire le proprie applicazioni. Dallo sviluppo locale nel tuo IDE alla distribuzione e gestione di applicazioni serverless complesse, questi strumenti ti aiutano a semplificare il flusso di lavoro, migliorare la qualità del codice e accelerare lo sviluppo di soluzioni affidabili. Lambda-based

- Sviluppo locale: scrivi e testa le funzioni Lambda più velocemente nel tuo ambiente di sviluppo preferito. Il AWS Toolkit for VS Code consente lo sviluppo, il debug e il test di funzioni locali con funzionalità di distribuzione diretta su Lambda.
- Infrastructure as Code (IaC): implementa e gestisci le applicazioni serverless in modo coerente, dai test locali alla produzione. AWS SAM e CloudFormation consentono di definire e gestire l'infrastruttura serverless tramite codice per implementazioni coerenti e controllate dalla versione. AWS CDK
- GitHub Azioni: automatizza le implementazioni Lambda direttamente dal tuo repository di codice. Con GitHub Actions, puoi configurare flussi di lavoro che implementano automaticamente le tue funzioni Lambda ogni volta che apporti modifiche al codice o alla configurazione, semplificando la pipeline. CI/CD
- Powertools for AWS Lambda: crea applicazioni serverless pronte per la produzione con meno codice personalizzato. Powertools for AWS Lambda (noto anche come Powertools for AWS) è un toolkit per sviluppatori open source che semplifica l'implementazione delle migliori pratiche



serverless come l'osservabilità, il recupero dei parametri e l'idempotenza in Python, Java e .NET. TypeScript

- Flussi di lavoro ed eventi: coordina le funzioni Lambda con servizi, API e sistemi esterni. AWS Lambda offre due opzioni di orchestrazione: [Lambda durable functions](#) per l'orchestrazione incentrata sulle applicazioni utilizzando linguaggi di programmazione standard all'interno di Lambda e per l'orchestrazione incentrata sul flusso di lavoro con progettazione visiva su più servizi. [AWS Step Functions](#) Amazon EventBridge offre funzionalità di gestione degli eventi per architetture basate sugli eventi. Per assistenza nella scelta di un approccio di orchestrazione, [consulta Durable functions o Step Functions](#).

## Strumenti di sviluppo locale

Gli ambienti di sviluppo locali consentono di lavorare offline e utilizzare funzionalità IDE avanzate, iterando rapidamente sulle funzioni Lambda. Questi strumenti consentono di eseguire il debug di funzioni complesse e di svilupparle in ambienti con connettività limitata. Inoltre supportano la collaborazione in team e l'integrazione con i sistemi di controllo delle versioni.

Per ulteriori informazioni sullo sviluppo locale delle funzioni Lambda, consulta [Sviluppo locale di funzioni Lambda](#). Questa pagina descrive come spostare lo sviluppo di funzioni Lambda dalla AWS console a Visual Studio Code, che fornisce un ricco ambiente di sviluppo con funzionalità come il debug e il completamento del codice. Per effettuare la transizione, è necessario configurare le credenziali AWS Toolkit for Visual Studio Code and, dopodiché è possibile utilizzare le funzionalità avanzate di VS Code mantenendo la possibilità di eseguire la distribuzione direttamente su AWS.

Lo sviluppo locale per Lambda offre diverse funzionalità chiave:

- Usa l'integrazione di Visual Studio Code con la console Lambda
- Configura gli ambienti di sviluppo Lambda locali
- Funzioni di debug e test a livello locale
- Applica le migliori pratiche per la gestione delle funzioni locali

Per ulteriori informazioni, consulta [Sviluppo locale di funzioni Lambda](#).

## Strumenti Infrastructure as Code (IaC)

Con gli strumenti Infrastructure as Code (IaC), puoi definire e gestire la tua architettura serverless utilizzando il codice. Questo approccio aiuta a mantenere la coerenza tra gli ambienti, ti dà il controllo

sulle versioni dell'infrastruttura e supporta DevOps le pratiche. L'IaC è particolarmente utile per automatizzare le implementazioni, garantire ambienti coerenti e gestire implementazioni in più regioni.

Gli strumenti e i concetti chiave di IaC per Lambda includono framework per la creazione di modelli, la gestione dell'implementazione e le migliori pratiche per l'infrastruttura serverless:

- Principi IaC fondamentali per lo sviluppo Lambda
- CloudFormation, e AWS SAM funzionalità AWS CDK
- Criteri di selezione e confronto degli strumenti
- Le migliori pratiche per l'implementazione di Lambda IaC

Che tu stia lavorando in modo indipendente su un piccolo progetto o come parte di un grande team che gestisce applicazioni serverless su scala aziendale, questi strumenti di sviluppo e distribuzione possono aiutarti a scrivere, distribuire e gestire le tue funzioni Lambda in modo più efficace.

Per ulteriori informazioni, consulta [Utilizzo di Lambda con l'infrastructure as code \(IaC\)](#).

## GitHub Azioni e strumenti

GitHub Actions fornisce funzionalità di distribuzione automatizzata per le funzioni Lambda direttamente dal tuo repository di codice. Creando file di flusso di lavoro nel tuo repository, puoi distribuire automaticamente le funzioni Lambda ogni volta che vengono apportate modifiche al codice o alla configurazione, semplificando la pipeline di integrazione continua e distribuzione continua (). CI/CD L'azione della funzione Deploy Lambda offre un'interfaccia YAML dichiarativa che semplifica il processo di distribuzione, gestisce AWS le credenziali tramite OpenID Connect (OIDC) e supporta vari scenari di distribuzione tra cui aggiornamenti del codice, modifiche alla configurazione e convalide a secco. Questa integrazione consente ai team di mantenere un processo di distribuzione coerente e automatizzato utilizzando i flussi di lavoro esistenti. GitHub

Per ulteriori informazioni, consulta [Utilizzo GitHub delle azioni per implementare le funzioni Lambda](#).

## Powertools per AWS Lambda

Powertools for AWS è un toolkit per sviluppatori open source che consente di implementare le migliori pratiche serverless con un codice personalizzato minimo. Disponibile per Python, Java e .NET TypeScript/Node.js, fornisce funzioni di utilità, decoratori e middleware che semplificano le attività di sviluppo Lambda più comuni. Il toolkit include funzionalità di osservabilità integrate

come registrazione strutturata, tracciamento e raccolta di metriche, come utilità per il recupero dei parametri, la gestione dei segreti e i modelli di idempotenza. Questi strumenti sono in linea con best practice ben progettate e aiutano gli sviluppatori a creare applicazioni serverless pronte per la produzione in modo più AWS efficiente. Riducendo il codice standard e standardizzando i modelli comuni, Powertools for AWS consente ai team di concentrarsi sulla logica aziendale mantenendo al contempo un'implementazione coerente delle migliori pratiche serverless in tutte le loro applicazioni.

Per ulteriori informazioni, consulta [Utensili elettrici per AWS Lambda](#).

## Strumenti per la gestione del flusso di lavoro e degli eventi

Le applicazioni Lambda possono essere utilizzate nell'orchestrazione di flussi di lavoro complessi e nella gestione di vari eventi. AWS fornisce strumenti specializzati per aiutarti a gestire questi aspetti dello sviluppo serverless. Scopri di più sull' AWS Step Functions orchestrazione dei flussi di lavoro e su Amazon EventBridge per la gestione degli eventi e su come integrarli con le tue funzioni Lambda. Questi strumenti possono migliorare in modo significativo la scalabilità e l'affidabilità delle applicazioni serverless fornendo una solida gestione dello stato e architetture basate sugli eventi. Utilizzando questi servizi, è possibile creare Lambda-based soluzioni più sofisticate e resilienti in grado di gestire processi aziendali complessi e reagire a un'ampia gamma di eventi di sistema e applicazioni.

Per ulteriori informazioni, consulta [Gestione dei flussi di lavoro e degli eventi Lambda](#).

## Sviluppo locale di funzioni Lambda

Puoi spostare le tue funzioni Lambda dalla console Lambda al tuo ambiente di sviluppo integrato (IDE) preferito, che fornisce un ambiente di sviluppo completo in cui puoi utilizzare altre opzioni di sviluppo locali come e. AWS SAM AWS CDK La console Lambda supporta l'apertura di funzioni direttamente in Visual Studio Code, Kiro o Cursor.

### Principali vantaggi dello sviluppo locale

Mentre la console Lambda fornisce un modo rapido per modificare e testare le funzioni, lo sviluppo locale offre funzionalità più avanzate:

- Funzionalità IDE avanzate: strumenti di debug, completamento del codice e refactoring
- Sviluppo offline: lavora e testa le modifiche localmente prima dell'implementazione nel cloud
- Integrazione dell'infrastruttura come codice: utilizzo senza interruzioni con AWS SAM AWS CDK, e Infrastructure Composer
- Gestione delle dipendenze: pieno controllo sulle dipendenze delle funzioni

## Prerequisiti

Prima di sviluppare funzioni Lambda localmente utilizzando Visual Studio Code, Kiro e Cursor, è necessario installarle utilizzando le istruzioni seguenti:

- Visual Studio Code: per le istruzioni di installazione, vedi [Scaricare Visual Studio Code](#).
- Kiro: per le istruzioni di installazione, vedi [Guida introduttiva a Kiro](#).
- Cursor: per le istruzioni di installazione, vedi [Download Cursor](#).

Devi disporre anche dei seguenti elementi:

- AWS Toolkit for Visual Studio Code: Per le istruzioni di installazione di Visual Studio Code, vedi [Configurazione di. AWS Toolkit for Visual Studio Code](#) Per una panoramica, consulta [AWS Toolkit for Visual Studio Code](#).
- AWS credenziali: per informazioni sulla configurazione delle credenziali, vedi [Configurazione delle credenziali. AWS](#)
- AWS SAMCLI: per le istruzioni di installazione, vedere Installazione di. [AWS SAMCLI](#)
- Docker installato (opzionale, ma richiesto per i test locali): per le istruzioni di installazione, consulta [Get Docker](#).

### Note

Se hai già un AWS account e un profilo configurati localmente, assicurati che la policy AdministratorAccess gestita sia aggiunta al tuo profilo configurato AWS .

## Autenticazione e controllo degli accessi

Per sviluppare funzioni Lambda localmente, hai bisogno di AWS credenziali per accedere e gestire in modo sicuro AWS le risorse per tuo conto, proprio come farebbero nel cloud.

Il AWS Toolkit for Visual Studio Code supporta i seguenti metodi di autenticazione:

- Credenziali a lungo termine per utenti IAM
- Credenziali temporanee relative ai ruoli assunti
- Federazione delle identità

- AWS credenziali dell'utente root dell'account (non consigliato)

Questa sezione guida l'utente nell'ottenimento e nella configurazione di queste credenziali utilizzando le credenziali a lungo termine degli utenti IAM.

Ottieni le credenziali IAM

Se hai già un utente IAM con chiavi di accesso, tieni pronti sia l'ID della chiave di accesso che la chiave di accesso segreta per la sezione successiva. Se non disponi di queste chiavi, segui questi passaggi per crearle:

 Note

È necessario utilizzare sia l'ID chiave di accesso sia la chiave di accesso segreta insieme per autenticare le richieste dell'utente.

Per creare un utente IAM e le chiavi di accesso:

1. Apri la console IAM all'indirizzo <https://console.aws.amazon.com/iam/>
2. Nel pannello di navigazione, seleziona Utenti.
3. Selezionare Create user (Crea utente).
4. Per Nome utente, inserisci un nome e scegli Avanti.
5. In Imposta autorizzazioni, scegli Allega direttamente le politiche.
6. Seleziona AdministratorAccess e scegli Avanti.
7. Selezionare Create user (Crea utente).
8. Nel banner di successo, scegli Visualizza utente.
9. Selezionare Create access key (Crea chiave di accesso).
10. Per Caso d'uso, seleziona Codice locale.
11. Seleziona la casella di controllo di conferma e scegli Avanti.
12. (Facoltativo) Inserisci un valore per il tag descrittivo.
13. Selezionare Create access key (Crea chiave di accesso).
14. Copia immediatamente la chiave di accesso e la chiave di accesso segreta. Non potrai più accedere alla chiave di accesso segreta dopo aver lasciato questa pagina.

**⚠ Important**

Non condividere mai la tua chiave segreta né affidarla al controllo del codice sorgente. Conserva queste chiavi in modo sicuro ed eliminale quando non sono più necessarie.

**ℹ Note**

Per maggiori informazioni, consulta [Creare un utente IAM nel tuo AWS account](#) e [Gestire le chiavi di accesso per gli utenti IAM](#) nella IAM User Guide.

Configura AWS credenziali utilizzando il AWS Kit di strumenti

La tabella seguente riassume il processo di configurazione delle credenziali completato nella procedura seguente.

| Cosa fare                                                               | Perché?                                                  |
|-------------------------------------------------------------------------|----------------------------------------------------------|
| Apri il pannello Accedi                                                 | Avvio dell'autenticazione                                |
| Usa la palette dei comandi, cerca AWS<br>Aggiungi una nuova connessione | Accedi all'interfaccia utente di accesso                 |
| Scegli IAM Credential                                                   | Usa le tue chiavi di accesso per l'accesso programmatico |
| Inserisci il nome del profilo, la chiave di accesso, la chiave segreta  | Fornisci le credenziali per la connessione               |
| Vedi l'aggiornamento di AWS Explorer                                    | Conferma di essere connesso                              |

Completa i seguenti passaggi per autenticarti con il tuo AWS account:

**ℹ Note**

I passaggi seguenti si applicano a Visual Studio Code, Kiro e Cursor. Poiché Kiro e Cursor sono basati su Visual Studio Code, il flusso di autenticazione è lo stesso in tutti e tre gli IDE.

## 1. Apri il pannello Accedi nel tuo IDE:

Per avviare il processo di autenticazione, seleziona l' AWS icona nel riquadro di navigazione a sinistra o apri la palette dei comandi (Cmd+Shift+P su Mac o Ctrl+Shift+P su) e cerca e seleziona Aggiungi una nuova connessione. Windows/Linux AWS

## 2. Nel pannello di accesso, scegli IAM Credentials e seleziona Continua.

### Note

Per procedere, devi consentire alle estensioni AWS IDE di accedere ai tuoi dati.

## 3. Inserisci il nome del tuo profilo, l'ID della chiave di accesso e la chiave di accesso segreta, quindi seleziona Continua.

## 4. Verifica la connessione controllando l' AWS Explorer nel tuo IDE per i AWS servizi e le risorse.

Per informazioni sulla configurazione dell'autenticazione con credenziali a lungo termine, consulta [Utilizzo di credenziali a lungo termine per autenticare AWS SDK e strumenti.](#)

Per informazioni sulla configurazione dell'autenticazione, consulta le credenziali [AWS](#) IAM nella Guida per l'utente. AWS Toolkit for Visual Studio Code

## Passaggio dalla console allo sviluppo locale

### Note

Se hai apportato modifiche alla console, assicurati di non avere modifiche non implementate prima di passare allo sviluppo locale.

Per spostare una funzione Lambda dalla console Lambda a un IDE locale, completa i seguenti passaggi:

1. Aprire la [console Lambda](#).
2. Scegli il nome della funzione .
3. Seleziona la scheda Codice sorgente.
4. Apri la tua funzione nell'IDE che preferisci scegliendo la freccia del menu a discesa accanto al pulsante. Le opzioni supportate sono Apri in Visual Studio Code, Apri in Kiro e Apri in Cursor.

 Note

La selezione dell'IDE viene mantenuta. Una volta scelto un IDE, le visite successive mostrano quell'IDE come pulsante principale.

5. Quando richiesto, consenti al browser di aprire l'IDE selezionato.

Quando apri la funzione nel tuo IDE, Lambda crea un progetto locale con il codice della funzione in una posizione temporanea progettata per test e implementazione rapidi. Ciò include il codice della funzione, le dipendenze e una struttura di base del progetto che puoi utilizzare per lo sviluppo locale.

 Note

Il pulsante Apri in IDE richiede la versione 3.69.0 o successiva del AWS Toolkit. Se hai installato una versione precedente del AWS Toolkit, potresti visualizzare un messaggio. `Cannot open the handler` Per risolvere questo problema, aggiorna il tuo AWS Toolkit alla versione più recente.

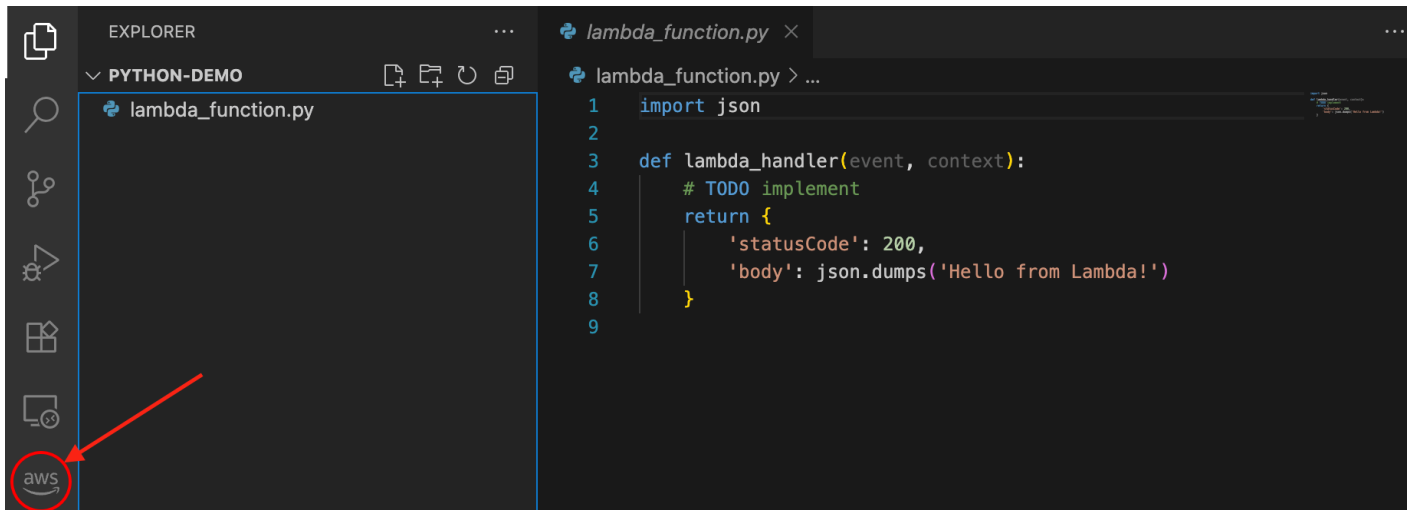
Per dettagli sull'utilizzo AWS nel tuo IDE, consulta la Guida [per l'AWS Toolkit for Visual Studio Code](#) utente.

## Lavorare con le funzioni a livello locale

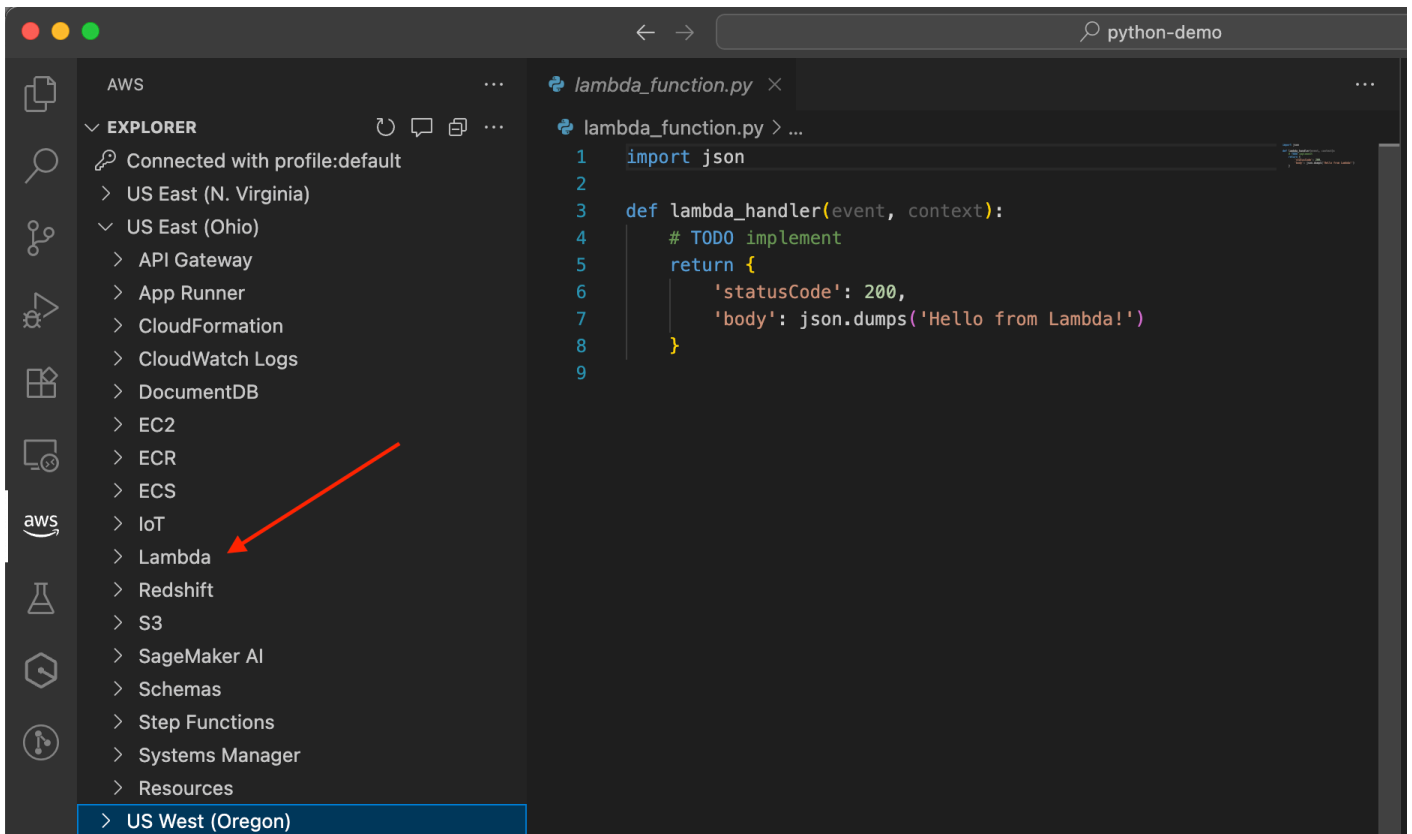
Dopo aver aperto la funzione nel tuo IDE preferito, segui questi passaggi per accedere e gestire le tue funzioni. Per l'esempio seguente abbiamo utilizzato Visual Studio Code.

1. Seleziona l' AWS icona nella barra laterale per aprire Explorer: AWS

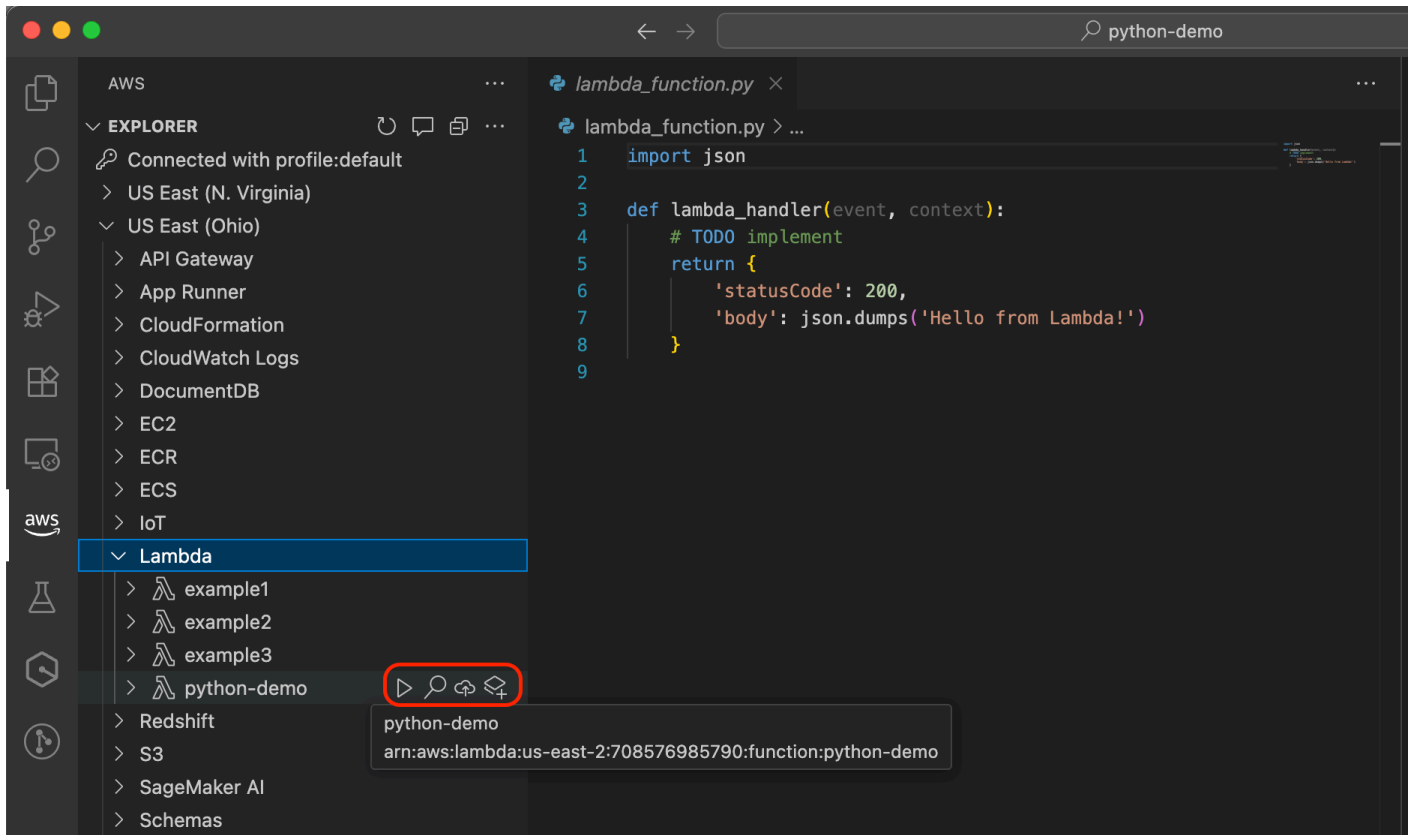




2. In AWS Explorer, seleziona la regione con la tua funzione Lambda:



3. Nella regione selezionata, espandi la sezione Lambda per visualizzare e gestire le tue funzioni:



Con la funzione aperta nel tuo IDE, puoi:

- Modifica il codice della funzione con supporto linguistico completo e completamento del codice.
- Usa l'[LocalStack integrazione](#) per testare le funzioni Lambda che effettuano chiamate API ad altri utenti Servizi AWS durante l'esecuzione, ad esempio leggendo da tabelle DynamoDB o scrivendo su bucket Amazon S3. LocalStack è un emulatore di servizi cloud che fornisce un ambiente di sviluppo locale completo per testare le integrazioni dei servizi. Puoi anche [usarlo AWS SAMCLI per testare la tua funzione in un contenitore locale](#). Se la tua funzione effettua chiamate API ad altre Servizi AWS, tali chiamate raggiungeranno AWS risorse reali, non emulate.
- Esegui il debug della tua funzione con punti di interruzione e ispezione delle variabili. Per ulteriori informazioni, consulta [Esecuzione e debug delle funzioni Lambda direttamente dal codice](#) nella Guida per l'utente. AWS Toolkit for Visual Studio Code
- Ridistribuisci la funzione aggiornata AWS utilizzando l'icona del cloud.
- Installa e gestisci le dipendenze per la tua funzione.

## Converti la tua funzione in una AWS SAM modello e usa gli strumenti IaC

In tutti gli IDE supportati (Visual Studio Code, Kiro e Cursor), puoi convertire la tua funzione Lambda in un AWS SAM modello scegliendo l'icona  Converti in AWS SAM applicazione accanto alla tua funzione Lambda. Ti viene richiesto di selezionare una posizione del progetto. AWS SAM Una volta selezionata, la funzione Lambda viene convertita in un `template.yaml` file salvato nel nuovo AWS SAM progetto.

Con la tua funzione convertita in AWS SAM modello, puoi:

- Controlla il controllo delle versioni della tua infrastruttura
- Automatizza le implementazioni
- Funzioni di debug in remoto
- Aggiungi AWS risorse aggiuntive alla tua applicazione
- Mantieni ambienti coerenti per tutto il ciclo di vita dello sviluppo
- Usa Infrastructure Composer per modificare visivamente il tuo modello AWS SAM

Per ulteriori informazioni sull'uso degli strumenti IaC, consulta le seguenti guide:

- [La Guida per AWS Serverless Application Model gli sviluppatori](#)
- [La guida Kit di sviluppo per il cloud AWS \(AWS CDK\) per gli sviluppatori](#)
- [La guida per gli sviluppatori di Infrastructure Composer](#)
- [La guida per AWS CloudFormation l'utente](#)

Questi strumenti forniscono funzionalità aggiuntive per la definizione, il test e l'implementazione delle applicazioni serverless.

### Fasi successive

Per saperne di più sull'utilizzo delle funzioni Lambda in Visual Studio Code, consulta le seguenti risorse:

- [Utilizzo delle funzioni AWS Lambda](#) nella Guida per l'utente del AWS Toolkit for Visual Studio Code
- [Utilizzo delle applicazioni serverless](#) nella Guida per l'utente di AWS Toolkit for Visual Studio Code

- [L'infrastruttura come codice](#) nella Lambda Developer Guide

## Utilizzo GitHub delle azioni per implementare le funzioni Lambda

Puoi usare [GitHub Actions](#) per distribuire automaticamente le funzioni Lambda quando invii modifiche al codice o alla configurazione nel tuo repository. L'azione [Implementa funzione Lambda](#) fornisce un'interfaccia YAML dichiarativa e semplice che elimina la complessità delle fasi di implementazione manuale.

### Flusso di lavoro di esempio

Per configurare l'implementazione automatizzata della funzione Lambda, crea un file di workflow nella directory `.github/workflows/` del repository:

Example GitHub Flusso di lavoro delle azioni per l'implementazione di Lambda

```
name: Deploy AWS Lambda
on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    permissions:
      id-token: write # Required for OIDC authentication
      contents: read # Required to check out the repository
    steps:
      - uses: actions/checkout@v4

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::123456789012:role/GitHubActionRole
          aws-region: us-east-1

      - name: Deploy Lambda Function
        uses: aws-actions/aws-lambda-deploy@v1
        with:
          function-name: my-lambda-function
```

```
code-artifacts-dir: ./dist
```

Questo flusso di lavoro viene eseguito quando invii le modifiche al ramo `main`. Controlla il tuo repository, configura AWS le credenziali utilizzando OpenID Connect (OIDC) e implementa la tua funzione utilizzando il codice nella directory. `./dist`

Per ulteriori esempi, tra cui l'aggiornamento della configurazione della funzione, la distribuzione tramite bucket S3 e la convalida a secco, consulta il README della funzione Deploy Lambda. <https://github.com/aws-actions/aws-lambda-deploy>

## Risorse aggiuntive

- [AWS GitHub Azione Configura credenziali](#)
- [Configurazione di OpenID Connect in AWS](#)

## Utilizzo di Lambda con l'infrastructure as code (IaC)

Le funzioni Lambda raramente vengono eseguite in modo isolato. Spesso fanno invece parte di un'applicazione serverless con altre risorse come database, code e spazio di archiviazione. Con [Infrastructure as code \(IaC\)](#), puoi automatizzare i processi di distribuzione per implementare e aggiornare in modo rapido e ripetibile intere applicazioni serverless che coinvolgono molte risorse separate. AWS Questo approccio accelera il ciclo di sviluppo, semplifica la gestione della configurazione e garantisce che le risorse vengano implementate sempre allo stesso modo.

## Strumenti di IaC per Lambda

### CloudFormation

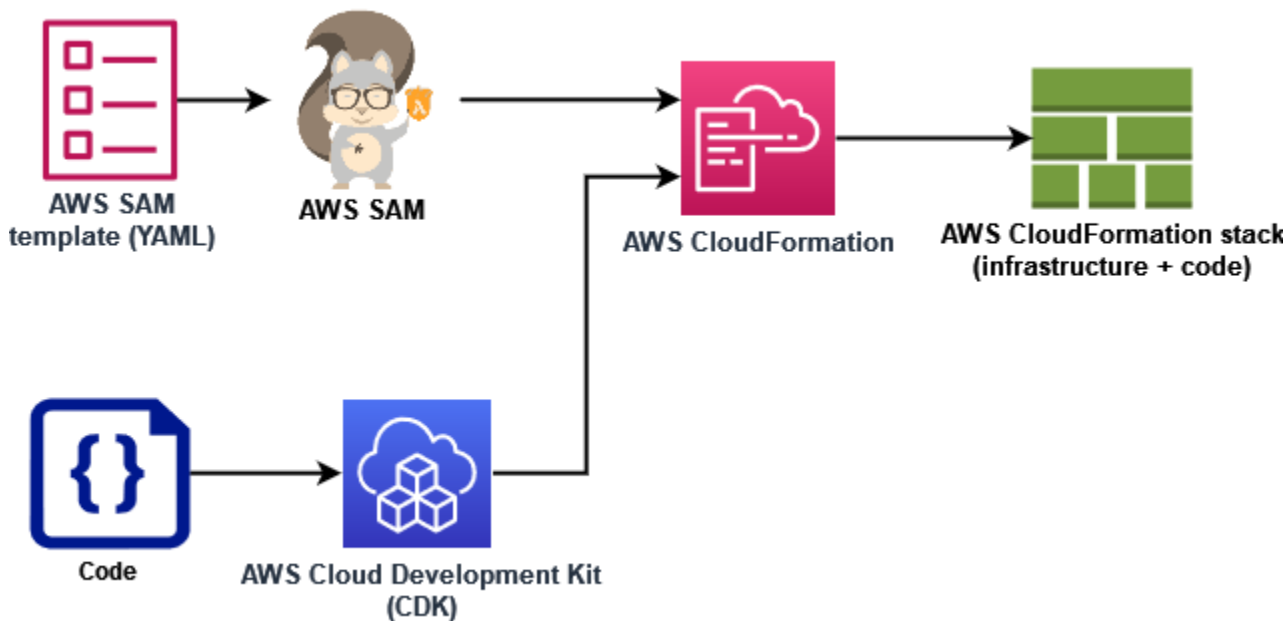
CloudFormation è il servizio IaC fondamentale di AWS. Puoi utilizzare i modelli [YAML o JSON](#) per modellare ed eseguire il provisioning dell'intera AWS infrastruttura, comprese le funzioni Lambda. CloudFormation gestisce le complessità legate alla creazione, all'aggiornamento e all'eliminazione delle risorse. AWS

### AWS Serverless Application Model (AWS SAM)

AWS SAM è un framework open source basato su CloudFormation. Fornisce una sintassi semplificata per definire applicazioni serverless. Usa i [modelli AWS SAM](#) per effettuare rapidamente il provisioning di funzioni Lambda, API, database e origini eventi con poche righe di YAML.

## Kit di sviluppo per il cloud AWS (AWS CDK)

Il CDK è un approccio basato sul codice a IaC. Puoi definire la tua Lambda-based architettura usando Python TypeScript JavaScript, Java, C#/Net o Go. Scegli il tuo linguaggio preferito e utilizza elementi di programmazione come parametri, condizionali, cicli, composizione ed ereditarietà per definire il risultato desiderato della tua infrastruttura. Il CDK genera quindi i modelli sottostanti per la distribuzione. CloudFormation Per un esempio di come utilizzare Lambda con CDK, consulta [Implementazione di funzioni Lambda con AWS CDK](#).



AWS fornisce anche un servizio chiamato AWS Infrastructure Composer a sviluppare modelli IaC utilizzando una semplice interfaccia grafica. Con Infrastructure Composer, si progetta un'architettura applicativa trascinando, raggruppando e connettendo in una tela visiva. Servizi AWS Infrastructure Composer crea quindi un AWS SAM modello o un CloudFormation modello a partire dal progetto che è possibile utilizzare per distribuire l'applicazione.

Nella sezione [the section called “Utilizzo di Infrastructure AWS SAM Composer”](#) seguente, Infrastructure Composer viene utilizzato per sviluppare un modello per un'applicazione serverless basata su una funzione Lambda esistente.

## Utilizzo delle funzioni Lambda in AWS SAM e Infrastructure Composer

In questo tutorial, puoi iniziare a utilizzare IaC con Lambda creando un AWS SAM modello da una funzione Lambda esistente e quindi creando un'applicazione serverless in Infrastructure Composer aggiungendo altre risorse. AWS

Mentre svolgi questo tutorial, imparerai alcuni concetti fondamentali, ad esempio come AWS vengono specificate le risorse in. AWS SAM Imparerai anche come utilizzare Infrastructure Composer per creare un'applicazione serverless da distribuire utilizzando o. AWS SAM CloudFormation

Per completare questo tutorial, eseguirai i seguenti passaggi:

- Creazione di una funzione Lambda di esempio
- Usa la console Lambda per visualizzare il AWS SAM modello per la funzione
- Esporta la configurazione della tua funzione AWS Infrastructure Composer e progetta una semplice applicazione serverless basata sulla configurazione della tua funzione
- Salva un AWS SAM modello aggiornato da utilizzare come base per distribuire la tua applicazione serverless

### Prerequisiti

In questo tutorial, si utilizza la [funzionalità di sincronizzazione](#) locale di Infrastructure Composer per salvare i file di modello e codice nel computer di compilazione locale. Per utilizzare questa funzionalità, è necessario un browser che supporti l'API File System Access, che consente alle applicazioni web di leggere, scrivere e salvare file nel file system locale. Ti consigliamo di utilizzare Google Chrome o Microsoft Edge. Per ulteriori informazioni sull'API File System Access, consulta la pagina [What is the File System Access API?](#)

### Creazione di una funzione Lambda

In questo primo passaggio, creerai una funzione Lambda da utilizzare nei passaggi successivi del tutorial. Per semplicità, utilizzerai la console Lambda per creare una funzione di base "Hello world" tramite il runtime Python 3.11.

### Creazione di una funzione Lambda "Hello world" tramite la console

1. Aprire la [console Lambda](#).
2. Scegli Crea funzione.

3. Lascia selezionato Crea da zero e, in Informazioni di base, immetti **LambdaIaCDemo** come Nome della funzione.
4. In Runtime, scegli Python 3.11.
5. Scegli Crea funzione.

Visualizza il AWS SAM modello per la tua funzione

Prima di esportare la configurazione della funzione in Infrastructure Composer, utilizza la console Lambda per visualizzare la configurazione corrente della funzione come modello AWS SAM . Seguendo i passaggi di questa sezione, imparerai a conoscere l'anatomia di un AWS SAM modello e a definire risorse come le funzioni Lambda per iniziare a specificare un'applicazione serverless.

Per visualizzare AWS SAM modello per la tua funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegli la funzione creata in precedenza (LambdaIaCDemo).
3. Nel riquadro Panoramica della funzione, scegli Modello.

Al posto del diagramma che rappresenta la configurazione della tua funzione, vedrai un AWS SAM modello per la tua funzione. Il modello avrà un aspetto simile al seguente.

```
# This AWS SAM template has been generated from your function's
# configuration. If your function has one or more triggers, note
# that the AWS resources associated with these triggers aren't fully
# specified in this template and include placeholder values.Open this template
# in AWS Application Composer or your favorite IDE and modify
# it to specify a serverless application with other AWS resources.
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Specification template describing your function.
Resources:
  LambdaIaCDemo:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 3
      Handler: lambda_function.lambda_handler
      Runtime: python3.11
```



```

Architectures:
  - x86_64
EventInvokeConfig:
  MaximumEventAgeInSeconds: 21600
  MaximumRetryAttempts: 2
EphemeralStorage:
  Size: 512
RuntimeManagementConfig:
  UpdateRuntimeOn: Auto
SnapStart:
  ApplyOn: None
PackageType: Zip
Policies:
  Statement:
    - Effect: Allow
      Action:
        - logs:CreateLogGroup
      Resource: arn:aws:logs:us-east-1:123456789012:*
    - Effect: Allow
      Action:
        - logs:CreateLogStream
        - logs:PutLogEvents
      Resource:
        - >-
          arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/
LambdaIaCDemo:*

```

Prendiamoci un momento per esaminare il modello YAML per la tua funzione e comprendere alcuni concetti chiave.

Il modello inizia con la dichiarazione `Transform: AWS::Serverless-2016-10-31`. Questa dichiarazione è necessaria perché dietro le quinte, i AWS SAM template vengono distribuiti tramite CloudFormation. L'utilizzo dell'istruzione `Transform` identifica il modello come file di modello AWS SAM.

Dopo la dichiarazione `Transform`, arriva la sezione `Resources`. Qui vengono definite le AWS risorse che desideri distribuire con il tuo AWS SAM modello. AWS SAM i modelli possono contenere una combinazione di AWS SAM risorse e CloudFormation risorse. Questo perché durante la distribuzione, AWS SAM i modelli si espandono in CloudFormation modelli, quindi è possibile aggiungere qualsiasi CloudFormation sintassi valida a un AWS SAM modello.

Al momento, nella sezione Resources del modello è presente soltanto una risorsa definita, la funzione Lambda LambdaIaCDemo. Per aggiungere una funzione Lambda a un AWS SAM modello, si utilizza il `AWS::Serverless::Function` tipo di risorsa. La risorsa Properties di una funzione Lambda definisce il runtime, il gestore della funzione e altre opzioni di configurazione. Qui viene definito anche il percorso del codice sorgente della funzione da AWS SAM utilizzare per distribuire la funzione. Per saperne di più sulle risorse delle funzioni Lambda in AWS SAM, consulta [AWS::Serverless::Function](#) nella Developer Guide. AWS SAM

Oltre alle proprietà e alle configurazioni della funzione, il modello specifica anche una policy AWS Identity and Access Management (IAM) per la funzione. Questa policy autorizza la tua funzione a scrivere log su Amazon CloudWatch Logs. Quando crei una funzione nella console Lambda, Lambda associa automaticamente questa policy alla tua funzione. Per saperne di più su come specificare una policy IAM per una funzione in un AWS SAM modello, consulta la `policies` proprietà nella [pagina AWS::Serverless::Function](#) della AWS SAM Developer Guide.

Per saperne di più sulla struttura dei AWS SAM modelli, consulta [AWS SAM l'anatomia dei modelli](#).

Utilizzo AWS Infrastructure Composer per progettare un'applicazione serverless

Per iniziare a creare una semplice applicazione serverless utilizzando il AWS SAM modello della funzione come punto di partenza, è necessario esportare la configurazione della funzione in Infrastructure Composer e attivare la modalità di sincronizzazione locale di Infrastructure Composer. La sincronizzazione locale salva automaticamente il codice della funzione e il AWS SAM modello nel computer di compilazione locale e mantiene sincronizzato il modello salvato quando si aggiungono altre AWS risorse in Infrastructure Composer.

Per esportare una funzione Lambda in Infrastructure Composer

1. Nel riquadro Panoramica della funzione, scegli Esporta in Strumento per la creazione di applicazioni.

Per esportare la configurazione e il codice della funzione in Infrastructure Composer, Lambda crea un bucket Amazon S3 nel tuo account in cui archiviare temporaneamente questi dati.

2. Nella finestra di dialogo, scegli Conferma e crea progetto per accettare il nome predefinito per questo bucket ed esportare la configurazione e il codice della funzione in Infrastructure Composer.
3. (Facoltativo) Per scegliere un altro nome per il bucket Amazon S3 creato da Lambda, immetti un nuovo nome e scegli Conferma e crea progetto. I nomi dei bucket Amazon S3 devono essere univoci a livello globale e seguire le [regole di denominazione dei bucket](#).

Selezionando Conferma e crea progetto si apre la console Infrastructure Composer. Nella tela, vedrai la tua funzione Lambda.

4. Dal menu a discesa Menu, scegli Attiva sincronizzazione locale.
5. Nella finestra di dialogo che si apre, scegli Seleziona cartella e seleziona una cartella sul tuo computer di compilazione locale.
6. Scegli Attiva per attivare la sincronizzazione locale.

Per esportare la funzione in Infrastructure Composer, è necessaria l'autorizzazione a utilizzare determinate operazioni API. Se non sei in grado di esportare la funzione, consulta [the section called "Autorizzazioni richieste"](#) e assicurati di disporre delle autorizzazioni necessarie.

#### Note

Il bucket che Lambda crea quando esporti una funzione in Infrastructure Composer è soggetto ai [prezzi di Amazon S3](#) standard. Gli oggetti che Lambda inserisce nel bucket vengono eliminati automaticamente dopo 10 giorni, ma il bucket in sé non viene eliminato. Per evitare di aggiungere costi aggiuntivi alla tua Account AWS, segui le istruzioni in [Eliminazione di un bucket](#) dopo aver esportato la funzione in Infrastructure Composer. Per ulteriori informazioni sul bucket Amazon S3 creato da Lambda, consulta la pagina [the section called "Infrastructure Composer"](#).

Per progettare un'applicazione serverless in Infrastructure Composer

Dopo aver attivato la sincronizzazione locale, le modifiche apportate in Infrastructure Composer si riflettono nel AWS SAM modello salvato sul computer di compilazione locale. Ora puoi trascinare AWS risorse aggiuntive nell'area di disegno di Infrastructure Composer per creare la tua applicazione. In questo esempio, aggiungi una coda semplice Amazon SQS come trigger per la funzione Lambda e una tabella DynamoDB in cui la funzione può scrivere i dati.

1. Per aggiungere un trigger Amazon SQS alla funzione Lambda, effettua le seguenti operazioni:
  - a. Nel campo di ricerca della palette Risorse, immetti **SQS**.
  - b. Trascina la risorsa Coda SQS sul canvas e posizionala a sinistra della funzione Lambda.
  - c. Seleziona Dettagli e per ID logico immetti **LambdaIaCQueue**.
  - d. Scegli Save (Salva).

- e. Connetti le tue risorse Amazon SQS e Lambda facendo clic sulla porta Abbonamento sulla scheda della coda SQS e trascinandola sulla porta sinistra della scheda della funzione Lambda. La comparsa di una linea tra le due risorse indica che la connessione è stabilita. Inoltre, nella parte inferiore del canvas, Infrastructure Composer visualizza un messaggio che indica che le due risorse sono state connesse correttamente.
2. Per aggiungere una tabella Amazon DynamoDB in cui la funzione Lambda può scrivere dati, effettua le seguenti operazioni:
    - a. Nel campo di ricerca della palette Risorse, immetti **DynamoDB**.
    - b. Trascina la risorsa Tabella DynamoDB sul canvas e posizionala a destra della funzione Lambda.
    - c. Seleziona Dettagli e per ID logico immetti **LambdaIaCTable**.
    - d. Scegli Save (Salva).
    - e. Connetti la tabella DynamoDB alla funzione Lambda facendo clic sulla porta destra della scheda della funzione Lambda e trascinandola sulla porta sinistra della scheda di DynamoDB.

Ora che hai aggiunto queste risorse extra, diamo un'occhiata al AWS SAM modello aggiornato creato da Infrastructure Composer.

Per visualizzare il tuo aggiornamento AWS SAM modello

- Nel canvas di Infrastructure Composer, scegli Modello per passare dalla visualizzazione del canvas alla visualizzazione del modello.

Il AWS SAM modello dovrebbe ora contenere le seguenti risorse e proprietà aggiuntive:

- Una coda Amazon SQS con l'identificativo LambdaIaCQueue

```
LambdaIaCQueue:
  Type: AWS::SQS::Queue
  Properties:
    MessageRetentionPeriod: 345600
```

Quando aggiungi una coda Amazon SQS utilizzando Infrastructure Composer, questo imposta la proprietà `MessageRetentionPeriod`. Puoi anche impostare la proprietà `FifoQueue` selezionando `Dettagli` sulla scheda della coda SQS e selezionando o deselezionando `Coda Fifo`.

Per impostare altre proprietà per la coda, puoi modificare manualmente il modello per aggiungerle. Per ulteriori informazioni sulla risorsa `AWS::SQS::Queue` e sulle sue proprietà disponibili, consulta la sezione [AWS::SQS::Queue](#) nella Guida per l'utente di CloudFormation.

- Una proprietà `Events` nella definizione della funzione Lambda che specifica la coda Amazon SQS come trigger per la funzione

```
Events:
  LambdaIaCQueue:
    Type: SQS
    Properties:
      Queue: !GetAtt LambdaIaCQueue.Arn
      BatchSize: 1
```

La proprietà `Events` è composta da un tipo di evento e da un insieme di proprietà che dipendono dal tipo. Per ulteriori informazioni sulle diverse configurazioni Servizi AWS che è possibile configurare per attivare una funzione Lambda e sulle proprietà che è possibile impostare, consulta [EventSource](#) la Guida per gli AWS SAM sviluppatori.

- Una tabella DynamoDB con l'identificatore `LambdaIaCTable`

```
LambdaIaCTable:
  Type: AWS::DynamoDB::Table
  Properties:
    AttributeDefinitions:
      - AttributeName: id
        AttributeType: S
    BillingMode: PAY_PER_REQUEST
    KeySchema:
      - AttributeName: id
        KeyType: HASH
    StreamSpecification:
      StreamViewType: NEW_AND_OLD_IMAGES
```

Quando aggiungi una tabella DynamoDB utilizzando Infrastructure Composer, puoi impostare le chiavi della tabella scegliendo `Dettagli` nella scheda della tabella DynamoDB e modificando i valori

delle chiavi. Infrastructure Composer imposta anche i valori predefiniti anche per una serie di altre proprietà, tra cui `BillingMode` e `StreamViewType`.

Per saperne di più su queste proprietà e altre proprietà che puoi aggiungere al tuo AWS SAM modello, consulta [AWS: :DynamoDB: :Table nella Guida per l'utente. CloudFormation](#)

- Una nuova policy IAM che autorizza la funzione a eseguire operazioni CRUD sulla tabella DynamoDB che hai aggiunto.

```
Policies:
...
- DynamoDBCrudPolicy:
  TableName: !Ref LambdaIaCTable
```

Il AWS SAM modello finale completo dovrebbe essere simile al seguente.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Specification template describing your function.
Resources:
  LambdaIaCDemo:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 3
      Handler: lambda_function.lambda_handler
      Runtime: python3.11
      Architectures:
        - x86_64
      EventInvokeConfig:
        MaximumEventAgeInSeconds: 21600
        MaximumRetryAttempts: 2
      EphemeralStorage:
        Size: 512
      RuntimeManagementConfig:
        UpdateRuntimeOn: Auto
      SnapStart:
        ApplyOn: None
      PackageType: Zip
      Policies:
```

```

- Statement:
  - Effect: Allow
    Action:
      - logs:CreateLogGroup
    Resource: arn:aws:logs:us-east-1:594035263019:*
  - Effect: Allow
    Action:
      - logs:CreateLogStream
      - logs:PutLogEvents
    Resource:
      - arn:aws:logs:us-east-1:594035263019:log-group:/aws/lambda/

LambdaIaCDemo:*
  - DynamoDBCrudPolicy:
      TableName: !Ref LambdaIaCTable
  Events:
    LambdaIaCQueue:
      Type: SQS
      Properties:
        Queue: !GetAtt LambdaIaCQueue.Arn
        BatchSize: 1
  Environment:
    Variables:
      LAMBDAIACTABLE_TABLE_NAME: !Ref LambdaIaCTable
      LAMBDAIACTABLE_TABLE_ARN: !GetAtt LambdaIaCTable.Arn
  LambdaIaCQueue:
    Type: AWS::SQS::Queue
    Properties:
      MessageRetentionPeriod: 345600
  LambdaIaCTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: id
          AttributeType: S
      BillingMode: PAY_PER_REQUEST
      KeySchema:
        - AttributeName: id
          KeyType: HASH
      StreamSpecification:
        StreamViewType: NEW_AND_OLD_IMAGES

```

## Distribuisci la tua applicazione serverless utilizzando AWS SAM (facoltativo)

Se desideri utilizzare AWS SAM per distribuire un'applicazione serverless utilizzando il modello appena creato in Infrastructure Composer, devi prima installare il. AWS SAM CLI Per fare ciò, segui le istruzioni in [Installazione della CLI. AWS SAM](#)

Inoltre, prima di implementare l'applicazione devi aggiornare il codice della funzione che Infrastructure Composer ha salvato insieme al modello. Al momento, il file `lambda_function.py` salvato da Infrastructure Composer contiene solo il codice di base "Hello world" fornito da Lambda al momento della creazione della funzione.

Per aggiornare il codice della funzione, copia il codice seguente e incollalo nel file `lambda_function.py` che Infrastructure Composer ha salvato sul computer di compilazione locale. Hai specificato la directory in cui Infrastructure Composer deve salvare questo file quando hai attivato la modalità di sincronizzazione locale.

Questo codice accetta una coppia chiave-valore in un messaggio dalla coda Amazon SQS creata in Infrastructure Composer. Se sia la chiave sia il valore sono stringhe, il codice li utilizza per scrivere un elemento nella tabella DynamoDB definita nel modello.

### Codice della funzione Python aggiornato

```
import boto3
import os
import json

# define the DynamoDB table that Lambda will connect to
tablename = os.environ['LAMBDAIACTABLE_TABLE_NAME']

# create the DynamoDB resource
dynamo = boto3.client('dynamodb')

def lambda_handler(event, context):
    # get the message out of the SQS event
    message = event['Records'][0]['body']
    data = json.loads(message)
    # write event data to DDB table
    if check_message_format(data):
        key = next(iter(data))
        value = data[key]
        dynamo.put_item(
```



```
        TableName=tablename,
        Item={
            'id': {'S': key},
            'Value': {'S': value}
        }
    )
else:
    raise ValueError("Input data not in the correct format")

# check that the event object contains a single key value
# pair that can be written to the database
def check_message_format(message):
    if len(message) != 1:
        return False

    key, value = next(iter(message.items()))

    if not (isinstance(key, str) and isinstance(value, str)):
        return False

    else:
        return True
```

## Implementazione dell'applicazione serverless

Per distribuire l'applicazione utilizzando il AWS SAM CLI, procedi nel seguente modo. Affinché la funzione venga compilata e implementata correttamente, sul computer di compilazione e nel PATH deve essere installata la versione 3.11 di Python.

1. Esegui il seguente comando dalla directory in cui Infrastructure Composer ha salvato i tuoi file `template.yaml` e `lambda_function.py`.

```
sam build
```

Questo comando raccoglie gli artefatti di compilazione per l'applicazione e li colloca nel formato e nella posizione corretti per l'implementazione.

2. Per distribuire l'applicazione e creare le risorse Lambda, Amazon SQS e DynamoDB specificate nel AWS SAM modello, esegui il seguente comando.

```
sam deploy --guided
```

L'uso del `--guided` flag significa che ti AWS SAM verranno mostrate istruzioni per guidarti nel processo di distribuzione. Per questa implementazione, accetta le opzioni predefinite premendo Invio.

Durante il processo di distribuzione, AWS SAM crea le seguenti risorse nel tuo Account AWS

- Una CloudFormation [pila](#) denominata `sam-app`
- Una funzione Lambda con il formato del nome `sam-app-LambdaIaCDemo-99VXPpYQVv1M`
- Una coda Amazon SQS con il formato del nome `sam-app-LambdaIaCQueue-xL87VeKsGiIo`
- Una tabella DynamoDB con il formato del nome `sam-app-LambdaIaCTable-CN0S66C0VLNV`

AWS SAM crea anche i ruoli e le policy IAM necessari in modo che la funzione Lambda possa leggere i messaggi dalla coda Amazon SQS ed eseguire operazioni CRUD sulla tabella DynamoDB.

Test dell'applicazione implementata (facoltativo)

Per confermare che l'applicazione serverless sia stata implementata correttamente, invia un messaggio alla coda Amazon SQS contenente una coppia chiave-valore e verifica che Lambda scriva un elemento nella tabella DynamoDB utilizzando questi valori.

Test dell'applicazione serverless

1. Apri la pagina [Code](#) della console Amazon SQS e seleziona la coda che AWS SAM ha creato dal modello. Il formato del nome è `sam-app-LambdaIaCQueue-xL87VeKsGiIo`.
2. Scegli Invio e ricezione di messaggi e incolla il seguente JSON nel Corpo del messaggio nella sezione Invia messaggio.

```
{
  "myKey": "myValue"
}
```

3. Scegliere Invia messaggio.

L'invio del messaggio alla coda farà sì che Lambda chiami la funzione tramite lo strumento di mappatura dell'origine degli eventi definito nel modello AWS SAM . Per verificare che Lambda abbia chiamato la funzione come previsto, verifica che alla tabella DynamoDB sia stato aggiunto un elemento.

4. Apri la pagina [Tabelle](#) della console DynamoDB e scegli la tabella. Il formato del nome è sam-app-LambdaIaCTable-**CN0S66C0VLNV**.
5. Scegli Explore table items (Esplora elementi della tabella). Nel riquadro Elementi restituiti, dovresti vedere un elemento con l'id myKey e il valore myValue.

## Implementazione di funzioni Lambda con AWS CDK

Kit di sviluppo per il cloud AWS (AWS CDK) È un framework Infrastructure as code (IaC) che puoi utilizzare per definire l'infrastruttura AWS cloud utilizzando un linguaggio di programmazione a tua scelta. Per definire la tua infrastruttura cloud, devi innanzitutto scrivere un'app (in uno dei linguaggi supportati dal CDK) contenente uno o più stack. Quindi, lo sintetizzi in un CloudFormation modello e distribuisce le tue risorse sul tuo Account AWS. Segui i passaggi descritti in questo argomento per implementare una funzione Lambda che restituisce un evento da un endpoint Gateway Amazon API.

La AWS Construct Library, inclusa nel CDK, fornisce moduli che è possibile utilizzare per modellare le risorse fornite. Servizi AWS Per i servizi più diffusi, la libreria fornisce costrutti curati con impostazioni predefinite intelligenti e best practice. Puoi utilizzare il modulo [aws\\_lambda](#) per definire la tua funzione e le risorse di supporto con poche righe di codice.

### Prerequisiti

Prima di iniziare questo tutorial, installa AWS CDK eseguendo il seguente comando.

```
npm install -g aws-cdk
```

### Passaggio 1: configura il tuo AWS CDK project

Crea una directory per la tua nuova AWS CDK app e inizializza il progetto.

#### JavaScript

```
mkdir hello-lambda
cd hello-lambda
cdk init --language javascript
```

#### TypeScript

```
mkdir hello-lambda
cd hello-lambda
cdk init --language typescript
```

## Python

```
mkdir hello-lambda
cd hello-lambda
cdk init --language python
```

Una volta avviato il progetto, attiva l'ambiente virtuale del progetto e installa le dipendenze di base per AWS CDK.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

## Java

```
mkdir hello-lambda
cd hello-lambda
cdk init --language java
```

Importa questo progetto Maven nel tuo ambiente di sviluppo integrato (IDE) Java. Ad esempio, in Eclipse, usa File, Importa, Maven, Progetti Maven esistenti.

## C#

```
mkdir hello-lambda
cd hello-lambda
cdk init --language csharp
```

### Note

Il modello di AWS CDK applicazione utilizza il nome della directory del progetto per generare nomi per i file di origine e le classi. In questo esempio, la directory è denominata hello-lambda. Se utilizzi un nome di directory del progetto diverso, l'app non corrisponderà a queste istruzioni.

AWS CDK v2 include costrutti stabili per tutti Servizi AWS in un unico pacchetto chiamato. `aws-cdk-lib` Questo pacchetto viene installato come dipendenza quando iniziizzi il progetto. Quando lavori con determinati linguaggi di programmazione, il pacchetto viene installato quando crei il progetto per la prima volta.

## Fase 2: Definire il AWS CDK stack

Uno stack CDK è una raccolta di uno o più costrutti che definiscono le risorse. AWS Ogni stack CDK rappresenta uno CloudFormation stack nella tua app CDK.

Per definire il tuo stack CDK, segui le istruzioni del tuo linguaggio di programmazione preferito. Questo stack definisce quanto segue:

- Il nome logico della funzione: `MyFunction`
- La posizione del codice della funzione, specificata nella proprietà `code`. Per ulteriori informazioni, consulta [Codice dell'handler](#) nella Documentazione di riferimento API Kit di sviluppo per il cloud AWS (AWS CDK) .
- Il nome logico della REST API: `HelloApi`
- Il nome logico dell'endpoint API Gateway: `ApiGwEndpoint`

Nota che tutti gli stack CDK di questo tutorial utilizzano il Node.js [runtime](#) per la funzione Lambda. È possibile utilizzare diversi linguaggi di programmazione per lo stack CDK e la funzione Lambda per sfruttare i punti di forza di ciascun linguaggio. Ad esempio, puoi utilizzare lo stack CDK TypeScript per sfruttare i vantaggi della digitazione statica per il codice dell'infrastruttura. Puoi utilizzare la funzione Lambda JavaScript per sfruttare la flessibilità e il rapido sviluppo di un linguaggio digitato dinamicamente.

### JavaScript

Apri il file `lib/hello-lambda-stack.js` e sostituisci il contenuto con quanto riportato di seguito.

```
const { Stack } = require('aws-cdk-lib');
const lambda = require('aws-cdk-lib/aws-lambda');
const apigw = require('aws-cdk-lib/aws-apigateway');

class HelloLambdaStack extends Stack {
  /**
   *
   * @param {Construct} scope
   * @param {string} id
   * @param {StackProps=} props
   */
  constructor(scope, id, props) {
    super(scope, id, props);
```

```

const fn = new lambda.Function(this, 'MyFunction', {
  code: lambda.Code.fromAsset('lib/lambda-handler'),
  runtime: lambda.Runtime.NODEJS_LATEST,
  handler: 'index.handler'
});

const endpoint = new apigw.LambdaRestApi(this, 'MyEndpoint', {
  handler: fn,
  restApiName: "HelloApi"
});

}
}

module.exports = { HelloLambdaStack }

```

## TypeScript

Apri il file `lib/hello-lambda-stack.ts` e sostituisci il contenuto con quanto riportato di seguito.

```

import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as apigw from "aws-cdk-lib/aws-apigateway";
import * as lambda from "aws-cdk-lib/aws-lambda";
import * as path from 'node:path';

export class HelloLambdaStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props)
    const fn = new lambda.Function(this, 'MyFunction', {
      runtime: lambda.Runtime.NODEJS_LATEST,
      handler: 'index.handler',
      code: lambda.Code.fromAsset(path.join(__dirname, 'lambda-handler')),
    });

    const endpoint = new apigw.LambdaRestApi(this, `ApiGwEndpoint`, {
      handler: fn,
      restApiName: `HelloApi`,
    });

  }
}

```

## Python

Apri il file `/hello-lambda/hello_lambda/hello_lambda_stack.py` e sostituisci il contenuto con quanto riportato di seguito.

```
from aws_cdk import (
    Stack,
    aws_apigateway as apigw,
    aws_lambda as _lambda
)
from constructs import Construct

class HelloLambdaStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        fn = _lambda.Function(
            self,
            "MyFunction",
            runtime=_lambda.Runtime.NODEJS_LATEST,
            handler="index.handler",
            code=_lambda.Code.from_asset("lib/lambda-handler")
        )

        endpoint = apigw.LambdaRestApi(
            self,
            "ApiGwEndpoint",
            handler=fn,
            rest_api_name="HelloApi"
        )
```

## Java

Apri il file `/hello-lambda/src/main/java/com/myorg/HelloLambdaStack.java` e sostituisci il contenuto con quanto riportato di seguito.

```
package com.myorg;

import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.services.apigateway.LambdaRestApi;
```

```

import software.amazon.awscdk.services.lambda.Function;

public class HelloLambdaStack extends Stack {
    public HelloLambdaStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloLambdaStack(final Construct scope, final String id, final StackProps
props) {
        super(scope, id, props);

        Function hello = Function.Builder.create(this, "MyFunction")

        .runtime(software.amazon.awscdk.services.lambda.Runtime.NODEJS_LATEST)

        .code(software.amazon.awscdk.services.lambda.Code.fromAsset("lib/lambda-handler"))
            .handler("index.handler")
            .build();

        LambdaRestApi api = LambdaRestApi.Builder.create(this, "ApiGwEndpoint")
            .restApiName("HelloApi")
            .handler(hello)
            .build();
    }
}

```

## C#

Apri il file `src/HelloLambda/HelloLambdaStack.cs` e sostituisci il contenuto con quanto riportato di seguito.

```

using Amazon.CDK;
using Amazon.CDK.AWS.APIGateway;
using Amazon.CDK.AWS.Lambda;
using Constructs;

namespace HelloLambda
{
    public class HelloLambdaStack : Stack
    {
        internal HelloLambdaStack(Construct scope, string id, IStackProps props =
null) : base(scope, id, props)

```



```
{
  var fn = new Function(this, "MyFunction", new FunctionProps
  {
    Runtime = Runtime.NODEJS_LATEST,
    Code = Code.FromAsset("lib/lambda-handler"),
    Handler = "index.handler"
  });

  var api = new LambdaRestApi(this, "ApiGwEndpoint", new
  LambdaRestApiProps
  {
    Handler = fn
  });
}
}
```

### Fase 3: creazione del codice della funzione Lambda

1. Dalla root del tuo progetto (hello-lambda), crea la directory `/lib/lambda-handler` per il codice della funzione Lambda. Questa directory è specificata nella code proprietà del tuo stack. AWS CDK
2. Crea un nuovo file chiamato `index.js` nella `/lib/lambda-handler` directory. Incolla il codice seguente nel file. La funzione estrae proprietà specifiche dalla richiesta API e le restituisce come risposta JSON.

```
exports.handler = async (event) => {
  // Extract specific properties from the event object
  const { resource, path, httpMethod, headers, queryStringParameters, body } =
  event;
  const response = {
    resource,
    path,
    httpMethod,
    headers,
    queryStringParameters,
    body,
  };
  return {
    body: JSON.stringify(response, null, 2),
    statusCode: 200,
  };
}
```

```
};  
};
```

## Fase 4: Distribuisci il AWS CDK stack

1. Dalla root del progetto, esegui il comando [cdk synth](#):

```
cdk synth
```

Questo comando sintetizza un AWS CloudFormation modello dal tuo stack CDK. Il modello è un file YAML di circa 400 righe simile al seguente.

### Note

Se viene visualizzato il seguente errore, assicurarti di trovarti nella root della directory del tuo progetto.

```
--app is required either in command-line, in cdk.json or in ~/.cdk.json
```

## Example CloudFormation modello

```
Resources:  
  MyFunctionServiceRole3C357FF2:  
    Type: AWS::IAM::Role  
    Properties:  
      AssumeRolePolicyDocument:  
        Statement:  
          - Action: sts:AssumeRole  
            Effect: Allow  
            Principal:  
              Service: lambda.amazonaws.com  
        Version: "2012-10-17"  
      ManagedPolicyArns:  
        - Fn::Join:  
          - ""  
          - - "arn:"  
            - Ref: AWS::Partition  
            - :iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

```
Metadata:
  aws:cdk:path: HelloLambdaStack/MyFunction/ServiceRole/Resource
MyFunction1BAA52E7:
  Type: AWS::Lambda::Function
  Properties:
    Code:
      S3Bucket:
        Fn::Sub: cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}
      S3Key:
        ab111111cd32708dc4b83e81a21c296d607ff2cdef00f1d7f48338782f9213901.zip
      Handler: index.handler
    Role:
      Fn::GetAtt:
        - MyFunctionServiceRole3C357FF2
        - Arn
    Runtime: nodejs24.x
    ...
```

2. Esegui il comando [cdk deploy](#):

```
cdk deploy
```

Attendi che le tue risorse vengano create. L'output finale include l'URL per l'endpoint API Gateway. Esempio:

```
Outputs:
HelloLambdaStack.ApiGwEndpoint77F417B1 = https://abcd1234.execute-api.us-east-1.amazonaws.com/prod/
```

## Fase 5: esecuzione del test della funzione

Per richiamare la funzione Lambda, copia l'endpoint API Gateway e incollalo in un browser Web o esegui un comando `curl`:

```
curl -s https://abcd1234.execute-api.us-east-1.amazonaws.com/prod/
```

La risposta è una rappresentazione JSON delle proprietà selezionate dall'oggetto evento originale, che contiene informazioni sulla richiesta effettuata all'endpoint API Gateway. Esempio:

```
{
```

```
"resource": "/",
"path": "/",
"httpMethod": "GET",
"headers": {
  "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7",
  "Accept-Encoding": "gzip, deflate, br, zstd",
  "Accept-Language": "en-US,en;q=0.9",
  "CloudFront-Forwarded-Proto": "https",
  "CloudFront-Is-Desktop-Viewer": "true",
  "CloudFront-Is-Mobile-Viewer": "false",
  "CloudFront-Is-SmartTV-Viewer": "false",
  "CloudFront-Is-Tablet-Viewer": "false",
  "CloudFront-Viewer-ASN": "16509",
  "CloudFront-Viewer-Country": "US",
  "Host": "abcd1234.execute-api.us-east-1.amazonaws.com",
  ...
}
```

## Fase 6: eliminare le risorse

L'endpoint API Gateway è accessibile pubblicamente. Per evitare addebiti imprevisti, esegui il comando [cdk destroy](#) per eliminare lo stack e tutte le risorse associate.

```
cdk destroy
```

## Fasi successive

Per informazioni sulla scrittura di AWS CDK app nella lingua prescelta, consulta quanto segue:

### TypeScript

[Lavorare con l' AWS CDK in TypeScript](#)

### JavaScript

[Lavorare con la AWS CDK locanda JavaScript](#)

### Python

[Lavorare con AWS CDK in Python](#)

### Java

[Lavorare con AWS CDK in Java](#)

## C#

[Lavorare con AWS CDK in C#](#)

## Go

[Lavorare con AWS CDK in Go](#)

## Utensili elettrici per AWS Lambda

Powertools for AWS Lambda (noto anche come Powertools for AWS) fornisce funzioni di utilità, decoratori e middleware che gestiscono attività Lambda comuni come la registrazione strutturata, la traccia, la raccolta di metriche e la convalida degli input. Usa Powertools for AWS Lambda per implementare le best practice serverless e accelerare lo sviluppo di più funzioni Lambda. In questo modo si semplificano le attività di sviluppo più comuni nelle funzioni Lambda.

### Principali vantaggi di Powertools per AWS

Sebbene lo sviluppo Lambda sia possibile senza Powertools for AWS, il suo utilizzo offre diversi vantaggi:

- Built-in osservabilità: registrazione strutturata, tracciamento e metriche personalizzate
- Gestione dei segreti: recupero dei parametri, gestione dei segreti e idempotenza
- Miglioramento progressivo: scegli le utilità più adatte alle tue esigenze
- Sviluppo accelerato: analisi degli eventi, convalida ed elaborazione in batch
- Procedure consigliate: implementazione di modelli serverless AWS Well-Architected

### Integrazione di Powertools con AWS

Powertools for AWS consente di creare applicazioni serverless pronte per la produzione con meno codice personalizzato. Disponibile in Python, .NET e Java TypeScript/Node.js, Powertools for AWS può essere incluso tramite Lambda Layers o utilizzando il Language Package Manager. Ogni implementazione linguistica fornisce funzionalità di base come registrazione strutturata, tracciamento, raccolta di metriche e gestione degli eventi, pur mantenendo gli idiomi naturali di ogni linguaggio di programmazione. Queste implementazioni sono integrate da componenti specializzati per l'integrazione dei AWS servizi, che supportano il recupero dei parametri, l'elaborazione in batch e la gestione delle API, insieme a best practice come la propagazione degli ID di correlazione,

la gestione degli errori e i modelli di idempotenza. Insieme, queste funzionalità consentono agli sviluppatori di creare applicazioni serverless robuste e gestibili riducendo al contempo il sovraccarico di codice personalizzato.

- [Powertools per Lambda \(Python\) AWS](#)
- [Powertools per Lambda \(\) AWS TypeScript](#)
- [Powertools per AWS Lambda \(Java\)](#)
- [Powertools per AWS Lambda \(.NET\)](#)

## Fasi successive

Per saperne di più sull'utilizzo di Powertools per AWS, consulta le seguenti risorse:

- [Workshop Powertools for AWS Lambda](#)
- [Pattern serverless che utilizzano Powertools per AWS](#)
- [AWS obiettivo serverless ben progettato](#)
- [Creazione di API serverless con Powertools for Lambda AWS](#)

## Gestione dei flussi di lavoro e degli eventi Lambda

Quando crei applicazioni serverless con Lambda, spesso hai bisogno di modi per orchestrare l'esecuzione delle funzioni e gestire gli eventi. AWS fornisce diversi approcci per coordinare le funzioni Lambda:

- [Funzioni Lambda durable](#) per l'orchestrazione del flusso di lavoro basata sul codice all'interno di Lambda
- AWS Step Functions per l'orchestrazione visiva del flusso di lavoro su più servizi
- Amazon EventBridge Scheduler e Amazon EventBridge per architetture e pianificazione basate sugli eventi

Puoi anche integrare questi approcci insieme. Ad esempio, è possibile utilizzare EventBridge Scheduler per attivare funzioni durevoli o flussi di lavoro Step Functions quando si verificano eventi specifici o configurare i flussi di lavoro per pubblicare eventi su EventBridge Scheduler in punti di esecuzione definiti. I seguenti argomenti di questa sezione forniscono ulteriori informazioni su come utilizzare queste opzioni di orchestrazione.

## Code-first orchestrazione con funzioni durevoli

Le funzioni Lambda durable forniscono un approccio basato sul codice all'orchestrazione del flusso di lavoro, consentendoti di creare flussi di lavoro statici e di lunga durata direttamente all'interno delle tue funzioni Lambda. A differenza dei servizi di orchestrazione esterni, le funzioni durevoli mantengono la logica del flusso di lavoro in codice, semplificando la versione, il test e la manutenzione insieme alla logica aziendale.

Le funzioni durevoli sono ideali quando hai bisogno di:

- Usa linguaggi di programmazione standard: definisci i flussi di lavoro utilizzando linguaggi di programmazione familiari come JavaScript Python
- Long-running processi: esegui flussi di lavoro che possono durare fino a un anno, ben oltre il limite di 15 minuti delle funzioni Lambda standard
- Sviluppo semplificato: mantieni la logica del flusso di lavoro e la logica aziendale nella stessa base di codice per facilitare la manutenzione e il test
- Cost-effective attesa: sospende l'esecuzione durante gli stati di attesa senza consumare risorse di calcolo
- Built-in gestione dello stato: checkpoint automatico e persistenza dello stato senza configurazione di archiviazione esterna

Per informazioni sulla scelta tra funzioni durevoli e Funzioni Step, vedere Funzioni [durevoli o Funzioni Step](#).

Per ulteriori informazioni sulle funzioni durevoli, vedere Funzioni durevoli [Lambda](#).

## Orchestrazione dei flussi di lavoro con Step Functions

AWS Step Functions è un servizio di orchestrazione del flusso di lavoro che aiuta a coordinare più funzioni Lambda e altri AWS servizi in flussi di lavoro strutturati. Questi flussi di lavoro possono mantenere lo stato, gestire gli errori con sofisticati meccanismi di riprova ed elaborare i dati su larga scala.

Step Functions offre due tipi di flussi di lavoro per soddisfare diverse esigenze di orchestrazione:

### Flussi di lavoro standard

Ideale per flussi di lavoro verificabili e di lunga durata che richiedono una semantica di esecuzione esatta. I flussi di lavoro standard possono durare fino a un anno, fornire una cronologia dettagliata

delle esecuzioni e supportare il debug visivo. Sono adatti per processi come l'evasione degli ordini, le pipeline di elaborazione dei dati o i lavori di analisi in più fasi.

## Flussi di lavoro rapidi

Progettato per carichi di lavoro ad alta frequenza di eventi e di breve durata con semantica di esecuzione almeno una volta. I flussi di lavoro Express possono durare fino a cinque minuti e sono ideali per l'elaborazione di eventi ad alto volume, le trasformazioni di dati in streaming o gli scenari di acquisizione di dati IoT. Offrono una maggiore produttività e costi potenzialmente inferiori rispetto ai flussi di lavoro standard.

### Note

Per ulteriori informazioni sui tipi di flusso di lavoro Step Functions, vedi [Scelta del tipo di flusso di lavoro in Step Functions](#).

All'interno di questi flussi di lavoro, Step Functions fornisce due tipi di stati della mappa per l'elaborazione parallela:

## Mappa in linea

Elabora gli elementi di un array JSON all'interno della cronologia di esecuzione del flusso di lavoro principale. Inline Map supporta fino a 40 iterazioni simultanee ed è adatto per set di dati più piccoli o quando è necessario mantenere tutte le elaborazioni in un'unica esecuzione. Per ulteriori informazioni, vedere [Utilizzo dello stato della mappa in modalità Inline](#).

## Mappa distribuita

Consente l'elaborazione di carichi di lavoro paralleli su larga scala iterando su set di dati che superano i 256 KiB o richiedono più di 40 iterazioni simultanee. Con il supporto per un massimo di 10.000 esecuzioni parallele di flussi di lavoro secondari, Distributed Map eccelle nell'elaborazione di dati semistrutturati archiviati in Amazon S3, come file JSON o CSV, rendendolo ideale per l'elaborazione in batch e le operazioni ETL. Per ulteriori informazioni, [consulta Using Map state in modalità Distributed](#).

Combinando questi tipi di flusso di lavoro e gli stati delle mappe, Step Functions fornisce un set di strumenti flessibile e potente per orchestrare applicazioni serverless complesse, da operazioni su piccola scala a pipeline di elaborazione dati su larga scala.



Per iniziare a usare Lambda con Step Functions, consulta [Orchestrating Lambda functions with Step Functions](#). ???

## Gestione degli eventi con e Scheduler EventBridge EventBridge

Amazon EventBridge è un servizio di bus per eventi che ti aiuta a creare architetture basate sugli eventi. Instrada gli eventi tra AWS servizi, applicazioni integrate e applicazioni Software as a Service (SaaS). EventBridge Scheduler è uno scheduler senza server. Con EventBridge Scheduler, puoi creare, eseguire e gestire attività da un servizio centrale, richiamare le funzioni Lambda in base a una pianificazione utilizzando espressioni cron e rate o configurare invocazioni una tantum.

Amazon EventBridge e EventBridge Scheduler ti aiutano a creare architetture basate sugli eventi con Lambda. EventBridge Instrada gli eventi tra AWS servizi, applicazioni integrate e applicazioni SaaS, mentre EventBridge Scheduler fornisce funzionalità di pianificazione specifiche per richiamare le funzioni Lambda su base ricorrente o una tantum.

Questi servizi forniscono diverse funzionalità chiave per lavorare con le funzioni Lambda:

- Crea regole che corrispondano e indirizzino gli eventi alle funzioni Lambda utilizzando EventBridge
- Imposta invocazioni ricorrenti di funzioni utilizzando le espressioni cron e rate con Scheduler EventBridge
- Configura chiamate di funzioni una tantum in date e ore specifiche
- Definisci finestre temporali flessibili e riprova le politiche per le chiamate pianificate

Per ulteriori informazioni, consulta [Richiamo di una funzione Lambda su una pianificazione](#).

## Come eseguire test di funzioni e applicazioni serverless

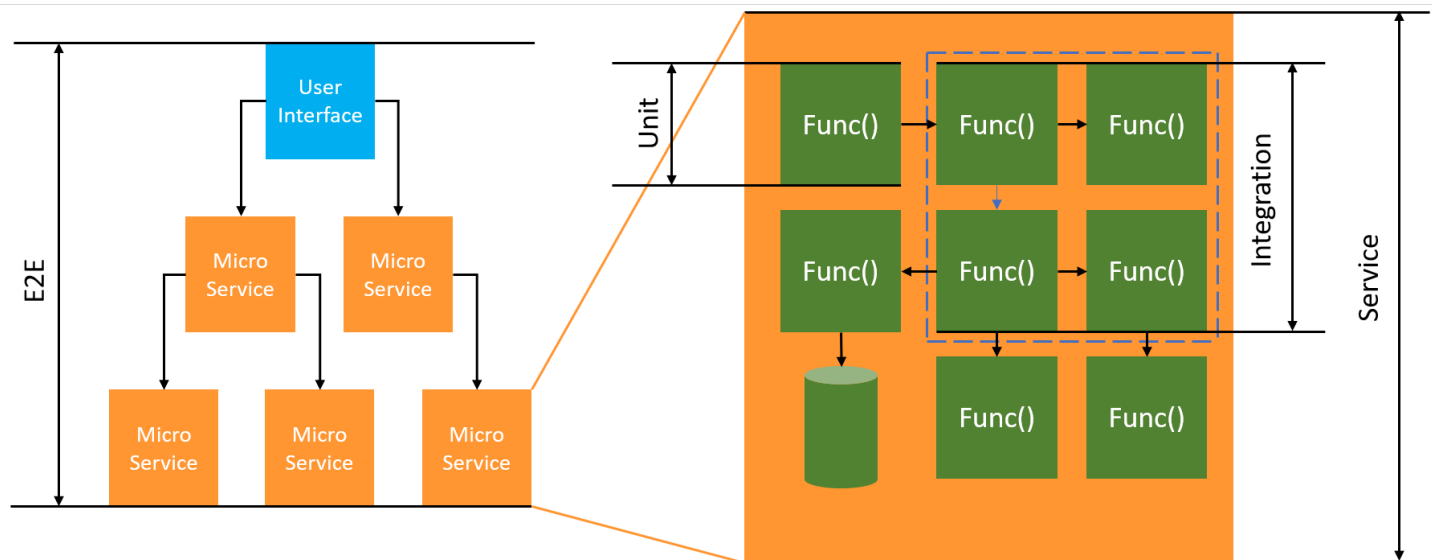
Il test delle funzioni serverless utilizza tipi e tecniche di test tradizionali, ma è necessario considerare anche il test delle applicazioni serverless nel loro insieme. Cloud-based i test forniscono la misura più accurata della qualità sia delle funzioni che delle applicazioni serverless.

Un'architettura applicativa serverless include servizi gestiti che forniscono funzionalità applicative critiche tramite chiamate API. Pertanto, è fondamentale che il ciclo di sviluppo preveda test automatici in grado di verificare il corretto funzionamento dell'interazione tra le funzioni e i servizi.

Se non crei test basati sul cloud, potresti riscontrare problemi dovuti alle differenze tra l'ambiente locale e quello implementato. Il processo di integrazione continua dovrebbe eseguire test su

un insieme di risorse con provisioning nel cloud prima di promuovere il codice nell'ambiente di implementazione successivo, come quello di controllo qualità (QA), staging o produzione.

Continua a leggere questa breve guida per scoprire le strategie di test per le applicazioni serverless oppure visita il [repository Serverless Test Samples](#) per approfondire esempi pratici, specifici per il linguaggio e il runtime selezionati.



Per i test serverless, è comunque necessario scrivere test unitari, di integrazione e end-to-end.

- **Test unitari:** vengono eseguiti su un blocco di codice isolato. Ad esempio, possono essere utilizzati per verificare la logica aziendale per calcolare le spese di spedizione in base a un articolo e a una destinazione specifici.
- **Test di integrazione:** coinvolgono due o più componenti o servizi che interagiscono, in genere in un ambiente cloud. Ad esempio, possono essere utilizzati per verificare che una funzione elabori gli eventi di una coda.
- **End-to-end test:** test che verificano il comportamento in un'intera applicazione. Ad esempio, possono essere utilizzati per verificare che l'infrastruttura sia configurata correttamente e che gli eventi fluiscano tra i servizi come previsto per registrare l'ordine di un cliente.

## Obiettivi aziendali specifici

Il test delle soluzioni serverless potrebbe richiedere più tempo per la configurazione. È necessario verificare le interazioni tra i servizi basate sugli eventi. Mentre leggi questa guida, tieni a mente questi motivi aziendali pratici:

- Aumenta la qualità della tua applicazione
- Riduci il tempo necessario per creare funzionalità e correggere bug

La qualità di un'applicazione dipende dal test di molti scenari. Considerate i vostri scenari aziendali e automatizzate i test da eseguire sui servizi cloud. Ciò aumenta la qualità della tua applicazione.

I bug e i problemi di configurazione costano meno se rilevati nelle prime fasi del ciclo di sviluppo. Problemi che passano inosservati fino a quando la produzione non richiede più impegno e più persone per essere risolti.

Una buona strategia di test serverless migliora la qualità del software e accelera le iterazioni. Verifica che le funzioni e le applicazioni Lambda funzionino come previsto nel cloud.

## Cosa testare

Consigliamo una strategia di test che verifichi i comportamenti dei servizi gestiti, la configurazione del cloud, le politiche di sicurezza e l'integrazione con il codice. Il test comportamentale, noto anche come black box test, verifica che un sistema funzioni come previsto senza conoscerne i componenti interni.

- Esegui test unitari per verificare la logica aziendale all'interno delle funzioni Lambda.
- Verifica che i servizi integrati siano effettivamente richiamati e che i parametri di input siano corretti.
- Controlla che un evento utilizzi tutti i servizi previsti in un flusso di lavoro dall'inizio alla fine.

Nell'architettura tradizionale basata su server, i team spesso testano solo il codice che viene eseguito sul server delle applicazioni. Considerano gli altri componenti, servizi o dipendenze esterni e non compresi nell'ambito di applicazione.

Le applicazioni serverless sono costituite da piccole unità di lavoro. Gli esempi includono le funzioni Lambda che recuperano prodotti da un database, elaborano elementi da una coda o ridimensionano un'immagine in archivio. Ogni componente viene eseguito nel proprio ambiente. I team gestiscono molte di queste piccole unità all'interno di un'unica applicazione.

Alcune funzionalità possono essere gestite interamente da servizi gestiti come Amazon S3 o create senza codice personalizzato. Non è necessario testare questi servizi gestiti. Tuttavia, è necessario verificare in che modo il codice si integra con essi.

## Come testare le soluzioni serverless

Probabilmente sai come testare le applicazioni distribuite localmente. Si scrivono test sul codice sul desktop o all'interno di contenitori. Ad esempio, potresti chiamare un servizio web locale e quindi controllare la risposta.

Le soluzioni serverless utilizzano il codice funzione e i servizi gestiti basati sul cloud, come code, database, bus di eventi e sistemi di messaggistica. Questi componenti si connettono tramite un'architettura basata sugli eventi, in cui i messaggi, chiamati eventi, fluiscono da una risorsa all'altra. Alcune interazioni sono sincrone, come un servizio web che restituisce immediatamente i risultati.

Altre sono asincrone, come l'inserimento di elementi in una coda o l'avvio di una fase del flusso di lavoro. La tua strategia di test deve coprire entrambi i tipi e testare le interazioni tra i servizi. Per le interazioni asincrone, potrebbe essere necessario rilevare effetti collaterali nei componenti a valle che non sono immediatamente visibili.

Non è possibile replicare completamente un ambiente cloud a livello locale. Ciò include code, tabelle del database, bus di eventi e politiche di sicurezza. Le differenze tra ambienti locali e cloud causano problemi. Queste differenze aumentano il tempo necessario per la riproduzione e la correzione dei bug.

Nelle applicazioni serverless, i componenti sono presenti interamente nel cloud. I test rispetto al codice e ai servizi cloud sono necessari per sviluppare funzionalità e correggere i bug.

## Tecniche di test

La tua strategia di test probabilmente include un mix di tecniche. Si utilizzano test interattivi rapidi per eseguire il debug delle funzioni nella console. Si scrivono unit test automatici per verificare la logica aziendale. Le chiamate a servizi esterni vengono verificate con simulazioni. Puoi anche testare emulatori che imitano un servizio.

- [the section called “Test nel cloud”](#): distribuisce l'infrastruttura e il codice per testarli con servizi, politiche di sicurezza e configurazioni effettivi. Cloud-based i test forniscono la misura più accurata della qualità del codice.

Il debug di una funzione nella console è un modo rapido per eseguire test nel cloud. Puoi scegliere tra esempi di eventi di test o creare un evento personalizzato. Puoi anche condividere gli eventi di test con il tuo team tramite la console.

Per automatizzare i test nel ciclo di vita dello sviluppo e della compilazione, esegui i test all'esterno della console. Consulta le sezioni di test specifiche della lingua in questa guida per le strategie di automazione.

- [the section called “Test con mock”](#): i mock sono oggetti nel codice che simulano un servizio esterno. Forniscono un comportamento predefinito per verificare le chiamate e i parametri del servizio. Un falso è un falso che utilizza scorciatoie per semplificare o velocizzare i test. Ad esempio, un oggetto di accesso ai dati falso potrebbe restituire dati da un datastore in memoria. Le simulazioni possono semplificare le dipendenze complesse, ma potrebbero portare a un numero maggiore di simulazioni per sostituire le dipendenze annidate.
- [the section called “Test a livello locale utilizzando AWS SAM CLI”](#): utilizza la AWS SAM CLI per richiamare localmente le funzioni Lambda nei contenitori Docker che utilizzano lo stesso ambiente di runtime di Lambda. AWS Puoi testare la logica delle funzioni e l'elaborazione degli eventi senza implementarli nel cloud.
- [the section called “Test con emulazione”](#): Usa l'[LocalStack integrazione in VS Code](#) per emulare più AWS servizi localmente per testare le integrazioni dei servizi.

## Test nel cloud

I test nel cloud sono utili per tutte le fasi dei test: test unitari, test di integrazione e test end-to-end. I test eseguiti su codice e servizi basati su cloud forniscono la misura più accurata della qualità del codice.

Un modo semplice per eseguire una funzione Lambda nel cloud è con un evento di test nella Console di gestione AWS. Un evento di test è un input JSON per la funzione. Se la tua funzione non richiede input, l'evento può essere un documento JSON vuoto. ( {} ) La console fornisce eventi di esempio per molte integrazioni di servizi. Puoi condividere gli eventi con il tuo team per semplificare i test.

Scopri come eseguire il [debug di una funzione di esempio nella console](#).

### Note

Sebbene l'esecuzione delle funzioni nella console sia un modo rapido per eseguire il debug, l'automazione dei cicli di test è essenziale per aumentare la qualità delle applicazioni e la velocità di sviluppo.

Gli esempi di automazione dei test sono disponibili nel [repository Serverless Test Samples](#). La seguente linea di comando esegue un [esempio di test di integrazione Python](#) automatizzato:

```
python -m pytest -s tests/integration -v
```

Sebbene il test venga eseguito localmente, interagisce con risorse basate sul cloud. Queste risorse sono state distribuite utilizzando lo strumento da riga di AWS SAM comando AWS Serverless Application Model and. Il codice di test recupera innanzitutto gli output dello stack distribuito, come l'endpoint API, la funzione ARN e il ruolo di sicurezza.

Quindi, invia una richiesta all'endpoint API. La risposta contiene un elenco di bucket Amazon S3. Questo test viene eseguito su risorse basate sul cloud per verificare che siano distribuite, protette e funzionanti.

```
===== test session starts =====
platform darwin -- Python 3.10.10, pytest-7.3.1, pluggy-1.0.0
-- /Users/t/code/aws/serverless-test-samples/python-test-samples/apigw-lambda/
venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/t/code/aws/serverless-test-samples/python-test-samples/apigw-
lambda
plugins: mock-3.10.0
collected 1 item

tests/integration/test_api_gateway.py::TestApiGateway::test_api_gateway

--> Stack outputs:

HelloWorldApi
= https://p7teqs3162.execute-api.us-east-2.amazonaws.com/Prod/hello/
> API Gateway endpoint URL for Prod stage for Hello World function

PythonTestDemo
= arn:aws:lambda:us-east-2:123456789012:function:testing-apigw-lambda-
PythonTestDemo-iSij8evaTdxl
> Hello World Lambda Function ARN

PythonTestDemoIamRole
= arn:aws:iam::123456789012:role/testing-apigw-lambda-PythonTestDemoRole-
IZELQQ9MG4HQ
> Implicit IAM Role created for Hello World function
```

```
--> Found API endpoint for "testing-apigw-lambda" stack...
--> https://p7teqs3162.execute-api.us-east-2.amazonaws.com/Prod/hello/
API Gateway response:
amplify-dev-123456789-deployment|myapp-prod-p-loggingbucket-123456|s3-java-
bucket-123456789
PASSED

===== 1 passed in 1.53s =====
```

I test nel cloud offrono i seguenti vantaggi per lo sviluppo di applicazioni native del cloud:

- Puoi testare ogni servizio disponibile.
- Utilizzi sempre le API del servizio e i valori restituiti più recenti.
- Un ambiente di test cloud somiglia molto al tuo ambiente di produzione.
- I test possono verificare policy di sicurezza, Service Quotas, configurazioni e parametri specifici dell'infrastruttura.
- Ogni sviluppatore può creare rapidamente uno o più ambienti di test nel cloud.
- I test sul cloud aumentano la sicurezza che il codice venga eseguito correttamente in produzione.

I test nel cloud presentano alcuni aspetti negativi. Le implementazioni cloud richiedono in genere più tempo rispetto alle distribuzioni desktop locali.

Strumenti come [AWS Serverless Application Model \(AWS SAM\) Accelerate](#), [AWS Cloud Development Kit \(AWS CDK\) watch mode](#) e [SST](#) (di terze parti) riducono questa latenza.

Questi strumenti monitorano l'infrastruttura e il codice, quindi implementano automaticamente gli aggiornamenti nell'ambiente cloud.

#### Note

Scopri come [creare l'infrastruttura come codice](#) nella [Serverless Developer Guide](#) per saperne di più su AWS Serverless Application Model, CloudFormation, e. Kit di sviluppo per il cloud AWS (AWS CDK)

A differenza dei test locali, i test sul cloud utilizzano risorse che potrebbero comportare costi. Gli ambienti di test isolati potrebbero aggiungere lavoro ai tuoi DevOps team, specialmente nelle organizzazioni con severi controlli sugli account. Tuttavia, il tempo dedicato agli sviluppatori per

configurare un ambiente locale complesso può costare di più rispetto all'utilizzo di ambienti cloud usa e getta creati con strumenti Infrastructure as Code.

I test nel cloud, anche alla luce di queste considerazioni, restano il modo migliore per garantire la qualità delle soluzioni serverless.

## Test con mock

Il test con mock è una tecnica in cui crei oggetti sostitutivi nel tuo codice per simulare il comportamento di un servizio cloud.

Ad esempio, potresti scrivere un test che utilizzi una simulazione del servizio Amazon S3. Il mock restituisce una risposta impostata ogni volta che viene chiamato il `CreateObject` metodo. Non chiama Amazon S3 o altri endpoint di servizio.

I framework fittizi spesso generano oggetti fittizi per te. Alcuni framework sono generici. Altri hanno come target gli AWS SDK, come [Moto](#), una libreria Python per simulare servizi. AWS

Gli oggetti fittizi differiscono dagli emulatori. Gli sviluppatori creano dei mock come parte del codice di test. Gli emulatori sono applicazioni autonome che presentano le stesse funzionalità dei sistemi che imitano.

Di seguito puoi trovare alcuni vantaggi legati all'utilizzo dei mock:

- I mock possono simulare servizi di terze parti che sfuggono al controllo dell'applicazione, come API e provider di software as a service (SaaS), senza bisogno di accedere direttamente a tali servizi.
- I mock sono utili per testare le condizioni di errore, soprattutto quando tali condizioni sono difficili da simulare, come un'interruzione del servizio.
- Una volta configurato, il mock consente di eseguire rapidamente test locali.
- I mock possono fornire un comportamento sostitutivo praticamente per qualsiasi tipo di oggetto, quindi le strategie di simulazione possono riguardare una più ampia varietà di servizi rispetto agli emulatori.
- Quando diventano disponibili nuove funzionalità o comportamenti, i test con mock possono reagire più rapidamente. Utilizzando un framework fittizio generico, puoi simulare nuove funzionalità non appena l'SDK aggiornato sarà disponibile. AWS

I test con mock presentano i seguenti svantaggi:



- I mock richiedono in genere una quantità non trascurabile di installazione e configurazione, in particolare quando si cerca di determinare i valori restituiti da diversi servizi per simulare correttamente le risposte.
- I mock devono essere scritti, configurati e gestiti dagli sviluppatori, aumentando le loro responsabilità.
- Potrebbe essere necessario avere accesso al cloud per comprendere le API e restituire i valori dei servizi.
- I mock possono essere difficili da mantenere. Quando le firme delle API cloud fittizie cambiano o gli schemi dei valori restituiti evolvono, è necessario aggiornare i mock. I mock richiedono aggiornamenti anche se si estende la logica dell'applicazione per effettuare chiamate a nuove API.
- I test che utilizzano mock potrebbero avere un esito positivo negli ambienti desktop ma fallire nel cloud. I risultati potrebbero non corrispondere all'API corrente. La configurazione e le quote del servizio non possono essere testate.
- I framework fittizi sono limitati nel testare o rilevare le politiche di AWS Identity and Access Management (IAM) o le limitazioni delle quote. Sebbene i mock siano più efficaci nella simulazione quando l'autorizzazione fallisce o viene superata una quota, i test non possono determinare quale risultato si verifichi effettivamente in un ambiente di produzione.

## Test a livello locale utilizzando AWS SAM CLI

AWS SAMCLIUsalo per [testare le tue funzioni nei contenitori Docker](#) utilizzando lo stesso ambiente di runtime di AWS Lambda. Puoi testare la logica delle funzioni e l'elaborazione degli eventi localmente senza implementarli nel cloud. Se la tua funzione effettua chiamate API ad altre Servizi AWS, tali chiamate raggiungono AWS risorse reali.

I vantaggi del test con contenitori locali includono quanto segue:

- Utilizza ambienti AWS Lambda di runtime per test accurati.
- Consente rapide iterazioni di sviluppo locale senza implementazione nel cloud.
- Supporta il debug con strumenti di sviluppo locale familiari.

I test con contenitori locali presentano le seguenti limitazioni:

- Servizio AWS le chiamate provenienti dalla vostra funzione interagiscono con AWS risorse reali, il che potrebbe comportare costi e influire sui dati di produzione.
- Richiede l'installazione e l'esecuzione di Docker localmente.

## Test con emulazione

Gli emulatori sono applicazioni eseguite localmente che simulano AWS i servizi fornendo API e valori di ritorno simili. LocalStack è un popolare strumento di emulazione che fornisce un ambiente di sviluppo locale completo per testare le integrazioni dei servizi.

LocalStack è un emulatore AWS cloud che puoi utilizzare per testare le applicazioni serverless a livello locale. Puoi testare le funzioni Lambda che si integrano con servizi come DynamoDB, Amazon S3 e Amazon SQS senza connetterti a servizi effettivi. AWS Puoi utilizzarlo [LocalStack nel Toolkit for VS Code AWS](#).

I vantaggi dei test con gli emulatori sono molteplici:

- Gli emulatori possono aiutare a velocizzare le iterazioni e i test di sviluppo locali.
- Gli emulatori offrono un ambiente familiare per gli sviluppatori abituati a sviluppare codice in un ambiente locale. Ad esempio, se hai familiarità con lo sviluppo di un'applicazione di livello n, potresti avere un motore di database e un server Web, simili a quelli in esecuzione in produzione, in esecuzione sul tuo computer locale per fornire funzionalità di test rapide, locali e isolate.
- Gli emulatori non richiedono alcuna modifica all'infrastruttura cloud (come gli account cloud per sviluppatori), quindi sono facili da implementare con i modelli di test esistenti.
- Poiché gli emulatori non utilizzano AWS risorse effettive, non ti verranno addebitati costi imprevisti quando avvii più servizi o lasci funzionare alcune risorse per lunghi periodi di tempo.

I test con emulatori presentano i seguenti svantaggi:

- Gli emulatori possono essere difficili da configurare e replicare, specialmente se utilizzati nelle pipeline. CI/CD Ciò può contribuire ad aumentare il carico di lavoro del personale IT o degli sviluppatori che gestiscono il proprio software.
- Le funzionalità e le API emulate sono in genere in ritardo rispetto agli aggiornamenti del servizio. Ciò può causare errori perché il codice testato non corrisponde all'API effettiva e impedisce l'adozione di nuove funzionalità.
- Gli emulatori richiedono miglioramenti in termini di supporto, aggiornamenti, correzioni di bug e parità delle funzionalità. Questi sono di responsabilità dell'autore dell'emulatore, che potrebbe essere una società terza.
- I test che si basano sugli emulatori possono fornire risultati positivi a livello locale, ma fallire nel cloud a causa delle politiche di sicurezza della produzione, delle configurazioni tra servizi o del superamento delle quote Lambda.

## Best practice

Le sezioni seguenti forniscono consigli per testare correttamente le applicazioni serverless.

Puoi trovare esempi pratici di test e automazione dei test nel [repository Serverless Test Samples](#).

### Dai priorità ai test nel cloud

I test nel cloud sono un'opzione molto valida per garantire la copertura dei test più affidabile, accurata e completa. L'esecuzione dei test nel contesto del cloud verifica in modo completo non solo la logica aziendale ma anche le politiche di sicurezza, le configurazioni dei servizi, le quote e le firme API e i valori restituiti più aggiornati.

### Struttura il tuo codice per la testabilità

Semplifica i test e le funzioni Lambda separando il codice dalla logica aziendale principale. Lambda-specific

Il gestore delle funzioni Lambda deve essere un adattatore agile in grado di acquisire i dati degli eventi e trasmettere solo i dettagli importanti ai metodi della logica aziendale. Con questa strategia, puoi eseguire test completi sulla tua logica aziendale senza preoccuparti dei dettagli. Lambda-specific  
Le tue funzioni AWS Lambda non dovrebbero richiedere la configurazione di un ambiente complesso o una grande quantità di dipendenze per creare e inizializzare il componente in fase di test.

In generale, è consigliabile scrivere un gestore che estrae e convalida i dati dagli eventi e dagli oggetti di contesto in entrata, per poi inviare tale input ai metodi che eseguono la logica aziendale.

### Accelera i cicli di feedback sullo sviluppo

Esistono strumenti e tecniche per accelerare i cicli di feedback sullo sviluppo. Ad esempio, [AWS SAM Accelerate](#) e [AWS CDK watch mode](#) riducono entrambi il tempo necessario per aggiornare gli ambienti cloud.

Gli esempi nel repository GitHub [Serverless Test Samples](#) esplorano alcune di queste tecniche.

Inoltre, ti consigliamo di creare e testare le risorse cloud il prima possibile durante lo sviluppo, anziché solo dopo aver controllato il codice sorgente. Questa pratica consente di accelerare l'esplorazione e la sperimentazione durante lo sviluppo di soluzioni. Inoltre, l'automazione dell'implementazione da una macchina di sviluppo ti aiuta a scoprire i problemi di configurazione del cloud più rapidamente e riduce gli sprechi di tempo per gli aggiornamenti e i processi di revisione del codice.

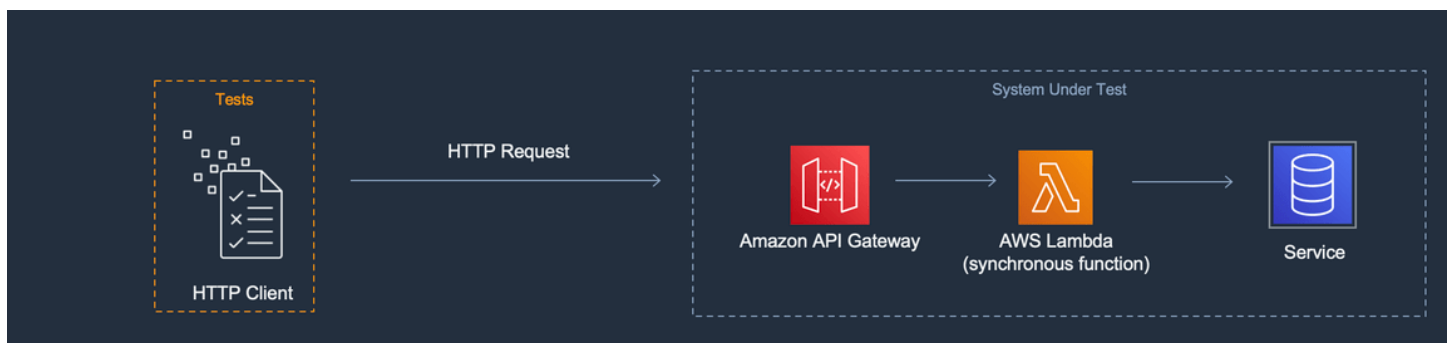
## Concentrati sui test di integrazione

Quando si sviluppano applicazioni con Lambda, una buona pratica è quella di testare i componenti insieme.

I test eseguiti su due o più componenti architetturici sono chiamati test di integrazione. L'obiettivo dei test di integrazione è comprendere non solo come viene eseguito il codice tra i componenti, ma anche come si comporta l'ambiente che ospita il codice. End-to-end i test sono tipi speciali di test di integrazione che verificano i comportamenti in un'intera applicazione.

Per creare test di integrazione, implementa la tua applicazione in un ambiente cloud. Questa operazione può essere eseguita da un ambiente locale o tramite una CI/CD pipeline. Quindi, scrivi dei test per esercitare il sistema sottoposto a test (SUT) e convalidare il comportamento previsto.

Ad esempio, il sistema sottoposto a test potrebbe essere un'applicazione che utilizza Gateway API, Lambda e DynamoDB. Un test potrebbe effettuare una chiamata HTTP sintetica a un endpoint API Gateway e verificare che la risposta includa il payload previsto. Questo test verifica che il codice AWS Lambda sia corretto e che ogni servizio sia configurato correttamente per gestire la richiesta, comprese le autorizzazioni IAM tra di essi. Inoltre, è possibile progettare un test in cui vengano scritti record di varie dimensioni per verificare che le Service Quotas, come la dimensione massima dei record in DynamoDB, siano impostate correttamente.



## Crea ambienti di test isolati

In genere, i test nel cloud richiedono ambienti di sviluppo isolati, in modo che test, dati ed eventi non si sovrappongano.

Un approccio consiste nel fornire a ogni sviluppatore un account dedicato. AWS In questo modo si evitano conflitti con la denominazione delle risorse che possono verificarsi quando più sviluppatori lavorano in una base di codice condivisa, tentano di distribuire risorse o richiamare un'API.

I processi di test automatizzati dovrebbero creare risorse con un nome univoco per ogni stack. Ad esempio, puoi impostare script o file di configurazione TOML in modo che i [comandi AWS SAM CLI `aws sam deploy` o `aws sam sync`](#) specifichino automaticamente uno stack con un prefisso univoco.

In alcuni casi, gli sviluppatori condividono un account. AWS Ciò potrebbe essere dovuto alla presenza di risorse nello stack costose da utilizzare o da fornire e configurare. Ad esempio, un database potrebbe essere condiviso per semplificare l'impostazione e il corretto seeding dei dati.

Se gli sviluppatori condividono un account, è necessario stabilire dei limiti per identificare la proprietà ed eliminare le sovrapposizioni. Un modo per farlo è anteporre ai nomi degli stack gli ID utente degli sviluppatori. Un altro approccio comune è quello di configurare stack basati su rami di codice. Grazie ai confini dei rami, gli ambienti sono isolati, ma gli sviluppatori possono comunque condividere risorse, come un database relazionale. Questo approccio è una procedura consigliata quando gli sviluppatori lavorano su più di un ramo alla volta.

I test nel cloud sono utili per tutte le fasi di test, inclusi i test unitari, i test di integrazione e i test end-to-end. Mantenere un isolamento adeguato è essenziale, ma è comunque necessario che l'ambiente di controllo qualità (QA) assomigli il più possibile all'ambiente di produzione. Per questo motivo, i team aggiungono processi di controllo delle modifiche per gli ambienti QA.

Generalmente, per gli ambienti di preproduzione e produzione vengono fissati dei limiti a livello di account per isolare i carichi di lavoro dai problemi di tipo *noisy neighbor* e implementare controlli di sicurezza con privilegi minimi per proteggere i dati sensibili. I carichi di lavoro hanno delle quote assegnate. Non è auspicabile che i test consumino le quote allocate per la produzione (rischio di effetto *noisy neighbor*) o che abbiano accesso ai dati dei clienti. I test di carico sono un'altra attività da isolare dallo stack di produzione.

In tutti i casi, gli ambienti devono essere configurati con avvisi e controlli per evitare spese inutili. Ad esempio, è possibile limitare il tipo, il livello o la dimensione delle risorse che possono essere create e impostare avvisi e-mail quando i costi stimati superano una determinata soglia.

## Usa i mock per una logica aziendale isolata

I framework fittizi sono uno strumento prezioso per scrivere test unitari rapidi. Questi sono particolarmente utili quando i test riguardano logiche aziendali interne complesse, come calcoli o simulazioni matematiche o finanziarie. Cerca test unitari che prevedono un gran numero di casi di test o varianti di input, in cui tali input non modificano lo schema o il contenuto delle chiamate ad altri servizi cloud.

Il codice verificato dai test unitari con mock dovrebbe essere verificato anche dai test nel cloud. Questa operazione è consigliata perché l'ambiente su un laptop per sviluppatori o una macchina di compilazione potrebbe essere configurato in modo diverso rispetto a un ambiente di produzione nel cloud. Ad esempio, le funzioni Lambda potrebbero utilizzare più memoria o impiegare più tempo rispetto a quanto allocato quando vengono eseguite con determinati parametri di input. Oppure il codice potrebbe includere variabili di ambiente non configurate o che non sono state configurate allo stesso modo, e le differenze potrebbero causare un comportamento diverso o un errore del codice.

I vantaggi dei mock sono minori per i test di integrazione, poiché il livello di impegno necessario per implementare i mock necessari aumenta con il numero di punti di connessione. End-to-end i test non dovrebbero utilizzare simulazioni, poiché questi test generalmente si occupano di stati e logiche complesse che non possono essere facilmente simulati con framework fittizi.

Infine, evita di utilizzare servizi cloud fittizi per verificare la corretta implementazione delle chiamate del servizio. Al contrario, per convalidare l'implementazione a livello di comportamento, configurazione e funzionalità, è consigliabile effettuare le chiamate ai servizi cloud direttamente nell'ambiente cloud.

## Utilizza gli emulatori con parsimonia

Gli emulatori possono essere utili in alcuni casi d'uso, ad esempio per un team di sviluppo con accesso a Internet limitato, inaffidabile o lento. Tuttavia, nella maggior parte dei casi, assicurati di usare gli emulatori con parsimonia.

Evitando gli emulatori, puoi creare e innovare con le funzionalità di servizio più recenti e le API aggiornate. Non resterai bloccato ad aspettare le versioni dei fornitori per raggiungere la parità di funzionalità. Riduci le spese iniziali e continue per l'acquisto e la configurazione su più sistemi di sviluppo e macchine di costruzione. Inoltre, si evita il problema che molti servizi cloud semplicemente non dispongono di emulatori. Una strategia di test che dipende dall'emulazione rende impossibile utilizzare tali servizi (portando a soluzioni alternative potenzialmente più costose) o produrre codice e configurazioni non ben testati.

Quando si utilizza l'emulazione per i test, è comunque necessario eseguire il test nel cloud per verificare la configurazione e testare le interazioni con i servizi cloud che possono essere simulate o ricorrere all'utilizzo di mock solo in un ambiente emulato.

## Problematiche legate ai test locali

Quando utilizzate emulatori e chiamate simulate per eseguire i test sul desktop locale, potreste riscontrare delle incongruenze nei test man mano che il codice progredisce da un ambiente all'altro

nella pipeline. CI/CD I test unitari per convalidare la logica di business dell'applicazione sul desktop potrebbero non testare accuratamente gli aspetti critici dei servizi cloud.

Gli esempi seguenti forniscono alcuni casi a cui prestare attenzione quando si eseguono test localmente con mock ed emulatori:

### Esempio: la funzione Lambda crea un bucket S3

Se la logica di una funzione Lambda dipende dalla creazione di un bucket S3, un test completo dovrebbe confermare che Amazon S3 è stato chiamato e che il bucket è stato creato correttamente.

- In una configurazione di test con mock, potresti simulare una risposta con esito positivo e potenzialmente aggiungere un caso di test per gestire una risposta non esito negativo.
- In uno scenario di test di emulazione, l'CreateBucketAPI potrebbe essere chiamata, ma devi sapere che l'identità che effettua la chiamata locale non proviene dal servizio Lambda. L'identità chiamante non assume un ruolo di sicurezza come nel cloud, quindi viene utilizzata un'autenticazione segnaposto, possibilmente con un ruolo o un'identità utente più permissivi che è diversa se eseguita nel cloud.

Le configurazioni di simulazione ed emulazione verificano cosa fa la funzione Lambda se chiama Amazon S3; tuttavia, tali test non verificano che la funzione Lambda, così come configurata, sia in grado di creare correttamente il bucket Amazon S3. È necessario assicurarsi che al ruolo assegnato alla funzione sia associata una policy di sicurezza che consenta alla funzione di eseguire l'operazione `s3:CreateBucket`. In caso contrario, è probabile che la funzione fallisca quando viene distribuita in un ambiente cloud.

### Esempio: una funzione Lambda elabora i messaggi da una coda Amazon SQS

Se una coda Amazon SQS è l'origine di una funzione Lambda, un test completo deve verificare che quando un messaggio viene inserito in una coda la funzione Lambda venga richiamata correttamente.

I test di emulazione e i test simulati sono generalmente impostati per eseguire direttamente il codice della funzione Lambda e per simulare l'integrazione di Amazon SQS passando un payload di eventi JSON (o un oggetto deserializzato) come input del gestore della funzione.

I test locali che simulano l'integrazione di Amazon SQS verificano cosa fa la funzione Lambda quando viene richiamata da Amazon SQS con un determinato payload, ma non verifica che Amazon SQS richiami correttamente la funzione Lambda quando viene distribuita in un ambiente cloud.

Di seguito troverai alcuni esempi di problemi di configurazione che potresti riscontrare con Amazon SQS e Lambda:

- Il timeout di visibilità di Amazon SQS è troppo basso e comporta più chiamate quando ne era prevista una sola.
- Il ruolo di esecuzione della funzione Lambda non consente la lettura dei messaggi dalla coda (tramite `sqs:ReceiveMessage`, `sqs:DeleteMessage` o `sqs:GetQueueAttributes`).
- L'evento di esempio passato alla funzione Lambda supera la quota relativa alla dimensione dei messaggi di Amazon SQS. Pertanto, il test non è valido perché Amazon SQS non sarebbe mai in grado di inviare un messaggio di tali dimensioni.

Come mostrano questi esempi, con molta probabilità si otterranno risultati inattendibili dai test che riguardano la logica aziendale ma non le configurazioni tra i servizi cloud.

## Domande frequenti

Ho una funzione Lambda che esegue calcoli e restituisce un risultato senza chiamare altri servizi. Devo davvero testarla nel cloud?

Sì. Le funzioni Lambda hanno parametri di configurazione che possono modificare l'esito del test. Tutto il codice della funzione Lambda dipende dalle impostazioni di [timeout](#) e [memoria](#), e se tali aspetti non sono impostati correttamente potrebbero causare errori della funzione.

Le policy Lambda consentono inoltre la registrazione standard dell'output su Amazon. [CloudWatch](#). Anche se il codice non chiama CloudWatch direttamente, è necessaria l'autorizzazione per abilitare la registrazione. Questa autorizzazione richiesta non può essere simulata o emulata con precisione.

In che modo i test nel cloud possono contribuire ai test unitari? Se è nel cloud e si connette ad altre risorse, non è un test di integrazione?

Definiamo i test unitari come test che operano su componenti architettonici in modo isolato, ma ciò non impedisce ai test di includere componenti che potrebbero chiamare altri servizi o utilizzare alcune comunicazioni di rete.

Molte applicazioni serverless dispongono di componenti architettonici che possono essere testati in modo isolato, anche nel cloud. Un esempio può essere quello di una funzione Lambda che accetta l'input, elabora i dati e invia un messaggio a una coda Amazon SQS. Un test unitario di questa funzione verificherebbe probabilmente se i valori di input determinano la presenza di determinati valori nel messaggio in coda.



Prendi in considerazione un test scritto utilizzando lo schema Arrange, Act, Assert (organizzazione, azione, affermazione):

- Arrange: alloca le risorse (una coda per ricevere messaggi e la funzione sottoposta a test).
- Act: chiama la funzione sottoposta a test.
- Assert: recupera il messaggio inviato dalla funzione e convalida l'output.

Un approccio di test con mock implicherebbe la simulazione della coda con un oggetto fittizio in corso di elaborazione e la creazione di un'istanza durante il processo della classe o del modulo che contiene il codice della funzione Lambda. Durante la fase Assert, il messaggio in coda verrebbe recuperato dall'oggetto fittizio.

In un approccio basato sul cloud, il test creerebbe una coda Amazon SQS ai fini del test e implementerebbe la funzione Lambda con variabili di ambiente configurate per utilizzare la coda Amazon SQS isolata come destinazione di output. Dopo aver eseguito la funzione Lambda, il test recupererebbe il messaggio dalla coda di Amazon SQS.

Il test basato su cloud eseguirebbe lo stesso codice, affermerebbe lo stesso comportamento e convaliderebbe la correttezza funzionale dell'applicazione. Tuttavia, avrebbe l'ulteriore vantaggio di poter convalidare le impostazioni della funzione Lambda: il ruolo IAM, le politiche IAM e le impostazioni di timeout e memoria della funzione.

## Risorse e passaggi successivi

Utilizza le seguenti risorse per ottenere ulteriori informazioni ed esplorare esempi pratici di test.

### Implementazioni di esempio

Il repository [Serverless Test Samples](#) su GitHub contiene esempi concreti di test che seguono i modelli e le migliori pratiche descritti in questa guida. Il repository contiene codice di esempio e procedure dettagliate dei processi di test con mock, emulazione e cloud descritti nelle sezioni precedenti. Usa questo repository per essere aggiornato sulle ultime linee guida sui test serverless di AWS.

### Approfondimenti

Visita [Serverless Land](#) per accedere ai blog, ai video e ai corsi di formazione più recenti sulle tecnologie serverless. AWS

Si consiglia inoltre la lettura dei seguenti post del AWS blog:

- [Accelerare lo sviluppo serverless con AWS SAM Accelerate](#) (AWS post sul blog)
- [Aumentare la velocità di sviluppo con CDK Watch](#) (AWS post sul blog)
- [Simulazioni delle integrazioni dei servizi con AWS Step Functions](#) Local (post sul blog)AWS
- [Guida introduttiva al test di applicazioni](#) serverless (post sul blog)AWS

## Strumenti

- AWS SAM — [Test e debug di applicazioni serverless](#)
- AWS SAM — [Integrazione con test automatici](#)
- Lambda: [Test delle funzioni Lambda nella console Lambda](#)

## Gestione delle autorizzazioni in AWS Lambda

È possibile utilizzare AWS Identity and Access Management (IAM) per gestire le autorizzazioni in. AWS Lambda Esistono due categorie principali di autorizzazioni da considerare quando si lavora con le funzioni Lambda:

- Autorizzazioni necessarie alle tue funzioni Lambda per eseguire azioni API e accedere ad altre risorse AWS
- Autorizzazioni necessarie ad altri AWS utenti ed entità per accedere alle tue funzioni Lambda. Per ulteriori informazioni, consulta [the section called “Autorizzazioni di accesso \(autorizzazioni per altre entità di accedere alle tue funzioni\)”](#).

Le funzioni Lambda spesso devono accedere ad altre AWS risorse ed eseguire varie operazioni API su tali risorse. Ad esempio, è possibile che tu disponga di una funzione Lambda che risponde a un evento aggiornando le voci in un database Amazon DynamoDB. In questo caso, la funzione necessita delle autorizzazioni per accedere al database, nonché delle autorizzazioni per inserire o aggiornare elementi in quel database.

Definisci le autorizzazioni di cui ha bisogno la tua funzione Lambda in un ruolo IAM speciale chiamato [ruolo di esecuzione](#). In questo ruolo, puoi allegare una policy che definisce tutte le autorizzazioni necessarie alla tua funzione per accedere ad altre AWS risorse e leggere da fonti di eventi. Ogni funzione Lambda deve avere un ruolo di esecuzione. Come minimo, il tuo ruolo di esecuzione deve avere accesso ad Amazon CloudWatch perché le funzioni Lambda accedono ai CloudWatch log per impostazione predefinita. Puoi collegare la [policy gestita da AWSLambdaBasicExecutionRole](#) al tuo ruolo di esecuzione per soddisfare questo requisito.

Per concedere Account AWS ad altre organizzazioni e servizi le autorizzazioni per accedere alle tue risorse Lambda, hai alcune opzioni:

- Puoi utilizzare politiche [basate sull'identità per concedere](#) ad altri utenti l'accesso alle tue risorse Lambda. Identity-based le policy possono essere applicate direttamente agli utenti o a gruppi e ruoli associati a un utente.
- Puoi utilizzare politiche [basate sulle risorse](#) per fornire altri account e Servizi AWS autorizzazioni per accedere alle tue risorse Lambda. Quando un utente prova ad accedere a una risorsa Lambda, Lambda considera sia le policy basate su identità dell'utente che la policy basata sulla risorsa della risorsa. Quando un AWS servizio come Amazon Simple Storage Service (Amazon S3) chiama la tua funzione Lambda, Lambda considera solo la policy basata sulle risorse.
- Puoi utilizzare un modello [controllo degli accessi basato su attributi \(ABAC\)](#), per controllare l'accesso alle funzioni Lambda. Con ABAC, puoi collegare i tag a una funzione Lambda, inviarli in determinate richieste API o collegarli al principale IAM da cui proviene la richiesta. Per controllare gli accessi alla funzione, puoi invece specificare gli stessi tag nell'elemento di condizione di una policy IAM.

In AWS, è consigliabile concedere solo le autorizzazioni necessarie per eseguire un'attività (autorizzazioni con privilegi minimi). <https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege> Per implementarlo in Lambda, consigliamo di iniziare con una [policy gestita da AWS](#). Puoi utilizzare queste policy gestite così come sono o come punto di partenza per scrivere le tue policy più restrittive.

Per aiutarti a ottimizzare le autorizzazioni per l'accesso con privilegi minimi, Lambda fornisce alcune condizioni aggiuntive che puoi includere nelle tue policy. Per ulteriori informazioni, consulta [the section called “Risorse e condizioni”](#).

Per ulteriori informazioni su IAM, consulta la [Guida per l'utente di IAM](#).

## Definizione delle autorizzazioni della funzione Lambda con un ruolo di esecuzione

Il ruolo di esecuzione di una funzione Lambda è un ruolo AWS Identity and Access Management (IAM) che concede alla funzione l'autorizzazione all'accesso Servizi AWS e alle risorse. Ad esempio, potresti creare un ruolo di esecuzione autorizzato a inviare log ad Amazon CloudWatch e caricare dati di traccia su AWS X-Ray. Questa pagina fornisce informazioni su come creare, visualizzare e gestire il ruolo di esecuzione di una funzione Lambda.

Lambda assume automaticamente il tuo ruolo di esecuzione quando richiami la tua funzione. Dovresti evitare di chiamare `sts:AssumeRole` manualmente per assumere il ruolo di esecuzione nel codice della funzione. Se il tuo caso d'uso richiede che il ruolo venga assunto autonomamente, dovrai includere il ruolo stesso come principale attendibile nella policy di attendibilità del ruolo. Per ulteriori informazioni su come modificare una policy di attendibilità del ruolo, consulta [Modifica di una policy di attendibilità del ruolo \(console\)](#) nella Guida per l'utente di IAM.

Affinché Lambda possa assumere correttamente il ruolo di esecuzione, la [policy di attendibilità](#) del ruolo deve specificare il principale del servizio Lambda (`lambda.amazonaws.com`) come servizio attendibile.

### Argomenti

- [Creazione di un ruolo di esecuzione nella console di IAM](#)
- [Creazione e gestione dei ruoli con AWS CLI](#)
- [Garantisci l'accesso minimo ai privilegi per il tuo ruolo di esecuzione Lambda](#)
- [Visualizzazione e aggiornamento delle autorizzazioni nel ruolo di esecuzione](#)
- [Utilizzo delle politiche AWS gestite nel ruolo di esecuzione](#)
- [Utilizzo dell'ARN della funzione di origine per controllare il comportamento di accesso alla funzione](#)

## Creazione di un ruolo di esecuzione nella console di IAM

Per impostazione predefinita, Lambda crea un ruolo di esecuzione con autorizzazioni minime quando si [crea una funzione nella console Lambda](#). In particolare, questo ruolo di esecuzione include la policy [AWSLambdaBasicExecutionRole gestita](#), che conferisce alla tua funzione le autorizzazioni di base per registrare gli eventi su Amazon CloudWatch Logs. Puoi selezionare **Crea ruolo predefinito** nella sezione **Autorizzazioni**.

Puoi scegliere un ruolo esistente selezionando `Usa un altro ruolo` nella sezione `Autorizzazioni`. Se la tua funzione Lambda necessita di autorizzazioni aggiuntive per eseguire attività come l'aggiornamento delle voci in un database Amazon DynamoDB in risposta agli eventi, puoi creare un ruolo di esecuzione personalizzato con le autorizzazioni necessarie. Per fare ciò, seleziona `Usa un altro ruolo` nella sezione `Autorizzazioni`, che apre un cassetto in cui puoi personalizzare le tue autorizzazioni.

#### Note

Se il role manager è abilitato nel tuo account, la console Lambda ti assegna un ruolo di esecuzione. Per personalizzare le autorizzazioni, crea la tua funzione, quindi modifica le autorizzazioni della funzione per selezionare un ruolo diverso o personalizzare il ruolo associato. Per ulteriori informazioni, consulta [Creare ruoli automaticamente con role manager](#) nella IAM User Guide.

Per configurare un ruolo di esecuzione dalla Console

1. Inserisci un `nome` per il ruolo nella sezione `Dettagli del ruolo`.
2. Nella sezione `Politica`, seleziona `Usa la politica esistente`.
3. Seleziona le politiche AWS gestite che desideri allegare al tuo ruolo. Ad esempio, se la tua funzione deve accedere a DynamoDB, seleziona la policy `AWSLambdaDynamoDBExecutionRole` gestita.
4. Scegliere `Crea ruolo`.

In alternativa, quando [crei una funzione nella console Lambda](#), puoi allegare alla funzione qualsiasi ruolo di esecuzione precedentemente creato. Se desideri associare un nuovo ruolo di esecuzione a una funzione esistente, segui i passaggi in [Aggiornamento del ruolo](#) di esecuzione di una funzione.

## Creazione e gestione dei ruoli con AWS CLI

Per creare un ruolo di esecuzione con AWS Command Line Interface (AWS CLI), utilizzare il `create-role` comando. Quando utilizzi questo comando, puoi specificare la policy di attendibilità in linea. La policy di attendibilità di un ruolo garantisce ai principali specificati l'autorizzazione per assumere tale ruolo. Nell'esempio seguente concedi al principale del servizio Lambda l'autorizzazione per assumere il ruolo. Nota che i requisiti per l'escape delle virgolette nella stringa JSON possono variare a seconda della shell.

```
aws iam create-role \
  --role-name lambda-ex \
  --assume-role-policy-document '{"Version": "2012-10-17",      "Statement":
  [{ "Effect": "Allow", "Principal": {"Service": "lambda.amazonaws.com"}, "Action":
  "sts:AssumeRole"}]}'
```

Puoi inoltre definire la policy di attendibilità per il ruolo utilizzando un file JSON separato. Nell'esempio seguente, `trust-policy.json` è un file che si trova nella directory attuale.

Example `trust-policy.json`

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

```
aws iam create-role \
  --role-name lambda-ex \
  --assume-role-policy-document file://trust-policy.json
```

Utilizza il comando `attach-policy-to-role` per aggiungere le autorizzazioni al ruolo. Il seguente comando aggiunge la policy gestita da `AWSLambdaBasicExecutionRole` al ruolo di esecuzione `lambda-ex`.

```
aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole
```

Dopo aver creato il ruolo di esecuzione, collegalo alla funzione. Quando si [crea una funzione nella console Lambda](#), potrai collegare alla funzione qualsiasi ruolo di esecuzione creato in precedenza.

Se desideri associare un nuovo ruolo di esecuzione a una funzione esistente, segui i passaggi indicati in [the section called “Aggiornamento del ruolo di esecuzione di una funzione”](#).

## Garantisci l'accesso minimo ai privilegi per il tuo ruolo di esecuzione Lambda

Quando si crea per la prima volta un ruolo IAM per la funzione Lambda durante la fase di sviluppo, a volte è possibile concedere altre autorizzazioni oltre a quanto richiesto. Prima di pubblicare la funzione nell'ambiente di produzione, una best practice consiste nel modificare la policy in modo da includere solo le autorizzazioni richieste. Per ulteriori informazioni, consulta [Applica le autorizzazioni](#) con privilegi minimi nella IAM User Guide.

Utilizzare IAM Access Analyzer per identificare le autorizzazioni necessarie per la policy del ruolo di esecuzione IAM. IAM Access Analyzer esamina i AWS CloudTrail log nell'intervallo di date specificato e genera un modello di policy con solo le autorizzazioni utilizzate dalla funzione in quel periodo. È possibile utilizzare il modello per creare una policy gestita con autorizzazioni granulari e quindi collegarla al ruolo IAM. In questo modo, concedi solo le autorizzazioni necessarie al ruolo per interagire con le AWS risorse per il tuo caso d'uso specifico.

Per ulteriori informazioni, consulta [Generare policy basate sull'attività di accesso](#) nella IAM User Guide.

## Visualizzazione e aggiornamento delle autorizzazioni nel ruolo di esecuzione

Questo argomento spiega come visualizzare e aggiornare il [ruolo di esecuzione](#) della funzione.

### Argomenti

- [Visualizzazione del ruolo di esecuzione di una funzione](#)
- [Aggiornamento del ruolo di esecuzione di una funzione](#)

### Visualizzazione del ruolo di esecuzione di una funzione

Per visualizzare il ruolo di esecuzione di una funzione, usa la console Lambda.

Per visualizzare il ruolo di esecuzione di una funzione (console)

1. Aprire la pagina [Functions](#) (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Scegli Configuration (Configurazione), quindi Permissions (Autorizzazioni).

4. In Ruolo di esecuzione, puoi visualizzare il ruolo attualmente utilizzato come ruolo di esecuzione della funzione. Per comodità, puoi visualizzare tutte le risorse e le azioni a cui la funzione può accedere nella sezione Riepilogo delle risorse. Puoi anche scegliere un servizio dal menu a discesa per visualizzare tutte le autorizzazioni relative a tale servizio.

### Aggiornamento del ruolo di esecuzione di una funzione

È possibile aggiungere o rimuovere le autorizzazioni da un ruolo di esecuzione della funzione in qualsiasi momento, oppure configurare la funzione per l'utilizzo di un altro ruolo. Se la funzione richiede l'accesso a qualsiasi altro servizio o risorsa, è necessario aggiungere le autorizzazioni necessarie al ruolo di esecuzione.

Quando aggiungi le autorizzazioni alla funzione, aggiorni anche il relativo codice o la configurazione. Questa operazione forza l'arresto e la sostituzione delle istanze della funzione in esecuzione che hanno credenziali obsolete.

Per aggiornare il ruolo di esecuzione di una funzione, usa la console Lambda.

Per aggiornare il ruolo di esecuzione della funzione (console)

1. Aprire la pagina [Functions](#) (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Scegli Configuration (Configurazione), quindi Permissions (Autorizzazioni).
4. In Ruolo di esecuzione, scegli Modifica.
5. Se desideri aggiornare la tua funzione in modo da utilizzare un ruolo diverso come ruolo di esecuzione, scegli il nuovo ruolo nel menu a discesa sotto Ruolo esistente.

#### Note

Se desideri aggiornare le autorizzazioni all'interno di un ruolo di esecuzione esistente, puoi farlo solo nella console AWS Identity and Access Management (IAM).

Se desideri creare un nuovo ruolo da utilizzare come ruolo di esecuzione, scegli Crea un nuovo ruolo dai modelli di AWS policy in Ruolo di esecuzione. Quindi, inserisci un nome per il tuo nuovo ruolo in Nome ruolo e specifica le policy che desideri collegare al nuovo ruolo in Modelli di policy.

6. Seleziona Salva.



## Utilizzo delle politiche AWS gestite nel ruolo di esecuzione

Le seguenti politiche AWS gestite forniscono le autorizzazioni necessarie per utilizzare le funzionalità Lambda.

| Modifica                                                                                                                                  | Descrizione                                                                                                                                                                                                                            | Data             |
|-------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">AWSLambdaMSKExecutionRole</a> : Lambda ha aggiunto l'autorizzazione <a href="#">kafka:DescribeCluster V2</a> a questa policy. | AWSLambdaMSKExecutionRole concede le autorizzazioni per leggere e accedere ai record da un cluster Amazon Managed Streaming for Apache Kafka (Amazon MSK), gestire interfacce di ENIs rete elastiche () e scrivere nei log. CloudWatch | 17 giugno 2022   |
| <a href="#">AWSLambdaBasicExecutionRole</a> — Lambda ha iniziato a tracciare le modifiche a questa politica.                              | AWSLambdaBasicExecutionRole concede le autorizzazioni per caricare i log su CloudWatch.                                                                                                                                                | 14 febbraio 2022 |
| <a href="#">AWSLambdaDynamoDBExecutionRole</a> — Lambda ha iniziato a tenere traccia delle modifiche a questa politica.                   | AWSLambdaDynamoDBExecutionRole concede le autorizzazioni per leggere i record da un flusso Amazon DynamoDB e scrivere nei log. CloudWatch                                                                                              | 14 febbraio 2022 |
| <a href="#">AWSLambdaKinesisExecutionRole</a> — Lambda ha iniziato a tracciare le modifiche a questa politica.                            | AWSLambdaKinesisExecutionRole concede le autorizzazioni per leggere eventi da un flusso di dati Amazon Kinesis e scrivere su Logs. CloudWatch                                                                                          | 14 febbraio 2022 |

| Modifica                                                                                                           | Descrizione                                                                                                                                                                                                                            | Data             |
|--------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">AWSLambdaMSKExecutionRole</a> — Lambda ha iniziato a tenere traccia delle modifiche a questa politica. | AWSLambdaMSKExecutionRole concede le autorizzazioni per leggere e accedere ai record da un cluster Amazon Managed Streaming for Apache Kafka (Amazon MSK), gestire interfacce di ENIs rete elastiche () e scrivere nei log. CloudWatch | 14 febbraio 2022 |
| <a href="#">AWSLambdaSQSQueueExecutionRole</a> — Lambda ha iniziato a tracciare le modifiche a questa politica.    | AWSLambdaSQSQueueExecutionRole concede le autorizzazioni per leggere un messaggio da una coda Amazon Simple Queue Service (Amazon SQS) e scrivere nei log. CloudWatch                                                                  | 14 febbraio 2022 |
| <a href="#">AWSLambdaVPCLAccessExecutionRole</a> — Lambda ha iniziato a tracciare le modifiche a questa politica.  | AWSLambdaVPCLAccessExecutionRole concede le autorizzazioni per la gestione all' ENIs interno di un Amazon VPC e la scrittura nei log. CloudWatch                                                                                       | 14 febbraio 2022 |
| <a href="#">AWSXRayDaemonWriteAccess</a> — Lambda ha iniziato a tracciare le modifiche a questa politica.          | AWSXRayDaemonWriteAccess concede le autorizzazioni per caricare i dati non elaborati su X-Ray.                                                                                                                                         | 14 febbraio 2022 |

| Modifica                                                                                                                        | Descrizione                                                                                                                                                                           | Data             |
|---------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">CloudWatchLambdaInsightsExecutionRolePolicy</a> —<br>Lambda ha iniziato a tracciare le modifiche a questa politica. | CloudWatchLambdaInsightsExecutionRolePolicy concede le autorizzazioni per scrivere metriche di runtime a CloudWatch Lambda Insights.                                                  | 14 febbraio 2022 |
| <a href="#">AmazonS3 — ObjectLambdaExecutionRolePolicy</a><br>Lambda ha iniziato a tracciare le modifiche a questa politica.    | AmazonS3ObjectLambdaExecutionRolePolicy concede le autorizzazioni per interagire con l'oggetto Lambda di Amazon Simple Storage Service (Amazon S3) e per scrivere su Logs. CloudWatch | 14 febbraio 2022 |

Per alcune funzionalità, la console Lambda tenta di aggiungere autorizzazioni mancanti al ruolo di esecuzione in una policy gestita dal cliente. Queste policy possono diventare numerose. Aggiungere le policy gestite AWS pertinenti al ruolo di esecuzione prima di abilitare le funzionalità per evitare la creazione di policy aggiuntive.

Quando si utilizza una [mappatura origine eventi](#) per invocare la funzione, Lambda utilizza il ruolo di esecuzione per leggere i dati dell'evento. Ad esempio, la mappatura dell'origine eventi per Kinesis legge gli eventi provenienti da un flusso di dati e li invia alla funzione in batch.

Quando un servizio assume un ruolo nel tuo account, puoi includere le chiavi di contesto delle condizioni globali `aws:SourceAccount` e `aws:SourceArn` nella policy di attendibilità del ruolo per limitare l'accesso al ruolo solo alle richieste generate dalle risorse previste. Per ulteriori informazioni consulta [Prevenzione del problema "confused deputy" tra servizi per Servizio di token di sicurezza AWS](#).

Oltre alle policy AWS gestite, la console Lambda fornisce modelli per la creazione di policy personalizzate con autorizzazioni per casi d'uso aggiuntivi. Quando si crea una funzione nella console Lambda, è possibile scegliere di creare un nuovo ruolo di esecuzione con le autorizzazioni da uno o più modelli. Questi modelli vengono applicati automaticamente al momento della creazione

di una funzione da un blueprint, oppure quando si configurano le opzioni che richiedono l'accesso ad altri servizi. [Modelli di esempio sono disponibili nell'archivio di questa guida. GitHub](#)

## Utilizzo dell'ARN della funzione di origine per controllare il comportamento di accesso alla funzione

È normale che il codice della funzione Lambda effettui richieste API ad altri Servizi AWS. Per effettuare queste richieste, Lambda genera un insieme effimero di credenziali assumendo il ruolo di esecuzione della tua funzione. Queste credenziali sono disponibili come variabili d'ambiente durante l'invocazione della funzione. Quando si lavora con AWS SDKs, non è necessario fornire le credenziali per l'SDK direttamente nel codice. Per impostazione predefinita, la catena di fornitori di credenziali controlla in sequenza ogni punto in cui è possibile impostare le credenziali e seleziona la prima disponibile, in genere le variabili d'ambiente (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` e `AWS_SESSION_TOKEN`).

Lambda inserisce la funzione di origine ARN nel contesto delle credenziali se la richiesta è una richiesta AWS API proveniente dall'ambiente di esecuzione. Lambda inserisce anche l'ARN della funzione di origine per le seguenti richieste API AWS che Lambda effettua per tuo conto al di fuori dell'ambiente di esecuzione:

| Servizio            | Azione                                                | Motivo                                                                               |
|---------------------|-------------------------------------------------------|--------------------------------------------------------------------------------------|
| CloudWatch Registri | CreateLogGroup ,<br>CreateLogStream ,<br>PutLogEvents | Per archiviare i log in un gruppo di CloudWatch log Logs                             |
| X-Ray               | PutTraceSegments                                      | Invio dei dati di traccia a X-Ray                                                    |
| Amazon EFS          | ClientMount                                           | Connessione di una funzione a un file system Amazon Elastic File System (Amazon EFS) |

Le altre chiamate AWS API che Lambda effettua al di fuori dell'ambiente di esecuzione per tuo conto utilizzando lo stesso ruolo di esecuzione non contengono la funzione di origine ARN. Esempi di tali chiamate API al di fuori dell'ambiente di esecuzione includono:

- Chiama a AWS Key Management Service (AWS KMS) per crittografare e decrittografare automaticamente le variabili di ambiente.
- Chiamate ad Amazon Elastic Compute Cloud (Amazon EC2) per creare interfacce ENIs di rete elastiche () per una funzione abilitata al VPC.
- [Chiamate a Servizi AWS, come Amazon Simple Queue Service \(Amazon SQS\), per la lettura da un'origine di eventi configurata come mappatura delle sorgenti di eventi.](#)

Con l'ARN della funzione di origine nel contesto delle credenziali puoi verificare se una chiamata alla tua risorsa proviene dal codice di una funzione Lambda specifica. Per verificarlo, usa la chiave di condizione `lambda:SourceFunctionArn` in una policy basata sulle identità IAM o una [policy di controllo dei servizi \(SCP\)](#).

#### Note

Non puoi utilizzare la chiave di condizione `lambda:SourceFunctionArn` nelle policy basate sulle risorse.

Implementando questa condizione nelle tue policy basate sull'identità SCPs, puoi oppure implementare controlli di sicurezza per le azioni API che il tuo codice funzionale esegue su altri. Servizi AWS Questo ha alcune funzioni di sicurezza chiave, come aiutarti a identificare l'origine di una perdita di credenziali.

#### Note

La chiave di condizione `lambda:SourceFunctionArn` è diversa dalle chiavi di condizione `lambda:FunctionArn` e `aws:SourceArn`. La chiave di condizione `lambda:FunctionArn` si applica solo a [mappature di origine eventi](#) e aiuta a definire quali funzioni può richiamare l'origine eventi. La chiave di `aws:SourceArn` condizione si applica solo alle politiche in cui la funzione Lambda è la risorsa di destinazione e aiuta a definire quali altre Servizi AWS risorse possono richiamare quella funzione. La chiave di `lambda:SourceFunctionArn` condizione può essere applicata a qualsiasi policy o SCP basata sull'identità per definire le funzioni Lambda specifiche che dispongono delle autorizzazioni per effettuare chiamate API specifiche ad altre risorse. AWS

Per utilizzare `lambda:SourceFunctionArn` nella tua policy, includila come condizione con uno qualsiasi degli [operatori di condizione ARN](#). Il valore della chiave deve essere un ARN valido.

Ad esempio, supponiamo che il codice della funzione Lambda effettui una chiamata `s3:PutObject` che si rivolge a un determinato bucket Amazon S3. Potresti volere che solo una funzione Lambda specifica abbia accesso `s3:PutObject` a quel bucket. In questo caso, il ruolo di esecuzione della funzione deve avere una policy collegata simile alla seguente:

Example policy che concede l'accesso a una funzione Lambda specifica a una risorsa Amazon S3

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleSourceFunctionArn",
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::lambda_bucket/*",
      "Condition": {
        "ArnEquals": {
          "lambda:SourceFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:source_lambda"
        }
      }
    }
  ]
}
```

Questa policy consente l'accesso `s3:PutObject` solo se l'origine è la funzione Lambda con ARN `arn:aws:lambda:us-east-1:123456789012:function:source_lambda`. La policy non consente l'accesso `s3:PutObject` ad alcun'altra identità chiamante. Ciò è vero anche se una funzione o entità diversa effettua una chiamata `s3:PutObject` con lo stesso ruolo di esecuzione.

#### Note

La chiave di condizione `lambda:SourceFunctionARN` non supporta gli alias delle funzioni o le versioni delle funzioni Lambda. Se utilizzi l'ARN per una particolare versione o alias di una funzione, la tua funzione non avrà l'autorizzazione per eseguire l'azione specificata.

Assicurarsi di utilizzare l'ARN non completo per la funzione senza un suffisso di versione o alias.

Puoi anche usare in `lambda:SourceFunctionArn` SCPs. Ad esempio, supponiamo di voler limitare l'accesso al bucket al codice di una singola funzione Lambda o a chiamate da un cloud privato virtuale (VPC) Amazon specifico. L'SCP seguente illustra questo scenario.

Example politica che nega l'accesso ad Amazon S3 in condizioni specifiche

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:*"
      ],
      "Resource": "arn:aws:s3:::lambda_bucket/*",
      "Effect": "Deny",
      "Condition": {
        "StringNotEqualsIfExists": {
          "aws:SourceVpc": [
            "vpc-12345678"
          ]
        }
      }
    },
    {
      "Action": [
        "s3:*"
      ],
      "Resource": "arn:aws:s3:::lambda_bucket/*",
      "Effect": "Deny",
      "Condition": {
        "ArnNotEqualsIfExists": {
          "lambda:SourceFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:source_lambda"
        }
      }
    }
  ]
}
```

```
}  
]
```

Questa policy nega tutte le azioni S3 a meno che non provengano da una specifica funzione Lambda con ARN `arn:aws:lambda*:*:123456789012:function:source_lambda` o a meno che non provengano dal VPC specificato. L'operatore `StringNotEqualsIfExists` dice a IAM di elaborare questa condizione solo se la chiave `aws:SourceVpc` è presente nella richiesta. Allo stesso modo, IAM considera l'operatore `ArnNotEqualsIfExists` solo se esiste `lambda:SourceFunctionArn`.



## Concedere ad altre AWS entità l'accesso alle tue funzioni Lambda

Per concedere Account AWS ad altre organizzazioni e servizi le autorizzazioni per accedere alle tue risorse Lambda, hai alcune opzioni:

- È possibile utilizzare le [policy basate sull'identità](#) per concedere agli utenti l'accesso alle risorse Lambda. Le policy basate su identità possono essere applicate direttamente agli utenti o ai gruppi e ruoli associati a un utente.
- Puoi utilizzare [policy basate sulle risorse](#) per concedere ad altri account e Servizi AWS autorizzazioni per accedere alle tue risorse Lambda. Quando un utente prova ad accedere a una risorsa Lambda, Lambda considera sia le policy basate su identità dell'utente che la policy basata sulla risorsa della risorsa. Quando un AWS servizio come Amazon Simple Storage Service (Amazon S3) richiama la funzione Lambda, Lambda considera solo la politica basata sulle risorse.
- Puoi utilizzare un modello [controllo degli accessi basato su attributi \(ABAC\)](#), per controllare l'accesso alle funzioni Lambda. Con ABAC, puoi collegare i tag a una funzione Lambda, inviarli in determinate richieste API o collegarli al principale IAM da cui proviene la richiesta. Per controllare gli accessi alla funzione, puoi invece specificare gli stessi tag nell'elemento di condizione di una policy IAM.

Per aiutarti a ottimizzare le autorizzazioni per l'accesso con privilegi minimi, Lambda fornisce alcune condizioni aggiuntive che puoi includere nelle tue policy. Per ulteriori informazioni, consulta [the section called “Risorse e condizioni”](#).

### Identity-based Politiche IAM per Lambda

Puoi utilizzare policy basate sull'identità in AWS Identity and Access Management (IAM) per concedere agli utenti del tuo account l'accesso a Lambda. Identity-based le policy possono essere applicate a utenti, gruppi di utenti o ruoli. È inoltre possibile concedere le autorizzazioni a utenti in un altro account in modo che assumano un ruolo nel proprio account ed eseguano l'accesso alle risorse Lambda.

Lambda fornisce policy AWS gestite che concedono l'accesso alle azioni dell'API Lambda e, in alcuni casi, l'accesso ad altri AWS servizi utilizzati per sviluppare e gestire le risorse Lambda. Lambda aggiorna le policy gestite in base alle necessità, per assicurare che gli utenti possano accedere a nuove caratteristiche quando queste vengono rilasciate.

- [AWS Lambda\\_FullAccess](#): garantisce l'accesso completo alle azioni Lambda e ad altri AWS servizi utilizzati per sviluppare e mantenere le risorse Lambda.

- [AWS Lambda\\_ReadOnlyAccess](#): garantisce l'accesso in sola lettura alle risorse Lambda.
- [AWSLambdaRole](#)— Concede le autorizzazioni per richiamare le funzioni Lambda.

AWS le policy gestite concedono l'autorizzazione alle azioni API senza limitare le funzioni o i livelli Lambda che un utente può modificare. Per un controllo ancora più accurato, è possibile creare le proprie policy che limitano l'ambito di applicazione delle autorizzazioni di un utente. Per informazioni sulla concessione dell'accesso ai livelli, consulta [the section called “Accesso al livello”](#)

## Argomenti

- [Concedere agli utenti l'accesso a una funzione Lambda](#)
- [Concedere agli utenti l'accesso a un livello Lambda](#)

## Concedere agli utenti l'accesso a una funzione Lambda

Utilizza le [policy basate sull'identità](#) per consentire a utenti, gruppi di utenti o ruoli di eseguire operazioni sulle funzioni Lambda.

### Note

Per una funzione definita come immagine di container, l'autorizzazione utente per accedere all'immagine deve essere configurata in Amazon Elastic Container Registry (Amazon ECR). Per un esempio, consulta le [policy del repository di Amazon ECR](#).

Di seguito viene illustrato un esempio di policy di autorizzazione con ambito di applicazione limitato. Ciò consente a un utente di creare e gestire le funzioni Lambda denominate con un prefisso designato (`intern-`) e configurate con un ruolo di esecuzione designato.

## Example Policy di sviluppo delle funzioni

### JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadOnlyPermissions",
      "Effect": "Allow",
```

```

        "Action": [
            "lambda:GetAccountSettings",
            "lambda:GetEventSourceMapping",
            "lambda:GetFunction",
            "lambda:GetFunctionConfiguration",
            "lambda:GetFunctionCodeSigningConfig",
            "lambda:GetFunctionConcurrency",
            "lambda:ListEventSourceMappings",
            "lambda:ListFunctions",
            "lambda:ListTags",
            "iam:ListRoles"
        ],
        "Resource": "*"
    },
    {
        "Sid": "DevelopFunctions",
        "Effect": "Allow",
        "NotAction": [
            "lambda:AddPermission",
            "lambda:PutFunctionConcurrency"
        ],
        "Resource": "arn:aws:lambda:*:*:function:intern-*"
    },
    {
        "Sid": "DevelopEventSourceMappings",
        "Effect": "Allow",
        "Action": [
            "lambda>DeleteEventSourceMapping",
            "lambda:UpdateEventSourceMapping",
            "lambda>CreateEventSourceMapping"
        ],
        "Resource": "*",
        "Condition": {
            "ArnLike": {
                "lambda:FunctionArn": "arn:aws:lambda:*:*:function:intern-*"
            }
        }
    },
    {
        "Sid": "PassExecutionRole",
        "Effect": "Allow",
        "Action": [
            "iam:ListRolePolicies",
            "iam:ListAttachedRolePolicies",

```

```

        "iam:GetRole",
        "iam:GetRolePolicy",
        "iam:PassRole",
        "iam:SimulatePrincipalPolicy"
    ],
    "Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
},
{
    "Sid": "ViewLogs",
    "Effect": "Allow",
    "Action": [
        "logs:*"
    ],
    "Resource": "arn:aws:logs:*:*:log-group:/aws/lambda/intern-*"
}
]
}

```

Le autorizzazioni nella policy sono organizzate in dichiarazioni in base alle [risorse e le condizioni](#) che supportano.

- **ReadOnlyPermissions** – La console Lambda utilizza queste autorizzazioni quando l'utente sfoglia e visualizza le funzioni. Esse non supportano i modelli di risorse o condizioni.

```

    "Action": [
        "lambda:GetAccountSettings",
        "lambda:GetEventSourceMapping",
        "lambda:GetFunction",
        "lambda:GetFunctionConfiguration",
        "lambda:GetFunctionCodeSigningConfig",
        "lambda:GetFunctionConcurrency",
        "lambda:ListEventSourceMappings",
        "lambda:ListFunctions",
        "lambda:ListTags",
        "iam:ListRoles"
    ],
    "Resource": "*"

```

- **DevelopFunctions**: utilizza qualsiasi operazione che opera su funzioni con prefisso `intern-`, eccetto `AddPermission` e `PutFunctionConcurrency`. `AddPermission` modifica la

[policy basata sulle risorse](#) sulla funzione e può avere implicazioni in termini di sicurezza.

PutFunctionConcurrency riserva la capacità di dimensionamento per una funzione e può prendere la capacità da altre funzioni.

```
"NotAction": [
    "lambda:AddPermission",
    "lambda:PutFunctionConcurrency"
],
"Resource": "arn:aws:lambda:*:*:function:intern-*
```

- **DevelopEventSourceMappings**: gestisci le mappature di origine eventi sulle funzioni con prefisso `intern-`. Queste operazioni operano su mappature di origine eventi, ma è possibile limitarle per funzione con una condizione.

```
"Action": [
    "lambda:DeleteEventSourceMapping",
    "lambda:UpdateEventSourceMapping",
    "lambda:CreateEventSourceMapping"
],
"Resource": "*",
"Condition": {
    "StringLike": {
        "lambda:FunctionArn": "arn:aws:lambda:*:*:function:intern-*"
    }
}
```

- **PassExecutionRole** – Visualizzare e trasferire solo un ruolo denominato `intern-lambda-execution-role`, che deve essere creato e gestito da un utente con le autorizzazioni IAM. `PassRole` viene utilizzato quando si assegna un ruolo di esecuzione a una funzione.

```
"Action": [
    "iam:ListRolePolicies",
    "iam:ListAttachedRolePolicies",
    "iam:GetRole",
    "iam:GetRolePolicy",
    "iam:PassRole",
    "iam:SimulatePrincipalPolicy"
],
```

```
"Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
```

- **ViewLogs**— Utilizzare CloudWatch Logs per visualizzare i log delle funzioni il cui prefisso è `intern-`

```
"Action": [
    "logs:*"
],
"Resource": "arn:aws:logs:*:*:log-group:/aws/lambda/intern-*"
```

Questa policy consente a un utente di iniziare con Lambda, senza mettere le risorse di altri utenti a rischio. Non consente a un utente di configurare una funzione da attivare o chiamare altri AWS servizi, il che richiede autorizzazioni IAM più ampie. Inoltre, non include l'autorizzazione a servizi che non supportano politiche ad ambito limitato, come CloudWatch X-Ray. Utilizzare le policy di sola lettura per questi servizi per fornire all'utente l'accesso ai parametri e ai dati di traccia.

Quando configuri i trigger per la tua funzione, devi accedere per utilizzare il AWS servizio che richiama la tua funzione. Ad esempio, per configurare un trigger Amazon S3, è necessario disporre dell'autorizzazione per utilizzare le operazioni Amazon S3 che gestiscono le notifiche dei bucket. Molte di queste autorizzazioni sono incluse nella politica gestita. [AWSLambda\\_FullAccess](#)

Concedere agli utenti l'accesso a un livello Lambda

Utilizza le [policy basate sull'identità](#) per consentire a utenti, gruppi di utenti o ruoli di eseguire operazioni sui livelli Lambda. La seguente policy concede un'autorizzazione utente per creare i livelli e utilizzarli con le funzioni. I modelli di risorse consentono all'utente di lavorare in qualsiasi Regione AWS versione del livello, purché il nome del livello inizi con `test-`.

Example Policy per lo sviluppo dei livelli

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublishLayers",
      "Effect": "Allow",
      "Action": [
```

```

        "lambda:PublishLayerVersion"
    ],
    "Resource": "arn:aws:lambda:*:*:layer:test-*"
  },
  {
    "Sid": "ManageLayerVersions",
    "Effect": "Allow",
    "Action": [
      "lambda:GetLayerVersion",
      "lambda:DeleteLayerVersion"
    ],
    "Resource": "arn:aws:lambda:*:*:layer:test-*:*"
  }
]
}

```

È anche possibile implementare l'utilizzo di livelli durante la creazione e la configurazione della funzione con la condizione `lambda:Layer`. Ad esempio, è possibile evitare che gli utenti utilizzino livelli pubblicati da altri account. La policy seguente aggiunge una condizione per le operazioni `CreateFunction` e `UpdateFunctionConfiguration` per richiedere che i livelli specificati provengano dall'account 123456789012.

## JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ConfigureFunctions",
      "Effect": "Allow",
      "Action": [
        "lambda:CreateFunction",
        "lambda:UpdateFunctionConfiguration"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringLike": {
          "lambda:Layer": [
            "arn:aws:lambda:*:123456789012:layer:*:*"
          ]
        }
      }
    }
  ]
}

```

```
}  
  }  
] }  
}
```

Per garantire che la condizione venga applicata, verificare che nessun'altra dichiarazioni conceda l'autorizzazione all'utente per tali azioni.

## Utilizzo di policy basate sulle risorse in Lambda

Con le politiche di autorizzazione basate sulle risorse, puoi concedere ad altri Account AWS utenti, organizzazioni e l'accesso alle tue funzioni Lambda. Servizi AWS Una policy basata sulle risorse è un documento JSON che contiene una o più dichiarazioni. Ogni dichiarazione definisce quanto segue:

- **Principal:** l'entità a cui vuoi concedere le autorizzazioni (un'altra Servizio AWS, un ruolo o un utente IAM o un altro Account AWS)
- **Action:** un elenco delle azioni API che desideri consentire o negare per il principale specificato
- **Effect:** se desideri consentire o negare al principale la possibilità di utilizzare le azioni API scelte
- **Resource:** la funzione, la versione o l'alias Lambda a cui vuoi applicare l'istruzione (puoi anche usare un carattere jolly per specificare tutte le versioni e gli alias della tua funzione)

Puoi anche utilizzare elementi opzionali come **Sid** (un identificatore di dichiarazione) e **Condition** (condizioni logiche per un controllo granulare degli accessi). Per un elenco completo degli elementi di policy supportati, consulta il riferimento agli elementi di policy JSON di [IAM nella Guida per l'utente](#). AWS Identity and Access Management

### Aggiungere autorizzazioni basate sulle risorse a una funzione Lambda

Puoi aggiungere autorizzazioni basate sulle risorse alla tua funzione Lambda utilizzando due metodi:

- **Policy JSON completa:** utilizza la console Lambda o l'azione [PutResourcePolicy](#) API per AWS CLI aggiungere un documento completo sulla policy JSON. Con una policy JSON completa, puoi utilizzare la gamma completa di chiavi di condizione globali [IAM](#), aggiungere più istruzioni con più principali e creare istruzioni esplicite. Deny La dimensione massima per una policy basata su risorse JSON è di 20 KB.
- **Autorizzazioni individuali:** utilizza la console o l'azione [AddPermission](#) API per aggiungere singole istruzioni. Allow Le autorizzazioni individuali supportano solo un set limitato di chiavi di condizione (`aws:SourceArns:SourceAccount`, `eaws:PrincipalOrgID`).



Ti consigliamo di definire policy JSON complete per aggiungere autorizzazioni basate sulle risorse alla tua funzione. La creazione di una policy JSON completa ti offre maggiore flessibilità e un controllo dettagliato sulle tue autorizzazioni.

#### Important

L'utilizzo `put-resource-policy` sostituisce qualsiasi politica basata sulle risorse esistente sulla risorsa. Se la risorsa dispone già delle autorizzazioni definite con, le sovrascrive. `add-permission` `put-resource-policy` `get-resource-policy` Da utilizzare per recuperare la policy esistente prima di apportare modifiche.

### Autorizzazioni richieste

Per utilizzare le azioni `PutResourcePolicy`, `GetResourcePolicy`, e `DeleteResourcePolicy` API, sono necessarie le seguenti autorizzazioni IAM:

| Azione API                        | Autorizzazioni richieste                                                                                            |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------|
| <code>PutResourcePolicy</code>    | <code>lambda:PutResourcePolicy</code> ,<br><code>lambda:AddPermission</code> e <code>lambda:RemovePermission</code> |
| <code>GetResourcePolicy</code>    | <code>lambda:GetResourcePolicy</code> e<br><code>lambda:GetPolicy</code>                                            |
| <code>DeleteResourcePolicy</code> | <code>lambda&gt;DeleteResourcePolicy</code> e<br><code>lambda:RemovePermission</code>                               |

### Console

Per creare una policy completa basata sulle risorse JSON

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Seleziona la funzione a cui desideri concedere l'accesso, quindi seleziona la scheda Configurazione.
3. Seleziona Autorizzazioni.

4. Nel riquadro delle dichiarazioni Resource-based politiche, scegli Modifica. Questa azione apre l'editor delle politiche JSON.
5. Aggiungi dichiarazioni alla policy della tua funzione. Puoi selezionare azioni API, aggiungere principali di diversi tipi (servizi, account, ruoli IAM) e aggiungere chiavi di condizione per controllare l'accesso.
6. Selezionare Salva.

Puoi anche modificare la policy basata sulle risorse della tua funzione direttamente nel riquadro Politica.

## AWS CLI

Per aggiungere una policy completa basata sulle risorse JSON

Per aggiungere una policy JSON completa alla tua funzione, usa il comando `put-resource-policy` AWS CLI

Il comando di esempio seguente aggiunge una policy basata sulle risorse alla funzione utilizzando una policy definita in un file denominato `policy.json` nel computer locale. Esegui il comando dalla directory che contiene il file. `resource-arn` È possibile specificare una versione o un alias della funzione oppure è possibile utilizzare la funzione non qualificata ARN per applicare la politica all'intera funzione.

```
aws lambda put-resource-policy --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function \
--policy file://policy.json
```

## Lambda APIs

Per aggiungere una politica di autorizzazioni JSON completa alla tua funzione, usa l'azione API. [PutResourcePolicy](#) Puoi anche eliminare la policy di una funzione utilizzando l'[DeleteResourcePolicy](#) azione o recuperare la policy attualmente associata a una funzione utilizzando l'azione. [GetResourcePolicy](#)

Per aggiungere autorizzazioni individuali alla policy di una funzione, utilizza l'azione [AddPermission](#) API.

## Visualizzazione della politica basata sulle risorse di una funzione

### Console

Per visualizzare le policy basate sulle risorse di una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione), quindi Permissions (Autorizzazioni).
4. Scorri verso il basso fino alle dichiarazioni **Resource-based politiche** per visualizzare la politica.

### AWS CLI

Per visualizzare una policy basata sulle risorse di una funzione, utilizzare il comando `get-resource-policy`. Puoi utilizzare l'ARN non qualificato della tua funzione o specificare una versione o un alias.

```
aws lambda get-resource-policy --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function
```

Puoi anche usare il comando: `get-policy`

```
aws lambda get-policy \
  --function-name my-function \
  --output text
```

### Lambda APIs

Per recuperare la policy basata sulle risorse di una funzione, usa l'azione API.

[GetResourcePolicy](#) Puoi utilizzare l'ARN non qualificato della tua funzione o specificare una versione o un alias ARN.

Puoi anche utilizzare l'azione API. [GetPolicy](#)

## Eliminazione della politica basata sulle risorse di una funzione

### Console

Per eliminare la politica basata sulle risorse di una funzione

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Seleziona la funzione per cui desideri eliminare la politica delle autorizzazioni.
3. Seleziona la scheda Configurazione, quindi Autorizzazioni.
4. Nel riquadro delle dichiarazioni Resource-based politiche, scegli Elimina.

### AWS CLI

Per eliminare la policy della funzione, usa il `delete-resource-policy` comando e specifica l'ARN della funzione, della versione o dell'alias da cui desideri eliminare la policy.

```
aws lambda delete-resource-policy --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function
```

Per rimuovere singole dichiarazioni, usa `remove-permission`

```
aws lambda remove-permission \
  --function-name example \
  --statement-id sns
```

### Lambda APIs

Per eliminare la policy basata sulle risorse associata a una funzione, a una versione della funzione o a un alias di funzione, usa l'azione API. [DeleteResourcePolicy](#)

Per rimuovere singole dichiarazioni da una policy, usa l'azione API. [RemovePermission](#)

### Aggiornamento delle politiche esistenti

Quando si aggiornano le autorizzazioni basate sulle risorse esistenti di una funzione, il comportamento dipende dal metodo utilizzato:

- `put-resource-policy`/[PutResourcePolicy](#)— Sostituisce l'intera policy esistente. Tutte le autorizzazioni individuali aggiunte in precedenza vengono sovrascritte.

- `add-permission`/[AddPermission](#)— Aggiunge una dichiarazione alla policy esistente senza sovrascriverla. Se chiami `add-permission` dopo `put-resource-policy`, la nuova istruzione viene aggiunta alla policy JSON esistente.

Per evitare di sovrascrivere involontariamente le autorizzazioni esistenti durante l'utilizzo di `put-resource-policy`, recupera prima la policy esistente della funzione. L'output di `get-resource-policy` include un campo `RevisionId`.

```
aws lambda get-resource-policy --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function
```

Quando alleggi una nuova policy, fornisci al `RevisionId` valore il `--revision-id` parametro per assicurarti di aggiornare la versione più recente. Se fornisci un ID di revisione precedente, Lambda non aggiorna la policy della tua funzione.

```
aws lambda put-resource-policy \
  --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function \
  --policy file://policy.json \
  --revision-id a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

#### Note

Le policy esistenti basate sulle risorse create con `AddPermission` continuano a funzionare senza modifiche. Non è necessaria alcuna modifica al codice della funzione per utilizzare la nuova API. `PutResourcePolicy`.

## Best practice di sicurezza

Con le policy basate sulle risorse JSON, puoi seguire i modelli di accesso con privilegi minimi. Puoi anche soddisfare i requisiti normativi per i dinieghi espliciti. Con le politiche JSON complete, puoi:

- Crea Deny dichiarazioni esplicite per bloccare principi o condizioni specifici.
- Utilizza le condizioni organizzative (`aws:PrincipalOrgID`, `aws:PrincipalOrgPaths`) per limitare l'accesso alle tue organizzazioni senza enumerare i singoli account.
- Ambita le autorizzazioni per account di origine o ARN specifici utilizzando l'intera gamma di chiavi di condizione globali IAM.

## Esempi di politiche basate sulle risorse

### Example Concessione dell'autorizzazione ad Amazon S3 con una dichiarazione di rifiuto

La seguente politica concede a tutti i bucket Amazon S3 in un account il permesso di richiamare una funzione, ad eccezione di un bucket che viene esplicitamente negato.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "allow-s3",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        }
      }
    },
    {
      "Sid": "deny-s3-bucket",
      "Effect": "Deny",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": [
        "arn:aws:lambda:us-east-2:111122223333:function:my-function",
        "arn:aws:lambda:us-east-2:111122223333:function:my-function:*"
      ],
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
        }
      }
    }
  ]
}
```

## Example Concessione dell'autorizzazione agli account di un'organizzazione

Le seguenti concessioni basate su criteri richiamano l'accesso a tutti i membri di un'organizzazione Account AWS .

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "org-access",
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Principal": "*",
      "Resource": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": "o-a1b2c3d4e5f"
        }
      }
    }
  ]
}
```

## Example Concessione dell'autorizzazione a più ruoli IAM con condizioni

La seguente policy concede l'autorizzazione a due ruoli IAM di utilizzare l' CreateAlias azione da un indirizzo IP specificato.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "allow-roles",
      "Effect": "Allow",
      "Action": "lambda:CreateAlias",
      "Principal": {
        "AWS": [
          "arn:aws:iam::444455556666:role/role-name",
          "arn:aws:iam::444455556666:role/role-name2"
        ]
      },
      "Resource": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
      "Condition": {
```

```

        "IpAddress": {
            "aws:SourceIp": "192.0.2.0"
        }
    }
}
]
}

```

Example Negare l'accesso a meno che il chiamante non faccia parte di un'organizzazione specificata

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "deny-access",
      "Effect": "Deny",
      "Action": "lambda:InvokeFunction",
      "Principal": "*",
      "Resource": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
      "Condition": {
        "ForAllValues:StringNotLike": {
          "aws:PrincipalOrgPaths": [
            "o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/*"
          ]
        }
      }
    },
    {
      "Sid": "allow-access",
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Principal": "*",
      "Resource": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
      "Condition": {
        "ForAnyValue:StringLike": {
          "aws:PrincipalOrgPaths": [
            "o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/*"
          ]
        }
      }
    }
  ]
}

```



## Operazioni API supportate

Le seguenti operazioni dell'API Lambda supportano le policy basate sulle risorse:

- [CreateAlias](#)
- [DeleteAlias](#)
- [DeleteFunction](#)
- [DeleteFunctionConcurrency](#)
- [DeleteFunctionEventInvokeConfig](#)
- [DeleteProvisionedConcurrencyConfig](#)
- [GetAlias](#)
- [GetFunction](#)
- [GetFunctionConcurrency](#)
- [GetFunctionConfiguration](#)
- [GetFunctionEventInvokeConfig](#)
- [GetPolicy](#)
- [GetProvisionedConcurrencyConfig](#)
- [Invoke](#)
- [InvokeFunctionUrl](#)(solo autorizzazione)
- [ListAliases](#)
- [ListFunctionEventInvokeConfigs](#)
- [ListProvisionedConcurrencyConfigs](#)
- [ListTags](#)
- [ListVersionsByFunction](#)
- [PublishVersion](#)
- [PutFunctionConcurrency](#)
- [PutFunctionEventInvokeConfig](#)
- [PutProvisionedConcurrencyConfig](#)
- [TagResource](#)
- [UntagResource](#)
- [UpdateAlias](#)
- [UpdateFunctionCode](#)

- [UpdateFunctionEventInvokeConfig](#)

## Concessione dell'accesso alla funzione Lambda a Servizi AWS

Quando [utilizzi un AWS servizio per richiamare la tua funzione](#), concedi l'autorizzazione in una dichiarazione su una politica basata sulle risorse. È possibile applicare l'istruzione all'intera funzione oppure limitare l'istruzione a una singola versione o alias.

### Note

Quando si aggiunge un trigger a una funzione con la console Lambda, la console aggiorna la policy basata sulle risorse della funzione per consentire al servizio di invocarla. Per concedere le autorizzazioni ad altri account o servizi che non sono disponibili nella console Lambda, puoi utilizzare l'interfaccia a riga di comando di AWS CLI.

Si consiglia di `put-resource-policy` utilizzarlo per definire una policy JSON completa. `Compute-resource-policy`, puoi aggiungere dichiarazioni di rifiuto, utilizzare l'intera gamma di chiavi di condizione IAM e gestire tutte le autorizzazioni in un unico documento. Puoi anche utilizzare `add-permission` per aggiungere singole dichiarazioni per semplici casi d'uso.

### Important

L'utilizzo `put-resource-policy` sostituisce qualsiasi politica basata sulle risorse esistente sulla risorsa. Se la risorsa dispone già delle autorizzazioni definite con, le sovrascrive.  
`add-permission put-resource-policy get-resource-policy` Da utilizzare per recuperare la policy esistente prima di apportare modifiche.

## Utilizzo di una policy JSON completa (consigliata)

La seguente politica di esempio concede ad Amazon Simple Notification Service l'autorizzazione a richiamare una funzione denominata `my-function`, ma solo per un argomento specifico. Inoltre, nega le invocazioni da un argomento particolare.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

        "Sid": "allow-sns-my-topic",
        "Effect": "Allow",
        "Principal": {
            "Service": "sns.amazonaws.com"
        },
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
        "Condition": {
            "ArnLike": {
                "aws:SourceArn": "arn:aws:sns:us-east-2:123456789012:my-topic"
            }
        }
    },
    {
        "Sid": "deny-sns-other-topic",
        "Effect": "Deny",
        "Principal": {
            "Service": "sns.amazonaws.com"
        },
        "Action": "lambda:InvokeFunction",
        "Resource": [
            "arn:aws:lambda:us-east-2:123456789012:function:my-function",
            "arn:aws:lambda:us-east-2:123456789012:function:my-function:*"
        ],
        "Condition": {
            "ArnLike": {
                "aws:SourceArn": "arn:aws:sns:us-east-2:123456789012:restricted-
topic"
            }
        }
    }
]
}

```

Salva la policy in un file denominato `policy.json` e applicala con: `put-resource-policy`

```

aws lambda put-resource-policy \
  --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function \
  --policy file://policy.json

```

Per Amazon S3, l'origine è un bucket il cui ARN non include un ID di account. È possibile che tu possa eliminare un bucket e che un altro account possa creare un bucket con lo stesso nome. Usa

la chiave `aws:SourceAccount` condizionale per assicurarti che solo le risorse del tuo account possano richiamare la funzione:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "allow-s3",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:s3:::my-bucket"
        }
      }
    }
  ]
}
```

## Utilizzo di `add-permission`

Per casi d'uso semplici, è possibile utilizzare `add-permission` per aggiungere singole dichiarazioni. Il comando seguente concede ad Amazon Simple Notification Service il permesso di richiamare una funzione denominata: `my-function`

```
aws lambda add-permission \
  --function-name my-function \
  --action lambda:InvokeFunction \
  --statement-id sns \
  --principal sns.amazonaws.com \
  --output text
```

Verrà visualizzato l'output seguente:

```
{
  "Sid": "sns",
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": "lambda:InvokeFunction",
  "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function"
}
```

### Note

Se chiami `add-permission` dopo `put-resource-policy`, la nuova dichiarazione viene aggiunta alla policy JSON esistente.

Per limitare le invocazioni a una risorsa specifica, usa l'opzione: `source-arn`

```
aws lambda add-permission \
  --function-name my-function \
  --action lambda:InvokeFunction \
  --statement-id sns-my-topic \
  --principal sns.amazonaws.com \
  --source-arn arn:aws:sns:us-east-2:123456789012:my-topic
```

Per Amazon S3, utilizza l'opzione `source-account` con l'ID del tuo account:

```
aws lambda add-permission \
  --function-name my-function \
  --action lambda:InvokeFunction \
  --statement-id s3-account \
  --principal s3.amazonaws.com \
  --source-arn arn:aws:s3:::amzn-s3-demo-bucket \
  --source-account 123456789012
```

Concedere l'accesso alle funzioni a un'organizzazione

Per concedere le autorizzazioni a tutti gli account di un' [AWS Organizations](#) organizzazione, ti consigliamo di `put-resource-policy` creare una policy JSON completa con la chiave di condizione `aws:PrincipalOrgID`

### Important

L'utilizzo `put-resource-policy` sostituisce qualsiasi politica basata sulle risorse esistente sulla risorsa. Se la risorsa dispone già delle autorizzazioni definite con, le sovrascrive.

`add-permission put-resource-policy get-resource-policy` Da utilizzare per recuperare la policy esistente prima di apportare modifiche.

Utilizzo di una policy JSON completa (consigliata)

Il seguente esempio di policy grant richiama l'accesso a tutti i Account AWS membri dell'organizzazione: `o-a1b2c3d4e5f`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "org-access",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": "o-a1b2c3d4e5f"
        }
      }
    }
  ]
}
```

Salva la policy in un file denominato `policy.json` e applicala:

```
aws lambda put-resource-policy \
  --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function \
  --policy file://policy.json
```

Con una policy JSON completa, puoi anche negare l'accesso a account specifici all'interno di un'organizzazione:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "allow-org",
```

```

    "Effect": "Allow",
    "Principal": "*",
    "Action": "lambda:InvokeFunction",
    "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "Condition": {
      "StringEquals": {
        "aws:PrincipalOrgID": "o-a1b2c3d4e5f"
      }
    }
  },
  {
    "Sid": "deny-specific-account",
    "Effect": "Deny",
    "Principal": {
      "AWS": "arn:aws:iam::999888777666:root"
    },
    "Action": "lambda:InvokeFunction",
    "Resource": [
      "arn:aws:lambda:us-east-2:123456789012:function:my-function",
      "arn:aws:lambda:us-east-2:123456789012:function:my-function:*"
    ]
  }
]
}

```

## Utilizzo di add-permission

Per casi d'uso semplici, puoi utilizzare `add-permission` con l'opzione `principal-org-id`. Il comando seguente concede l'accesso alla chiamata a tutti gli utenti dell'organizzazione: `o-a1b2c3d4e5f`

```

aws lambda add-permission \
  --function-name my-function \
  --statement-id org-access \
  --action lambda:InvokeFunction \
  --principal * \
  --principal-org-id o-a1b2c3d4e5f

```

### Note

In questo comando, `Principal` è `*`. Ciò significa che tutti gli utenti dell'organizzazione `o-a1b2c3d4e5f` ottengono le autorizzazioni di invocazione delle funzioni. Se si specifica un

ruolo Account AWS o come `Principal`, solo quel principale ottiene i permessi di richiamo della funzione, ma solo se fa anche parte dell'organizzazione. `o-a1b2c3d4e5f`

Per ulteriori informazioni, consulta [aws: PrincipalOrg ID](#) nella guida utente IAM.

Concessione dell'accesso alla funziona Lambda ad altri account

Per condividere una funzione con un'altra Account AWS, aggiungi una dichiarazione delle autorizzazioni per più account alla policy basata sulle risorse della [funzione](#). Ti consigliamo di `put-resource-policy` utilizzarlo per creare una policy JSON completa. `Conput-resource-policy`, puoi specificare più principali, aggiungere condizioni e creare dichiarazioni di rifiuto. È possibile utilizzarlo anche `add-permission` per casi d'uso semplici.

#### Important

L'utilizzo `put-resource-policy` sostituisce qualsiasi politica basata sulle risorse esistente sulla risorsa. Se la risorsa dispone già delle autorizzazioni definite con, le sovrascrive. `add-permission` `put-resource-policy` `get-resource-policy` Da utilizzare per recuperare la policy esistente prima di apportare modifiche.

Utilizzo di una policy JSON completa (consigliata)

La seguente policy di esempio concede due Account AWS autorizzazioni per richiamare una funzione, di cui una limitata a un ruolo IAM specifico:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "allow-account-invoke",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:root",
          "arn:aws:iam::444455556666:role/cross-account-role"
        ]
      },
      "Action": "lambda:InvokeFunction",
```



```

        "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function"
    }
  ]
}

```

Salva la policy in un file denominato `policy.json` e applicala. `resource-arn` Possono specificare una funzione, una versione o un alias:

```

aws lambda put-resource-policy \
  --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function \
  --policy file://policy.json

```

Puoi anche limitare l'accesso tra account a versioni specifiche utilizzando un alias. Le seguenti policy prevedono l'accesso solo tramite l'alias: `prod`

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "allow-account-prod-alias",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-
function:prod"
    }
  ]
}

```

Salva questo criterio in un file denominato `alias-policy.json` e applicalo all'alias:

```

aws lambda put-resource-policy \
  --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function:prod \
  --policy file://alias-policy.json

```

La policy basata sulle risorse concede l'autorizzazione all'altro account per accedere alla funzione, ma non consente agli utenti in tale account di superare le autorizzazioni. Gli utenti nell'altro account devono disporre delle [autorizzazioni utente](#) corrispondenti per utilizzare l'API Lambda.

È possibile concedere l'accesso tra account per qualsiasi operazione API che opera su una funzione esistente. Ad esempio, è possibile concedere l'accesso a un account `lambda:ListAliases` per ottenere un elenco di alias, oppure `lambda:GetFunction` per scaricare il codice funzione.

Per concedere ad altri account l'autorizzazione per più funzioni o per operazioni che non operano su una funzione, utilizzare [ruoli IAM](#).

### Utilizzo di add-permission

Per casi d'uso semplici, è possibile utilizzare `add-permission` per aggiungere singole dichiarazioni. L'esempio seguente concede all'account 111122223333 il permesso di invocare `my-function` con l'`prod` alias:

```
aws lambda add-permission \
  --function-name my-function:prod \
  --statement-id xaccount \
  --action lambda:InvokeFunction \
  --principal 111122223333 \
  --output text
```

Verrà visualizzato l'output seguente:

```
{"Sid": "xaccount", "Effect": "Allow", "Principal":
{"AWS": "arn:aws:iam::111122223333:root"}, "Action": "lambda:InvokeFunction", "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function"}
```

#### Note

Se chiami `add-permission` dopo `put-resource-policy`, la nuova dichiarazione viene aggiunta alla policy JSON esistente.

Per limitare l'accesso a un utente, un gruppo o un ruolo in un altro account, specificare l'ARN completo dell'identità come principale. Ad esempio, `arn:aws:iam::123456789012:user/developer`.

### Concessione dell'accesso ai livelli Lambda ad altri account

[Per condividere un layer con un altro Account AWS, aggiungi una dichiarazione di autorizzazione per più account alla policy basata sulle risorse del layer.](#) Esegui il [add-layer-version-permission](#) comando

e specifica l'ID dell'account come `principal`. In ogni istruzione, puoi concedere l'autorizzazione a un singolo account, a tutti gli account o a un'organizzazione in [AWS Organizations](#).

L'esempio seguente concede all'account 111122223333 l'accesso alla versione 2 del livello `bash-runtime`.

```
aws lambda add-layer-version-permission \  
  --layer-name bash-runtime \  
  --version-number 2 \  
  --statement-id xaccount \  
  --action lambda:GetLayerVersion \  
  --principal 111122223333 \  
  --output text
```

Verrà visualizzato un output simile al seguente:

```
{"Sid":"xaccount","Effect":"Allow","Principal":  
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:lam  
east-1:123456789012:layer:bash-runtime:2"}
```

Le autorizzazioni si applicano solo a un'unica versione di un livello. Ripeti la procedura ogni volta che crei la nuova versione di un livello.

Per concedere le autorizzazioni a tutti gli account in un'organizzazione [AWS Organizations](#), è possibile utilizzare l'opzione `organization-id`. L'esempio seguente concede a tutti gli account dell'organizzazione l'autorizzazione `o-t194hfs8cz` per utilizzare la versione 3 di `my-layer`.

```
aws lambda add-layer-version-permission \  
  --layer-name my-layer \  
  --version-number 3 \  
  --statement-id engineering-org \  
  --principal '*' \  
  --action lambda:GetLayerVersion \  
  --organization-id o-t194hfs8cz \  
  --output text
```

Verrà visualizzato l'output seguente:

```
{"Sid":"engineering-  
org","Effect":"Allow","Principal":"*", "Action":"lambda:GetLayerVersion","Resource":"arn:aws:lam
```

```
east-2:123456789012:layer:my-layer:3", "Condition": {"StringEquals":  
{"aws:PrincipalOrgID": "o-t194hfs8cz"}}}]"]
```

Per concedere l'autorizzazione a più account o organizzazioni, devi aggiungere più istruzioni.

## Utilizzo del controllo degli accessi basato sugli attributi in Lambda

Con il [controllo degli accessi basato su attributi \(ABAC\)](#), puoi usare i tag per controllare l'accesso alle risorse Lambda. Puoi allegare tag a determinate risorse Lambda, allegarli a determinate richieste API o allegarli al principale AWS Identity and Access Management (IAM) che effettua la richiesta. Per ulteriori informazioni su come AWS concedere l'accesso basato sugli attributi, consulta [Controllare l'accesso alle AWS risorse utilizzando i tag](#) nella IAM User Guide.

Puoi utilizzare ABAC per [concedere il privilegio minimo](#) senza specificare un nome della risorsa Amazon (ARN) o uno schema ARN nella policy IAM. Per controllare gli accessi puoi invece specificare un tag nell'[elemento di condizione](#) di una policy IAM. La scalabilità è più facile con ABAC perché non è necessario aggiornare le policy IAM quando vengono create nuove risorse. Invece, per controllare l'accesso aggiungi i tag alle nuove risorse.

In Lambda, i tag funzionano sulle seguenti risorse:

- Funzioni: per ulteriori informazioni sul tagging delle funzioni, consulta [the section called “Tag”](#).
- Configurazioni di firma del codice: per ulteriori informazioni sul tagging delle configurazioni di firma del codice, consulta [the section called “Tag di configurazione della firma del codice”](#).
- Strumenti di mappatura dell'origine degli eventi: per ulteriori informazioni sul tagging degli strumenti di mappatura dell'origine degli eventi, consulta [the section called “Tag dello strumento di mappatura dell'origine degli eventi”](#).

I tag non sono supportati per i livelli.

È possibile utilizzare le seguenti chiavi di condizione per scrivere regole di policy IAM basate sui tag:

- [aws: ResourceTag/tag-key](#): Controlla l'accesso in base ai tag allegati a una risorsa Lambda.
- [aws: RequestTag/tag-key](#): Richiede che i tag siano presenti in una richiesta, ad esempio quando si crea una nuova funzione.
- [aws: PrincipalTag/tag-key](#): Controlla cosa può fare il principale IAM (la persona che effettua la richiesta) in base ai tag allegati al suo [utente](#) o [ruolo IAM](#).
- [aws: TagKeys](#): Controlla se è possibile utilizzare chiavi di tag specifiche in una richiesta.

Puoi specificare solo le condizioni per le azioni che le supportano. Per un elenco delle operazioni supportate da ogni operazione Lambda, consulta [Operazioni, risorse e chiavi di condizione per AWS Lambda](#) in Service Authorization Reference. Per `aws: ResourceTag/tag-key` support, consulta «Tipi di risorse definiti da» AWS Lambda. Per `aws: RequestTag/tag-key` e `aws: TagKeys` support, consulta «Azioni definite da» AWS Lambda.

## Argomenti

- [Proteggere le funzioni con un tag](#)

### Proteggere le funzioni con un tag

La seguente procedura mostra un modo per impostare le autorizzazioni per le funzioni tramite ABAC. In questo scenario di esempio, creerai quattro policy di autorizzazione IAM. Quindi, collegherai queste policy a un nuovo ruolo IAM. Infine, creerai un utente IAM e gli assegnerai l'autorizzazione per assumere il nuovo ruolo.

## Argomenti

- [Prerequisiti](#)
- [Fase 1: richiesta di tag sulle nuove funzioni](#)
- [Fase 2: Consentire azioni basate su tag collegati a una funzione Lambda e al principale IAM](#)
- [Fase 3: Concessione delle autorizzazioni di elenco](#)
- [Fase 4: Concessione delle autorizzazioni IAM](#)
- [Fase 5: Creazione del ruolo IAM](#)
- [Fase 6: Creazione dell'utente IAM](#)
- [Fase 7: Test delle autorizzazioni](#)
- [Fase 8: eliminare le risorse](#)

## Prerequisiti

Assicurati di avere un [ruolo di esecuzione Lambda](#). Userai questo ruolo quando concedi le autorizzazioni IAM e crei una funzione Lambda.

### Fase 1: richiesta di tag sulle nuove funzioni

Quando utilizzi ABAC con Lambda, è consigliabile richiedere che tutte le funzioni abbiano i tag. Questo aiuta a garantire che le policy di autorizzazione ABAC funzionino come previsto.

[Crea una policy IAM](#) simile a quella del seguente esempio. Questa policy utilizza le [chiavi di TagKeys](#) [condizione](#) `aws: ResourceTag/tag-key`, `aws:` e `aws:` per richiedere che le nuove funzioni e il responsabile IAM che le crea abbiano entrambi il `project` tag. `RequestTag/tag-key` Il modificatore `ForAllValues` assicura che `project` sia l'unico tag consentito. Se non includi il modificatore `ForAllValues`, gli utenti potranno aggiungere altri tag alla funzione purché passino anche loro `project`.

Example: richiesta di tag sulle nuove funzioni

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:CreateFunction",
        "lambda:TagResource"
      ],
      "Resource": "arn:aws:lambda:*:*:function:*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/project": "${aws:PrincipalTag/project}",
          "aws:ResourceTag/project": "${aws:PrincipalTag/project}"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": "project"
        }
      }
    }
  ]
}
```

Fase 2: Consentire azioni basate su tag collegati a una funzione Lambda e al principale IAM

Crea una seconda policy IAM utilizzando la chiave di `ResourceTag/tag-key` [condizione](#) `aws:` per richiedere che il tag del principale corrisponda al tag associato alla funzione. La seguente policy di esempio consente ai principali con tag `project` di richiamare le funzioni con il tag `project`. Se una funzione ha altri tag, l'azione è negata.

Example: Richiesta di tag corrispondenti sulla funzione e sul principale IAM

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction",
        "lambda:GetFunction"
      ],
      "Resource": "arn:aws:lambda:*:*:function:*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/project": "${aws:PrincipalTag/project}"
        }
      }
    }
  ]
}
```

Fase 3: Concessione delle autorizzazioni di elenco

Crea una policy che consenta al principale di elencare le funzioni Lambda e i ruoli IAM. Ciò consente al principale di vedere tutte le funzioni Lambda e i ruoli IAM sulla console e quando si chiamano le operazioni API.

Example: Concessione delle autorizzazioni di elenco per Lambda e IAM

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllResourcesLambdaNoTags",
      "Effect": "Allow",
      "Action": [
```

```
        "lambda:GetAccountSettings",
        "lambda:ListFunctions",
        "iam:ListRoles"
    ],
    "Resource": "*"
}
]
```

#### Fase 4: Concessione delle autorizzazioni IAM

Crea una policy che consenta `iam: PassRole`. Questa autorizzazione è necessaria quando si assegna un ruolo di esecuzione a una funzione. Nella seguente policy di esempio, sostituisci l'ARN di esempio con l'ARN del tuo ruolo di esecuzione Lambda.

##### Note

Non utilizzare la chiave di condizione `ResourceTag` in una policy con l'operazione `iam: PassRole`. Non puoi utilizzare il tag su un ruolo IAM per controllare l'accesso a chi può passare tale ruolo. Per ulteriori informazioni sulle autorizzazioni necessarie per passare un ruolo a un servizio, vedere [Concessione a un utente delle autorizzazioni per passare un ruolo a un servizio. AWS](#)

Example: Concessione dell'autorizzazione per inviare il ruolo di esecuzione

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::111122223333:role/lambda-ex"
    }
  ]
}
```



```
}
```

## Fase 5: Creazione del ruolo IAM

Una best practice consiste nell'[utilizzare i ruoli per delegare le autorizzazioni](#). [Crea un ruolo IAM](#) chiamato `abac-project-role`:

- In Fase 1: Selezione dell'entità attendibile, scegli Account AWS , quindi Questo account.
- In Fase 2: Aggiunta delle autorizzazioni, collega le quattro policy IAM create nei passaggi precedenti.
- In Fase 3: Assegnazione di un nome, revisione e creazione, scegli Add tag (Aggiungi tag). In Chiave, inserire `project`. Non immettere un valore.

## Fase 6: Creazione dell'utente IAM

[Crea un utente IAM](#) chiamato `abac-test-user`. Nella sezione Set permissions (Imposta autorizzazioni), seleziona Attach existing policies directly (Collega direttamente le policy esistenti), quindi Create policy (Crea policy). Immetti la seguente definizione di policy. Sostituisci **111122223333** con il tuo [ID account AWS](#). Questa policy consente a `abac-test-user` di assumere il ruolo `abac-project-role`.

Example: Consentire all'utente IAM di assumere il ruolo ABAC

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::111122223333:role/abac-project-role"
  }
}
```

## Fase 7: Test delle autorizzazioni

1. Accedi alla console come `AWS abac-test-user`. Per ulteriori informazioni, consulta [Accesso come utente IAM](#).
2. Passare al ruolo `abac-project-role`. Per ulteriori informazioni, consulta [Cambio di un ruolo \(console\)](#).
3. [Crea una funzione Lambda](#):
  - In Permissions (Autorizzazioni), scegli Change default execution role (Modifica ruolo di esecuzione predefinito), quindi per Execution role (Ruolo di esecuzione), scegli Use an existing role (Utilizza un ruolo esistente). Scegli lo stesso ruolo di esecuzione che hai usato in [Fase 4: Concessione delle autorizzazioni IAM](#).
  - In Advanced settings (Impostazioni avanzate), seleziona Enable tags (Abilita tag) quindi scegli Add new tag (Aggiungi nuovo tag). In Chiave, inserire `project`. Non immettere un valore.
4. [Esegui il test della funzione](#).
5. Crea una seconda funzione Lambda e aggiungi un tag diverso, ad esempio `environment`. Questa operazione dovrebbe non riuscire perché la policy ABAC creata in [Fase 1: richiesta di tag sulle nuove funzioni](#) consente al principale di creare solo funzioni con tag `project`.
6. Crea una terza funzione senza tag. Questa operazione dovrebbe non riuscire perché la policy ABAC creata in [Fase 1: richiesta di tag sulle nuove funzioni](#) non consente al principale di creare funzioni senza tag.

Con questa strategia di autorizzazione, puoi controllare l'accesso senza creare nuove politiche per ogni nuovo utente. Per concedere l'accesso a nuovi utenti, è sufficiente concedere loro l'autorizzazione per assumere il ruolo che corrisponde al progetto assegnato.

## Fase 8: eliminare le risorse

Per eliminare il ruolo IAM

1. Aprire la pagina [Ruoli](#) della console IAM.
2. Seleziona il ruolo creato nella [fase 5](#).
3. Scegliere Elimina.
4. Per confermare l'eliminazione, inserisci il nome del ruolo nel campo di immissione del testo.
5. Scegliere Elimina.

## Per eliminare l'utente IAM

1. Apri la pagina [Utenti](#) nella console IAM.
2. Seleziona l'utente IAM creato nella [fase 6](#).
3. Scegliere Elimina.
4. Per confermare l'eliminazione, inserisci il nome utente nel campo di immissione del testo.
5. Scegli Elimina utente.

## Per eliminare la funzione Lambda

1. Aprire la pagina [Functions \(Funzioni\)](#) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **confirm** nel campo di immissione testo e scegli Delete (Elimina).

## Fine-tuning le sezioni Risorse e condizioni delle politiche

Puoi limitare l'ambito delle autorizzazioni di un utente specificando le risorse e le condizioni in una policy AWS Identity and Access Management (IAM). Ogni operazione in una policy supporta una combinazione di tipi di risorse e condizioni che varia a seconda del comportamento dell'operazione.

Ogni dichiarazione di policy IAM concede l'autorizzazione a un'operazione eseguita su una risorsa. Quando l'operazione non agisce su una risorsa designata oppure quando concedi l'autorizzazione per eseguire l'operazione su tutte le risorse, il valore della risorsa nella policy è un carattere jolly (\*). Per molte operazioni, puoi limitare le risorse che un utente può modificare specificando il nome della risorsa Amazon (ARN) di una risorsa o uno schema ARN che corrisponde a più risorse.

Per tipo di risorsa, la struttura generale di come limitare l'ambito di un'azione è la seguente:

- Funzioni: le operazioni alla base di una funzione possono essere limitate a una funzione specifica in base alla funzione, alla versione o all'ARN dell'alias.
- Strumenti di mappatura dell'origine degli eventi: le azioni possono essere limitate a risorse specifiche dello strumento di mappatura dell'origine degli eventi tramite ARN. Gli strumenti di mappatura dell'origine degli eventi sono sempre associati a una funzione. È inoltre possibile utilizzare la condizione `lambda:FunctionArn` per limitare le operazioni in base alla funzione associata.

- Livelli: le operazioni che riguardano l'utilizzo e le autorizzazioni relative ai livelli agiscono su una versione di un livello.
- Configurazione della firma del codice: le azioni possono essere limitate a risorse di configurazione di firma del codice specifiche tramite ARN.
- Tag: utilizza condizioni standard per i tag. Per ulteriori informazioni, consulta [the section called "Attribute-based controllo degli accessi"](#).

Per limitare le autorizzazioni in base alla risorsa, specifica la risorsa in base all'ARN.

Formato ARN della risorsa Lambda

- Funzione – `arn:aws:lambda:us-west-2:123456789012:function:my-function`
- Versione funzione – `arn:aws:lambda:us-west-2:123456789012:function:my-function:1`
- Alias funzione – `arn:aws:lambda:us-west-2:123456789012:function:my-function:TEST`
- Esecuzione duratura — `arn:aws:lambda:us-west-2:123456789012:function:my-function:1/durable-execution/execution-name/execution-id`
- Mappatura delle origini eventi – `arn:aws:lambda:us-west-2:123456789012:event-source-mapping:fa123456-14a1-4fd2-9fec-83de64ad683de6d47`
- Livello – `arn:aws:lambda:us-west-2:123456789012:layer:my-layer`
- Versione livello – `arn:aws:lambda:us-west-2:123456789012:layer:my-layer:1`
- Configurazione della firma del codice: `arn:aws:lambda:us-west-2:123456789012:code-signing-config:my-csc`

Ad esempio, la seguente politica consente a un utente di Account AWS 123456789012 richiamare una funzione denominata my-function nella regione Stati Uniti occidentali (Oregon) AWS .

Example Invocare la policy di una funzione

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
    "Sid": "Invoke",
    "Effect": "Allow",
    "Action": [
        "lambda:InvokeFunction"
    ],
    "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-  
function"
  }
]
```

Questo è un caso speciale in cui l'identificatore dell'operazione (`lambda:InvokeFunction`) differisce dall'operazione API ([Invoke](#)). Per altre operazioni, l'identificatore dell'operazione è il nome dell'operazione preceduto dal prefisso `lambda:`.

## Sections

- [Informazioni sulla sezione Condizione nelle policy](#)
- [Funzioni di riferimento nella sezione Risorse delle policy](#)
- [Riferimento alle esecuzioni durevoli nella sezione Risorse delle policy](#)
- [Comportamenti di azioni e funzioni IAM supportati](#)

## Informazioni sulla sezione Condizione nelle policy

Le condizioni sono un elemento opzionale della policy che applica una logica aggiuntiva per stabilire se è consentita un'operazione. Oltre alle [condizioni](#) comuni supportate da tutte le operazioni, Lambda definisce i tipi di condizione che è possibile utilizzare per limitare i valori dei parametri aggiuntivi su alcune operazioni.

Ad esempio, la condizione `lambda:Principal` consente di limitare il servizio o un account al quale un utente può concedere l'accesso a una chiamata su una [policy basata sulle risorse](#) della funzione. La policy seguente consente a un utente di concedere l'autorizzazione agli argomenti di Amazon Simple Notification Service (Amazon SNS) per richiamare una funzione denominata `test`.

## Example Gestione delle autorizzazioni di una policy della funzione

### JSON

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Sid": "ManageFunctionPolicy",
    "Effect": "Allow",
    "Action": [
      "lambda:AddPermission",
      "lambda:RemovePermission"
    ],
    "Resource": "arn:aws:lambda:us-west-2:123456789012:function:test:*",
    "Condition": {
      "StringEquals": {
        "lambda:Principal": "sns.amazonaws.com"
      }
    }
  }
]
```

La condizione richiede che il principale sia Amazon SNS e non un altro servizio o account. Il modello di risorsa richiede che il nome della funzione sia `test` e includa un numero di versione o un alias. Ad esempio, `test:v1`.

Per ulteriori informazioni su risorse e condizioni per Lambda e altri AWS servizi, consulta [Azioni, risorse e chiavi di condizione per AWS i servizi](#) nel Service Authorization Reference.

Funzioni di riferimento nella sezione Risorse delle policy

Puoi fare riferimento a una funzione Lambda in un rendiconto sulla policy utilizzando un Amazon Resource Name (ARN). Il formato dell'ARN di una funzione dipende dal fatto che si stia facendo riferimento all'intera funzione (non qualificato) o a una [versione](#) di funzione o a un [alias](#) (qualificato).

Quando effettuano chiamate all'API Lambda, gli utenti possono specificare una versione o un alias passando un ARN di versione o un alias ARN nel [GetFunction](#) `FunctionName` parametro o impostando un valore nel parametro. [GetFunction](#) `Qualifier` Lambda prende decisioni di autorizzazione confrontando l'elemento risorsa nella policy IAM con `FunctionName` e `Qualifier` passati nelle chiamate API. Se c'è un errore, Lambda nega la richiesta.

Sia che si stia consentendo o negando un'operazione sulla funzione, è necessario utilizzare i tipi ARN della funzione corretti nell'istruzione di policy per ottenere i risultati previsti. Ad esempio, se la policy

fa riferimento all'ARN non qualificato, Lambda accetta le richieste che fanno riferimento all'ARN non qualificato ma nega le richieste che fanno riferimento a un ARN qualificato.

### Note

Non puoi utilizzare un carattere jolly (\*) per la corrispondenza con l'ID account. Per informazioni sulla sintassi accettata, consulta [Documentazione di riferimento sulle policy JSON per IAM](#) nella Guida per l'utente di IAM.

Example permettere la chiamata di un ARN non qualificato

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction"
    }
  ]
}
```

Se la policy fa riferimento a uno specifico ARN qualificato, Lambda accetta le richieste che fanno riferimento a quell'ARN ma nega le richieste che fanno riferimento all'ARN non qualificato o a un ARN qualificato diverso, ad esempio `myFunction:2`.

Example permettere la chiamata di un ARN qualificato specifico

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
    "Action": "lambda:InvokeFunction",
    "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction:1"
  }
]
```

Se tua policy fa riferimento a un ARN qualificato utilizzando `:*`, Lambda accetta qualsiasi ARN qualificato ma nega le richieste che fanno riferimento all'ARN non qualificato.

Example permettere la chiamata di un ARN qualificato

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction:*"
    }
  ]
}
```

Se la tua policy fa riferimento a qualsiasi ARN che utilizza `*`, Lambda accetta qualsiasi ARN qualificato o non qualificato.

Example permettere la chiamata di un ARN qualificato o non qualificato

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
```



```

        "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction*"
    }
]
}

```

Riferimento alle esecuzioni durevoli nella sezione Risorse delle policy

Un'esecuzione durevole è una risorsa secondaria della versione di una funzione. Per la struttura dell'ARN a esecuzione durevole e le linee guida politiche, vedere. [the section called “Autorizzazioni IAM per funzioni durevoli”](#)

È possibile richiamare funzioni durevoli solo con un nome di funzione qualificato o ARN, pertanto le politiche che consentono l'invocazione di funzioni durevoli devono corrispondere agli ARN qualificati. Per ulteriori informazioni, consulta [the section called “ARNs Requisito qualificato”](#).

Se la tua policy fa riferimento alla funzione non qualificata ARN, Lambda nega le richieste di azioni di esecuzione durature. L'ARN non qualificato non corrisponde mai a un ARN di esecuzione duratura.

Example Politica che non garantisce l'accesso a esecuzioni durature

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:GetDurableExecution",
      "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myDurableFunction"
    }
  ]
}

```

Se la tua policy fa riferimento alla funzione ARN with: \*, Lambda accetta richieste di esecuzioni durevoli di ogni versione.

Example Consentire l'accesso a esecuzioni durevoli di qualsiasi versione

```

{
  "Version": "2012-10-17",

```

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": "lambda:GetDurableExecution",
    "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myDurableFunction:*"
  }
]
```

Se la tua policy fa riferimento alla funzione ARN `with:1/durable-execution/*`, Lambda accetta richieste di esecuzioni durevoli solo della versione 1.

Example Consentire l'accesso a esecuzioni durature di una versione specifica

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:GetDurableExecution",
      "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myDurableFunction:1/durable-execution/*"
    }
  ]
}
```

 I pattern di alias non corrispondono mai agli ARN di esecuzione duratura

I nomi alias non compaiono mai negli ARN a esecuzione durevole. Un modello di risorse come `my-function:prod/durable-execution/*` non corrisponde mai a un'esecuzione duratura perché Lambda risolve l'alias in un numero di versione quando la funzione viene eseguita.

Lambda autorizza l'`lambda:ListDurableExecutionsByFunction` azione contro la funzione ARN derivante dalla richiesta, non un ARN di esecuzione durevole. Le seguenti regole si applicano alla corrispondenza degli ARN:

- Una politica che concede la funzione non qualificata ARN soddisfa solo le richieste che utilizzano il nome della funzione non qualificata.

- Una policy che concede un ARN qualificato corrisponde solo alle richieste che utilizzano un nome di funzione qualificato.

Example Consente l'elenco delle esecuzioni durevoli con entrambe le forme di nome

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:ListDurableExecutionsByFunction",
      "Resource": [
        "arn:aws:lambda:us-west-2:123456789012:function:myDurableFunction",
        "arn:aws:lambda:us-west-2:123456789012:function:myDurableFunction:*"
      ]
    }
  ]
}
```

### Comportamenti di azioni e funzioni IAM supportati

Le azioni definiscono cosa può essere consentito tramite le policy IAM. Per un elenco delle operazioni supportate in Lambda, consulta [Operazioni, risorse e chiavi di condizione per AWS Lambda](#) in Service Authorization Reference. Nella maggior parte dei casi, quando un'azione IAM consente un'operazione API Lambda, il nome dell'azione IAM è lo stesso dell'operazione API Lambda, con le seguenti eccezioni:

| Azione API                           | Azione IAM             |
|--------------------------------------|------------------------|
| <a href="#">Invoke</a>               | lambda:InvokeFunction  |
| <a href="#">GetLayerVersion</a>      | lambda:GetLayerVersion |
| <a href="#">GetLayerVersionByArn</a> |                        |

Oltre alle risorse e alle condizioni definite in [Service Authorization Reference](#), Lambda supporta le seguenti risorse e condizioni per determinate azioni. Molte di queste sono correlate alle funzioni di riferimento nella sezione delle risorse delle policy. Le operazioni alla base di una funzione possono

essere limitate a una funzione specifica in base alla funzione, alla versione o all'ARN dell'alias, come descritto nella tabella riportata di seguito.

| Azione                                                                         | Risorsa                                 | Condizione                               |
|--------------------------------------------------------------------------------|-----------------------------------------|------------------------------------------|
| <a href="#">AddPermission</a>                                                  | Versione funzione<br><br>Alias funzione | N/A                                      |
| <a href="#">RemovePermission</a>                                               |                                         |                                          |
| <a href="#">Invoca</a> (Autorizzazione:)<br><code>lambda:InvokeFunction</code> |                                         |                                          |
| <a href="#">ListDurableExecutionsByFunction</a>                                | Versione funzione                       | N/A                                      |
| <a href="#">UpdateFunctionConfiguration</a>                                    | N/A                                     | <code>lambda:CodeSigningConfigArn</code> |
| <a href="#">CreateFunctionUrlConfig</a>                                        | Alias funzione                          | N/A                                      |
| <a href="#">DeleteFunctionUrlConfig</a>                                        |                                         |                                          |
| <a href="#">GetFunctionUrlConfig</a>                                           |                                         |                                          |
| <a href="#">UpdateFunctionUrlConfig</a>                                        |                                         |                                          |

## Sicurezza in AWS Lambda

La sicurezza del cloud AWS è la massima priorità. Come AWS cliente, puoi beneficiare di un data center e di un'architettura di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza.

La sicurezza è una responsabilità condivisa tra AWS e voi. Il [modello di responsabilità condivisa](#) descrive questo come sicurezza del cloud e sicurezza nel cloud:

- Sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che gira Servizi AWS nel AWS cloud. AWS ti fornisce anche servizi che puoi utilizzare in sicurezza. Third-party gli auditor testano e verificano regolarmente l'efficacia della nostra sicurezza nell'ambito dei programmi di

[AWS conformità](#). Per informazioni sui programmi di conformità a cui si applicano AWS Lambda, consulta [Servizi AWS la sezione Scope by Compliance Program](#).

- Sicurezza nel cloud: la tua responsabilità è determinata dal servizio AWS che utilizzi. Inoltre, sei responsabile anche di altri fattori, tra cui la riservatezza dei dati, i requisiti dell'azienda e le leggi e le normative applicabili.

Questa documentazione consente di comprendere come applicare il modello di responsabilità condivisa quando si usa Lambda. I seguenti argomenti illustrano come configurare Lambda per soddisfare gli obiettivi di sicurezza e conformità. Imparerai anche a utilizzarne altri Servizi AWS che ti aiutino a monitorare e proteggere le tue risorse Lambda. Gli argomenti includono la convalida della [conformità](#), la protezione [dei dati](#), la gestione delle [identità e degli accessi](#), gli endpoint [VPC](#) dell'interfaccia e la resilienza. [???](#)

Per maggiori informazioni sull'applicazione dei principi di sicurezza alle applicazioni Lambda, consulta [Sicurezza](#) in Serverless Land.

## Argomenti

- [Protezione dei dati in AWS Lambda](#)
- [Utilizzo di ruoli collegati ai servizi per Lambda](#)
- [Identity and Access Management per AWS Lambda](#)
- [Creare una strategia di governance per le funzioni e i livelli Lambda](#)
- [Convalida della conformità per AWS Lambda](#)
- [Resilienza in AWS Lambda](#)
- [Sicurezza dell'infrastruttura in AWS Lambda](#)
- [Protezione dei carichi di lavoro con endpoint pubblici](#)
- [Utilizzo della firma del codice per verificare l'integrità del codice con Lambda](#)

## Protezione dei dati in AWS Lambda

Il modello di [responsabilità AWS condivisa Il modello](#) di si applica alla protezione dei dati in AWS Lambda. Come descritto in questo modello, AWS è responsabile della protezione dell'infrastruttura globale che gestisce tutti i Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. L'utente è inoltre responsabile della configurazione della protezione e delle attività di gestione per i Servizi AWS utilizzati. Per ulteriori informazioni sulla privacy dei dati, consulta

[Domande frequenti sulla privacy dei dati](#) . Per informazioni sulla protezione dei dati in Europa, consulta il [General Data Protection Regulation \(GDPR\) Center](#).

Ai fini della protezione dei dati, ti consigliamo di proteggere Account AWS le credenziali e di configurare i singoli utenti con Centro identità AWS IAM or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- SSL/TLS Da utilizzare per comunicare con AWS le risorse. È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti con AWS CloudTrail. Per informazioni sull'utilizzo dei CloudTrail percorsi per acquisire AWS le attività, consulta [Lavorare con i CloudTrail percorsi](#) nella Guida per l'AWS CloudTrail utente.
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-3 per accedere AWS tramite un'interfaccia a riga di comando o un'API, usa un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il [Federal Information Processing Standard \(FIPS\) 140-3](#).

Ti consigliamo di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando lavori con Lambda o altro Servizi AWS utilizzando la console, l'API o AWS gli AWS CLI SDK. I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per la fatturazione o i log di diagnostica. Quando si fornisce un URL a un server esterno, suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la richiesta al server.

## Sections

- [Crittografia dei dati in transito](#)
- [Crittografia dei dati a riposo per AWS Lambda](#)

## Crittografia dei dati in transito

Gli endpoint API di Lambda supportano solo connessioni protette tramite HTTPS. Quando gestisci le risorse Lambda con Console di gestione AWS, AWS SDK o l'API Lambda, tutte le comunicazioni vengono crittografate con Transport Layer Security (TLS). Per un elenco completo degli endpoint API, consultare [Regioni ed endpoint AWS](#) in Riferimenti generali di AWS.

Quando si [connette la funzione a un file system](#), Lambda utilizza Crittografia in transito per tutte le connessioni. Per ulteriori informazioni, consulta [Crittografia dati in Amazon EFS](#) nella Amazon Elastic File System User Guide.

Quando si utilizzano le [variabili di ambiente](#), è possibile abilitare gli helper della crittografia della console per utilizzare il file crittografato lato client per proteggere le variabili di ambiente in transito. Per ulteriori informazioni, consulta [Protezione delle variabili di ambiente Lambda](#).

## Crittografia dei dati a riposo per AWS Lambda

AWS Lambda fornisce sempre la crittografia a riposo per le seguenti risorse:

- Variabili di ambiente
- I file che vengono caricati in Lambda, inclusi i pacchetti di implementazione e gli archivi dei livelli
- Oggetti dei criteri del filtro dello strumento di mappatura dell'origine degli eventi
- Dati di esecuzione durevoli

### Note

Per le funzioni e i livelli che utilizzano l'[archiviazione del codice S3 autogestita](#), Lambda non archivia una copia del codice sorgente. Il codice rimane nel bucket S3 e la crittografia inattiva del codice sorgente viene gestita dalla configurazione del bucket S3. Lambda utilizzerà KMSKeyArn per crittografare la versione decompressa del pacchetto utilizzata da Lambda per richiamare la funzione. Per ulteriori informazioni, consulta la sezione [Protezione dei dati con la crittografia lato server](#) nella Guida per l'utente di Amazon Simple Storage Service.

Facoltativamente, puoi configurare Lambda per utilizzare una chiave gestita dal cliente per crittografare le variabili di ambiente, i pacchetti di distribuzione.zip, gli oggetti dei criteri di filtro e i dati di esecuzione durevoli.

Amazon CloudWatch Logs crittografa AWS X-Ray anche i dati per impostazione predefinita e può essere configurato per utilizzare una chiave gestita dal cliente. [Per i dettagli, consulta Crittografia i dati di registro in CloudWatch Logs e Data protection in. AWS X-Ray](#)

## Monitoraggio delle chiavi di crittografia per Lambda

Quando utilizzi una chiave gestita AWS KMS dal cliente con Lambda, puoi usare. [AWS CloudTrail](#) Gli esempi seguenti sono CloudTrail gli eventi e GenerateDataKey le chiamate effettuate da Lambda per Decrypt accedere ai dati crittografati dalla chiave gestita dal cliente. DescribeKey

Per CloudTrail esempi specifici relativi ai dati di esecuzione durevoli, vedi [Monitoraggio delle chiavi KMS per funzioni durature](#).

## Decrypt

Se hai utilizzato una chiave gestita AWS KMS dal cliente per crittografare l'oggetto dei [criteri di filtro](#), Lambda invia Decrypt una richiesta per tuo conto quando tenti di accedervi in testo semplice (ad esempio, da una chiamata). ListEventSourceMappings L'evento di esempio seguente registra l'operazione Decrypt:

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAI23456789EXAMPLE:example",
    "arn": "arn:aws:sts::123456789012:assumed-role/role-name/example",
    "accountId": "123456789012",
    "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAI23456789EXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/role-name",
        "accountId": "123456789012",
        "userName": "role-name"
      },
      "attributes": {
        "creationDate": "2024-05-30T00:45:23Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "invokedBy": "lambda.amazonaws.com"
```



```

    },
    "eventTime": "2024-05-30T01:05:46Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "eu-west-1",
    "sourceIPAddress": "lambda.amazonaws.com",
    "userAgent": "lambda.amazonaws.com",
    "requestParameters": {
      "keyId": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
      "encryptionContext": {
        "aws-crypto-public-key": "ABCD
+7876787678+CDEFGHIJKL/888666888999888555444111555222888333111==",
        "aws:lambda:EventSourceArn": "arn:aws:sqs:eu-west-1:123456789012:sample-
source",
        "aws:lambda:FunctionArn": "arn:aws:lambda:eu-
west-1:123456789012:function:sample-function"
      },
      "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbbb",
    "readOnly": true,
    "resources": [
      {
        "accountId": "AWS Internal",
        "type": "AWS::KMS::Key",
        "ARN": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
      }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "sessionCredentialFromConsole": "true"
  }

```

## DescribeKey

Se hai utilizzato una chiave gestita AWS KMS dal cliente per crittografare l'oggetto dei [criteri di filtro](#), Lambda invia DescribeKey una richiesta per tuo conto quando tenti di accedervi (ad

esempio, da `GetEventSourceMapping` una chiamata). L'evento di esempio seguente registra l'operazione `DescribeKey`:

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAI23456789EXAMPLE:example",
    "arn": "arn:aws:sts::123456789012:assumed-role/role-name/example",
    "accountId": "123456789012",
    "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAI23456789EXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/role-name",
        "accountId": "123456789012",
        "userName": "role-name"
      },
      "attributes": {
        "creationDate": "2024-05-30T00:45:23Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2024-05-30T01:09:40Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "DescribeKey",
  "awsRegion": "eu-west-1",
  "sourceIPAddress": "54.240.197.238",
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36",
  "requestParameters": {
    "keyId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
  },
  "responseElements": null,
  "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
  "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbbb",
  "readOnly": true,
  "resources": [
    {
      "accountId": "AWS Internal",
      "type": "AWS::KMS::Key",

```

```

      "ARN": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "123456789012",
  "eventCategory": "Management",
  "tlsDetails": {
    "tlsVersion": "TLSv1.3",
    "cipherSuite": "TLS_AES_256_GCM_SHA384",
    "clientProvidedHostHeader": "kms.eu-west-1.amazonaws.com"
  },
  "sessionCredentialFromConsole": "true"
}

```

## GenerateDataKey

Quando utilizzi una chiave gestita AWS KMS dal cliente per crittografare l'oggetto dei [criteri di filtro](#) in una `UpdateEventSourceMapping` chiamata `CreateEventSourceMapping` o, Lambda invia `GenerateDataKey` una richiesta per tuo conto per generare una chiave dati per crittografare i criteri di filtro (crittografia a busta). L'evento di esempio seguente registra l'operazione `GenerateDataKey`:

```

{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAI23456789EXAMPLE:example",
    "arn": "arn:aws:sts::123456789012:assumed-role/role-name/example",
    "accountId": "123456789012",
    "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAI23456789EXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/role-name",
        "accountId": "123456789012",
        "userName": "role-name"
      },
      "attributes": {
        "creationDate": "2024-05-30T00:06:07Z",
        "mfaAuthenticated": "false"
      }
    }
  }
}

```

```

    }
  },
  "invokedBy": "lambda.amazonaws.com"
},
"eventTime": "2024-05-30T01:04:18Z",
"eventSource": "kms.amazonaws.com",
"eventName": "GenerateDataKey",
"awsRegion": "eu-west-1",
"sourceIPAddress": "lambda.amazonaws.com",
"userAgent": "lambda.amazonaws.com",
"requestParameters": {
  "numberOfBytes": 32,
  "keyId": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "encryptionContext": {
    "aws-crypto-public-key": "ABCD
+7876787678+CDEFGHIJKL/888666888999888555444111555222888333111==",
    "aws:lambda:EventSourceArn": "arn:aws:sqs:eu-west-1:123456789012:sample-
source",
    "aws:lambda:FunctionArn": "arn:aws:lambda:eu-
west-1:123456789012:function:sample-function"
  },
},
"responseElements": null,
"requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEEaaaaa",
"eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEebbbbb",
"readOnly": true,
"resources": [
  {
    "accountId": "AWS Internal",
    "type": "AWS::KMS::Key",
    "ARN": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
  }
],
"eventType": "AwsApiCall",
"managementEvent": true,
"recipientAccountId": "123456789012",
"eventCategory": "Management"
}

```

## Utilizzo di ruoli collegati ai servizi per Lambda

Lambda utilizza ruoli collegati ai servizi AWS Identity and Access Management (IAM). [https://docs.aws.amazon.com/IAM/latest/UserGuide/id\\_roles\\_terms-and-concepts.html#iam-term-service-linked-role](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_terms-and-concepts.html#iam-term-service-linked-role) Un ruolo collegato ai servizi è un tipo unico di ruolo IAM collegato direttamente a Lambda. Service-linked i ruoli sono predefiniti da Lambda e includono le autorizzazioni necessarie al servizio per chiamare altri servizi per conto dell'utente. AWS

Lambda definisce le autorizzazioni dei suoi ruoli collegati ai servizi e solo Lambda può assumerne i ruoli. Le autorizzazioni definite includono la policy di attendibilità e la policy delle autorizzazioni che non può essere allegata a nessun'altra entità IAM.

È possibile eliminare un ruolo collegato al servizio solo dopo avere eliminato le risorse correlate. Questo protegge le tue risorse Lambda perché non puoi rimuovere inavvertitamente il permesso di accesso alle risorse.

Per informazioni su altri servizi che supportano i ruoli collegati ai servizi, consulta [AWS i servizi che funzionano con IAM](#) e cerca i servizi che hanno Sì nella colonna dei ruoli. Service-linked Scegliere Sì in corrispondenza di un link per visualizzare la documentazione relativa al ruolo collegato al servizio per tale servizio.

### Service-linked autorizzazioni di ruolo per Lambda

Lambda utilizza il ruolo collegato al servizio denominato `AWSServiceRoleForLambda`. Ai fini dell'assunzione del ruolo, il ruolo collegato al servizio considera attendibili i seguenti servizi:

- `lambda.amazonaws.com`

La politica di autorizzazione del ruolo denominata `AWSLambdaServiceRolePolicy` consente a Lambda di completare le seguenti azioni sulle risorse specificate:

- Azione: `ec2:TerminateInstances` attivata `arn:aws:ec2:*:*:instance/*` con la condizione uguale `ec2:ManagedResourceOperator` `scaler.lambda.amazonaws.com`
- Azione: `ec2:DescribeInstanceStatus` e `ec2:DescribeInstances` su \*

Per consentire a utenti, gruppi o ruoli di creare, modificare o eliminare un ruolo orientato ai servizi, devi configurare le autorizzazioni. Per ulteriori informazioni, consulta le autorizzazioni dei [Service-linked ruoli](#) nella IAM User Guide.

Per gli aggiornamenti delle policy gestite, consulta Policy gestite [da Lambda](#).

## Creazione di un ruolo collegato al servizio per Lambda

Non hai bisogno di creare manualmente un ruolo collegato ai servizi. Quando crei un fornitore di capacità Lambda nell'API Console di gestione AWS, nella o nell' AWS API AWS CLI, Lambda crea il ruolo collegato al servizio per te.

Se elimini questo ruolo collegato al servizio, è possibile ricrearlo seguendo lo stesso processo utilizzato per ricreare il ruolo nell'account. Quando crei un fornitore di capacità Lambda, Lambda crea nuovamente il ruolo collegato al servizio per te.

Puoi anche utilizzare la console IAM per creare un ruolo collegato al servizio con il caso d'uso. `AWSServiceRoleForLambda` Nell'API AWS CLI o nell' AWS API, crea un ruolo collegato al servizio con il nome del servizio. `lambda.amazonaws.com` Per ulteriori informazioni, consulta [Creazione di un ruolo collegato ai servizi](#) nella Guida per l'utente IAM. Se elimini il ruolo collegato ai servizi, è possibile utilizzare lo stesso processo per crearlo nuovamente.

## Modifica di un ruolo collegato al servizio per Lambda

Lambda non consente di modificare il ruolo collegato al servizio. `AWSServiceRoleForLambda` Dopo avere creato un ruolo collegato al servizio, non sarà possibile modificarne il nome perché varie entità potrebbero farvi riferimento. È possibile tuttavia modificarne la descrizione utilizzando IAM. Per ulteriori informazioni, consulta [Modifica di un ruolo collegato al servizio](#) nella Guida per l'utente di IAM.

## Eliminazione di un ruolo collegato al servizio per Lambda

Se non è più necessario utilizzare una caratteristica o un servizio che richiede un ruolo collegato ai servizi, ti consigliamo di eliminare quel ruolo. In questo modo non sarà più presente un'entità non utilizzata che non viene monitorata e gestita attivamente. Tuttavia, è necessario effettuare la pulizia delle risorse associate al ruolo collegato ai servizi prima di poterlo eliminare manualmente.

### Note

Se il servizio Lambda utilizza il ruolo quando si tenta di eliminare le risorse, l'eliminazione potrebbe non riuscire. In questo caso, attendi alcuni minuti e quindi ripeti l'operazione.

## Per eliminare le risorse Lambda utilizzate da AWSServiceRoleForLambda

1. Rimuovi tutti i fornitori di capacità Lambda dal tuo account. Puoi farlo utilizzando la console Lambda, la CLI o l'API.
2. Verifica che nel tuo account non rimanga alcun fornitore di capacità Lambda prima di tentare di eliminare il ruolo collegato al servizio.

## Per eliminare manualmente il ruolo collegato ai servizi mediante IAM

Usa la console IAM AWS CLI, l'API o l' AWS API per eliminare il ruolo collegato al servizio.

AWSServiceRoleForLambda Per ulteriori informazioni, consulta [Eliminazione del ruolo collegato ai servizi](#) nella Guida per l'utente IAM.

## Regioni supportate per i ruoli collegati ai servizi Lambda

Lambda non supporta l'utilizzo di ruoli collegati ai servizi in tutte le regioni in cui il servizio è disponibile. AWSServiceRoleForLambda è supportato nelle seguenti regioni.

| Nome della Regione                                  | Identità della Regione | Supporto in Lambda |
|-----------------------------------------------------|------------------------|--------------------|
| Stati Uniti orientali (Virginia settentrionale)     | us-east-1              | Sì                 |
| Stati Uniti orientali (Ohio)                        | us-east-2              | Sì                 |
| Stati Uniti occidentali (California settentrionale) | us-west-1              | Sì                 |
| Stati Uniti occidentali (Oregon)                    | us-west-2              | Sì                 |
| Africa (Città del Capo)                             | af-south-1             | No                 |
| Asia Pacific (Hong Kong)                            | ap-east-1              | Sì                 |
| Asia Pacifico (Giacarta)                            | ap-southeast-3         | Sì                 |
| Asia Pacifico (Bangkok)                             | ap-southeast-7         | Sì                 |
| Asia Pacifico (Mumbai)                              | ap-south-1             | Sì                 |

| Nome della Regione                  | Identità della Regione | Supporto in Lambda |
|-------------------------------------|------------------------|--------------------|
| Asia Pacifico (Osaka)               | ap-northeast-3         | No                 |
| Asia Pacific (Seoul)                | ap-northeast-2         | No                 |
| Asia Pacifico (Singapore)           | ap-southeast-1         | Sì                 |
| Asia Pacifico (Sydney)              | ap-southeast-2         | Sì                 |
| Asia Pacifico (Tokyo)               | ap-northeast-1         | Sì                 |
| Canada (Centrale)                   | ca-central-1           | No                 |
| Europa (Francoforte)                | eu-central-1           | Sì                 |
| Europa (Irlanda)                    | eu-west-1              | Sì                 |
| Europa (Londra)                     | eu-west-2              | Sì                 |
| Europe (Milan)                      | eu-south-1             | No                 |
| Europa (Parigi)                     | eu-west-3              | No                 |
| Europa (Stoccolma)                  | eu-north-1             | No                 |
| Medio Oriente (Bahrein)             | me-south-1             | No                 |
| Medio Oriente (Emirati Arabi Uniti) | me-central-1           | No                 |
| Sud America (San Paolo)             | sa-east-1              | No                 |
| AWS GovCloud (US-East)              | us-gov-east-1          | No                 |
| AWS GovCloud (US-West)              | us-gov-west-1          | No                 |

## Identity and Access Management per AWS Lambda

AWS Identity and Access Management (IAM) è uno strumento Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle AWS risorse. Gli amministratori IAM



controllano chi può essere autenticato (connesso) e autorizzato (avere le autorizzazioni) a utilizzare le risorse Lambda. IAM è un software Servizio AWS che puoi utilizzare senza costi aggiuntivi.

## Argomenti

- [Destinatari](#)
- [Autenticazione con identità](#)
- [Gestione dell'accesso tramite policy](#)
- [Come AWS Lambda funziona con IAM](#)
- [Esempi di policy basate sull'identità per AWS Lambda](#)
- [AWS politiche gestite per AWS Lambda](#)
- [Risoluzione dei problemi AWS Lambda di identità e accesso](#)

## Destinatari

Il modo in cui utilizzi AWS Identity and Access Management (IAM) varia in base al tuo ruolo:

- Utente del servizio: richiedi le autorizzazioni all'amministratore se non riesci ad accedere alle funzionalità (consulta [Risoluzione dei problemi AWS Lambda di identità e accesso](#))
- Amministratore del servizio: determina l'accesso degli utenti e invia le richieste di autorizzazione (consulta [Come AWS Lambda funziona con IAM](#))
- Amministratore IAM: scrivi policy per gestire l'accesso (consulta [Esempi di policy basate sull'identità per AWS Lambda](#))

## Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi autenticarti come utente IAM o assumendo un ruolo IAM. Utente root dell'account AWS

Puoi accedere come identità federata utilizzando credenziali provenienti da una fonte di identità come Centro identità AWS IAM (IAM Identity Center), autenticazione Single Sign-On o credenziali. Google/Facebook Per ulteriori informazioni sull'accesso, consulta [Come accedere all' Account AWS](#) nella Guida per l'utente di Accedi ad AWS .

Per l'accesso programmatico, AWS fornisce un SDK e una CLI per firmare crittograficamente le richieste. Per ulteriori informazioni, consulta [AWS Signature Version 4 per le richieste API](#) nella Guida per l'utente di IAM.

## Account AWS utente root

Quando si crea un Account AWS, si inizia con un'identità di accesso denominata utente Account AWS root che ha accesso completo a tutte Servizi AWS le risorse. Consigliamo vivamente di non utilizzare l'utente root per le attività quotidiane. Per le attività che richiedono le credenziali dell'utente root, consulta [Attività che richiedono le credenziali dell'utente root](#) nella Guida per l'utente IAM.

## Identità federata

Come procedura ottimale, richiedi agli utenti umani di utilizzare la federazione con un provider di identità per accedere Servizi AWS utilizzando credenziali temporanee.

Un'identità federata è un utente della directory aziendale, del provider di identità Web o Servizio di directory che accede Servizi AWS utilizzando le credenziali di una fonte di identità. Le identità federate assumono ruoli che forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, si consiglia di utilizzare Centro identità AWS IAM. Per ulteriori informazioni, consulta [Che cos'è il Centro identità IAM?](#) nella Guida per l'utente di Centro identità AWS IAM .

## Utenti e gruppi IAM

Un [utente IAM](#) è una identità che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ti consigliamo di utilizzare credenziali temporanee invece di utenti IAM con credenziali a lungo termine. Per ulteriori informazioni, consulta [Richiedere agli utenti umani di utilizzare la federazione con un provider di identità per accedere AWS utilizzando credenziali temporanee](#) nella Guida per l'utente IAM.

Un [gruppo IAM](#) specifica una raccolta di utenti IAM e semplifica la gestione delle autorizzazioni per gestire gruppi di utenti di grandi dimensioni. Per ulteriori informazioni, consulta [Casi d'uso per utenti IAM](#) nella Guida per l'utente di IAM.

## Ruoli IAM

Un [ruolo IAM](#) è un'identità con autorizzazioni specifiche che fornisce credenziali temporanee. Puoi assumere un ruolo [passando da un ruolo utente a un ruolo IAM \(console\)](#) o chiamando un'operazione AWS CLI o AWS API. Per ulteriori informazioni, consulta [Metodi per assumere un ruolo](#) nella Guida per l'utente di IAM.

I ruoli IAM sono utili per l'accesso degli utenti federati, le autorizzazioni utente IAM temporanee, l'accesso multi-account, l'accesso multi-servizio e le applicazioni in esecuzione su Amazon EC2. Per

maggiori informazioni, consultare [Accesso a risorse multi-account in IAM](#) nella Guida per l'utente IAM.

## Gestione dell'accesso tramite policy

Puoi controllare l'accesso AWS creando policy e associandole a AWS identità o risorse. Una policy definisce le autorizzazioni quando è associata a un'identità o a una risorsa. AWS valuta queste politiche quando un preside effettua una richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per maggiori informazioni sui documenti delle policy JSON, consulta [Panoramica delle policy JSON](#) nella Guida per l'utente IAM.

Utilizzando le policy, gli amministratori specificano chi ha accesso a cosa definendo quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Un amministratore IAM crea le policy IAM e le aggiunge ai ruoli, che gli utenti possono quindi assumere. Le policy IAM definiscono le autorizzazioni indipendentemente dal metodo utilizzato per eseguirle.

### Policy basate sull'identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile collegare a un'identità (utente, gruppo o ruolo). Tali policy controllano le operazioni autorizzate per l'identità, nonché le risorse e le condizioni in cui possono essere eseguite. Per informazioni su come creare una policy basata su identità, consultare [Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente](#) nella Guida per l'utente IAM.

Le policy basate su identità possono essere policy in linea (con embedding direttamente in una singola identità) o policy gestite (policy autonome collegate a più identità). Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta [Scegliere tra policy gestite e policy in linea](#) nella Guida per l'utente di IAM.

### Policy basate sulle risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Gli esempi includono le policy di trust dei ruoli IAM e le policy dei bucket di Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. In una policy basata sulle risorse è obbligatorio [specificare un'entità principale](#).

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non è possibile utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

## Altri tipi di policy

AWS supporta tipi di policy aggiuntivi che possono impostare le autorizzazioni massime concesse dai tipi di policy più comuni:

- Limiti delle autorizzazioni: imposta il numero massimo di autorizzazioni che una policy basata su identità ha la possibilità di concedere a un'entità IAM. Per ulteriori informazioni, consulta [Limiti delle autorizzazioni per le entità IAM](#) nella Guida per l'utente di IAM.
- Politiche di controllo del servizio (SCPs): specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa in AWS Organizations. Per ulteriori informazioni, consultare [Policy di controllo dei servizi](#) nella Guida per l'utente di AWS Organizations.
- Politiche di controllo delle risorse (RCPs): imposta le autorizzazioni massime disponibili per le risorse nei tuoi account. Per ulteriori informazioni, consulta [Politiche di controllo delle risorse \(RCPs\)](#) nella Guida per l'AWS Organizations utente.
- Policy di sessione: policy avanzate passate come parametro quando si crea una sessione temporanea per un ruolo o un utente federato. Per maggiori informazioni, consultare [Policy di sessione](#) nella Guida per l'utente IAM.

## Più tipi di policy

Quando a una richiesta si applicano più tipi di policy, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si determina se consentire o meno una richiesta quando sono coinvolti più tipi di policy, consulta [Logica di valutazione delle policy](#) nella IAM User Guide.

## Come AWS Lambda funziona con IAM

Prima di utilizzare IAM per gestire l'accesso a Lambda, scopri quali funzionalità di IAM sono disponibili per l'uso con .

| Funzionalità IAM                             | Supporto Lambda |
|----------------------------------------------|-----------------|
| <a href="#">Policy basate sull'identità</a>  | Sì              |
| <a href="#">Policy basate su risorse</a>     | Sì              |
| <a href="#">Operazioni di policy</a>         | Sì              |
| <a href="#">Risorse relative alle policy</a> | Sì              |

| Funzionalità IAM                                                           | Supporto Lambda |
|----------------------------------------------------------------------------|-----------------|
| <a href="#">Chiavi di condizione della policy (specifica del servizio)</a> | Sì              |
| <a href="#">ACLs</a>                                                       | No              |
| <a href="#">ABAC (tag nelle policy)</a>                                    | Parziale        |
| <a href="#">Credenziali temporanee</a>                                     | Sì              |
| <a href="#">Inoltro delle sessioni di accesso (FAS)</a>                    | No              |
| <a href="#">Ruoli di servizio</a>                                          | Sì              |
| <a href="#">Ruoli collegati al servizio</a>                                | Parziale        |

Per avere una visione di alto livello di come Lambda e AWS altri servizi funzionano con la maggior parte delle funzionalità IAM, [AWS consulta i servizi che funzionano con](#) IAM nella IAM User Guide.

Policy basate sulle identità per Lambda

Supporta le policy basate sull'identità: sì

Le policy basate sull'identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruolo IAM). Tali policy definiscono le operazioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta [Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente](#) nella Guida per l'utente di IAM.

Con le policy basate sull'identità di IAM, è possibile specificare quali operazioni e risorse sono consentite o respinte, nonché le condizioni in base alle quali le operazioni sono consentite o respinte. Per informazioni su tutti gli elementi utilizzabili in una policy JSON, consulta [Guida di riferimento agli elementi delle policy JSON IAM](#) nella Guida per l'utente IAM.

Esempi di policy basate su identità per Lambda

Per visualizzare esempi di policy basate su identità Lambda, consulta [Esempi di policy basate sull'identità per AWS Lambda](#).

## Policy basate sulle risorse all'interno di Lambda

Supporta le policy basate sulle risorse: sì

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Esempi di policy basate sulle risorse sono le policy di attendibilità dei ruoli IAM e le policy di bucket Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. Quando è collegata a una risorsa, una policy definisce le operazioni che un principale può eseguire su tale risorsa e a quali condizioni. In una policy basata sulle risorse è obbligatorio [specificare un'entità principale](#). I principali possono includere account, utenti, ruoli, utenti federati o. Servizi AWS

Per consentire l'accesso multi-account, è possibile specificare un intero account o entità IAM in un altro account come entità principale in una policy basata sulle risorse. Per ulteriori informazioni, consulta [Accesso a risorse multi-account in IAM](#) nella Guida per l'utente IAM.

È possibile associare una policy basata su risorse a una funzione o a un livello Lambda. Questa policy definisce quali principali possono eseguire azioni sulla funzione o il livello.

Per informazioni su come collegare una policy basata su risorse a una funzione o un livello, consulta [Utilizzo di policy basate sulle risorse in Lambda](#).

### Operazioni di policy per Lambda

Supporta le operazioni di policy: sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento `Action` di una policy JSON descrive le operazioni che è possibile utilizzare per consentire o negare l'accesso in una policy. Includere le operazioni in una policy per concedere le autorizzazioni a eseguire l'operazione associata.

Per visualizzare un elenco di operazioni Lambda, consulta [Operazioni definite da AWS Lambda](#) nella Service Authorization Reference.

Le operazioni delle policy in Lambda utilizzano il seguente prefisso prima dell'operazione:

```
lambda
```

Per specificare più operazioni in una sola istruzione, occorre separarle con la virgola.

```
"Action": [  
    "lambda:action1",  
    "lambda:action2"  
]
```

Per visualizzare esempi di policy basate su identità Lambda, consulta [Esempi di policy basate sull'identità per AWS Lambda](#).

## Risorse di policy per Lambda

Supporta le risorse relative alle policy: sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento JSON `Resource` della policy specifica l'oggetto o gli oggetti ai quali si applica l'operazione. Come best practice, specifica una risorsa utilizzando il suo [nome della risorsa Amazon \(ARN\)](#). Per le azioni che non supportano le autorizzazioni a livello di risorsa, si utilizza un carattere jolly (\*) per indicare che l'istruzione si applica a tutte le risorse.

```
"Resource": "*"
```

Per visualizzare un elenco dei tipi di risorse Lambda e relativi ARNs, consulta Tipi di [risorse definiti da AWS Lambda](#) nel Service Authorization Reference. Per informazioni sulle operazioni con cui è possibile specificare l'ARN di ogni risorsa, consulta la sezione [Operazioni definite da AWS Lambda](#).

Per visualizzare esempi di policy basate su identità Lambda, consulta [Esempi di policy basate sull'identità per AWS Lambda](#).

## Chiavi di condizione delle policy per Lambda

Supporta le chiavi di condizione delle policy specifiche del servizio: sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento `Condition` specifica quando le istruzioni vengono eseguite in base a criteri definiti. È possibile compilare espressioni condizionali che utilizzano [operatori di condizione](#), ad esempio uguale a o minore di, per soddisfare la condizione nella policy con i valori nella richiesta. Per

visualizzare tutte le chiavi di condizione AWS globali, consulta le chiavi di [contesto delle condizioni AWS globali nella Guida](#) per l'utente IAM.

Per visualizzare un elenco di chiavi di condizione Lambda, consulta [Chiavi di condizione per AWS Lambda](#) in Service Authorization Reference. Per sapere con quali azioni e risorse puoi utilizzare una chiave di condizione, consulta [Actions defined by AWS Lambda](#).

Per visualizzare esempi di policy basate su identità Lambda, consulta [Esempi di policy basate sull'identità per AWS Lambda](#).

## ACLs a Lambda

Supporti ACLs: no

Le liste di controllo degli accessi (ACLs) controllano quali principali (membri dell'account, utenti o ruoli) dispongono delle autorizzazioni per accedere a una risorsa. ACLs sono simili alle politiche basate sulle risorse, sebbene non utilizzino il formato del documento di policy JSON.

## ABAC con Lambda

Supporta ABAC (tag nelle policy): parzialmente

Il controllo degli accessi basato su attributi (ABAC) è una strategia di autorizzazione che definisce le autorizzazioni in base agli attributi, chiamati tag. È possibile allegare tag a entità e AWS risorse IAM, quindi progettare politiche ABAC per consentire operazioni quando il tag del principale corrisponde al tag sulla risorsa.

Per controllare l'accesso basato su tag, fornire informazioni sui tag nell'[elemento condizione](#) di una policy utilizzando le chiavi di condizione `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`.

Se un servizio supporta tutte e tre le chiavi di condizione per ogni tipo di risorsa, il valore per il servizio è Sì. Se un servizio supporta tutte e tre le chiavi di condizione solo per alcuni tipi di risorsa, allora il valore sarà Parziale.

Per maggiori informazioni su ABAC, consulta [Definizione delle autorizzazioni con autorizzazione ABAC](#) nella Guida per l'utente di IAM. Per visualizzare un tutorial con i passaggi per l'impostazione di ABAC, consulta [Utilizzo del controllo degli accessi basato su attributi \(ABAC\)](#) nella Guida per l'utente di IAM.

Per ulteriori informazioni sul tagging delle risorse Lambda, consulta [Utilizzo del controllo degli accessi basato sugli attributi in Lambda](#).



## Utilizzo di credenziali temporanee con Lambda

Supporta le credenziali temporanee: sì

Le credenziali temporanee forniscono l'accesso a breve termine alle AWS risorse e vengono create automaticamente quando si utilizza la federazione o si cambia ruolo. AWS consiglia di generare dinamicamente credenziali temporanee anziché utilizzare chiavi di accesso a lungo termine. Per ulteriori informazioni, consulta [Credenziali di sicurezza temporanee in IAM](#) e [Servizi AWS compatibili con IAM](#) nella Guida per l'utente IAM.

## Inoltro delle sessioni di accesso per Lambda

Supporta l'inoltro delle sessioni di accesso (FAS): no

Le sessioni di accesso inoltrato (FAS) utilizzano le autorizzazioni del principale che chiama e, in combinazione con la richiesta Servizio AWS, Servizio AWS per effettuare richieste ai servizi downstream. Per i dettagli delle policy relative alle richieste FAS, consulta [Forward access sessions](#).

## Ruoli di servizio per Lambda

Supporta i ruoli di servizio: sì

Un ruolo di servizio è un [ruolo IAM](#) che un servizio assume per eseguire operazioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta [Create a role to delegate permissions to an Servizio AWS](#) nella Guida per l'utente IAM.

In Lambda, un ruolo di servizio è noto come [ruolo di esecuzione](#).

### Warning

La modifica delle autorizzazioni per un ruolo di servizio potrebbe compromettere la funzionalità di Lambda.

## Ruoli collegati ai servizi per Lambda

Supporta i ruoli legati ai servizi: Parziale

Un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un. Servizio AWS Il servizio può assumere il ruolo per eseguire un'operazione per tuo conto. I ruoli collegati al servizio vengono

visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati al servizio, ma non modificarle.

Lambda non dispone di ruoli collegati al servizio, a differenza di Lambda@Edge. Per ulteriori informazioni, consulta [Service-Linked Roles for Lambda @Edge nella](#) Amazon Developer Guide. CloudFront

Per ulteriori informazioni su come creare e gestire i ruoli collegati ai servizi, consulta [Servizi AWS supportati da IAM](#). Trova un servizio nella tabella che include un Yes nella colonna Service-linked role (Ruolo collegato ai servizi). Scegli il collegamento Sì per visualizzare la documentazione relativa al ruolo collegato ai servizi per tale servizio.

## Esempi di policy basate sull'identità per AWS Lambda

Per impostazione predefinita, gli utenti e i ruoli non sono autorizzati a creare o modificare le risorse Lambda. Per concedere agli utenti l'autorizzazione a eseguire azioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM.

Per informazioni su come creare una policy basata su identità IAM utilizzando questi documenti di policy JSON di esempio, consulta [Creazione di policy IAM \(console\)](#) nella Guida per l'utente di IAM.

Per informazioni dettagliate sulle azioni e sui tipi di risorse definiti da Lambda, incluso il formato di ARNs per ogni tipo di risorsa, consulta [Azioni, risorse e chiavi di condizione AWS Lambda](#) nel Service Authorization Reference.

### Argomenti

- [Best practice per le policy](#)
- [Uso della console Lambda](#)
- [Consentire agli utenti di visualizzare le loro autorizzazioni](#)

### Best practice per le policy

Le policy basate su identità determinano se qualcuno può creare, accedere o eliminare risorse Lambda nel tuo account. Queste azioni possono comportare costi aggiuntivi per l'Account AWS. Quando si creano o modificano policy basate sull'identità, seguire queste linee guida e raccomandazioni:

- Inizia con le policy AWS gestite e passa alle autorizzazioni con privilegi minimi: per iniziare a concedere autorizzazioni a utenti e carichi di lavoro, utilizza le policy AWS gestite che concedono

le autorizzazioni per molti casi d'uso comuni. Sono disponibili nel tuo Account AWS. Ti consigliamo di ridurre ulteriormente le autorizzazioni definendo politiche gestite dai clienti AWS specifiche per i tuoi casi d'uso. Per maggiori informazioni, consulta [Policy gestite da AWS](#) o [Policy gestite da AWS per le funzioni dei processi](#) nella Guida per l'utente di IAM.

- **Applicazione delle autorizzazioni con privilegio minimo** - Quando si impostano le autorizzazioni con le policy IAM, concedere solo le autorizzazioni richieste per eseguire un'attività. È possibile farlo definendo le azioni che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come autorizzazioni con privilegio minimo. Per maggiori informazioni sull'utilizzo di IAM per applicare le autorizzazioni, consulta [Policy e autorizzazioni in IAM](#) nella Guida per l'utente di IAM.
- **Condizioni d'uso nelle policy IAM per limitare ulteriormente l'accesso** - Per limitare l'accesso ad azioni e risorse è possibile aggiungere una condizione alle policy. Ad esempio, è possibile scrivere una condizione di policy per specificare che tutte le richieste devono essere inviate utilizzando SSL. Puoi anche utilizzare le condizioni per concedere l'accesso alle azioni del servizio se vengono utilizzate tramite uno specifico Servizio AWS, ad esempio CloudFormation. Per maggiori informazioni, consultare la sezione [Elementi delle policy JSON di IAM: condizione](#) nella Guida per l'utente di IAM.
- **Utilizzo dello strumento di analisi degli accessi IAM per convalidare le policy IAM e garantire autorizzazioni sicure e funzionali** - Lo strumento di analisi degli accessi IAM convalida le policy nuove ed esistenti in modo che aderiscano al linguaggio (JSON) della policy IAM e alle best practice di IAM. Lo strumento di analisi degli accessi IAM offre oltre 100 controlli delle policy e consigli utili per creare policy sicure e funzionali. Per maggiori informazioni, consultare [Convalida delle policy per il Sistema di analisi degli accessi IAM](#) nella Guida per l'utente di IAM.
- **Richiedi l'autenticazione a più fattori (MFA)**: se hai uno scenario che richiede utenti IAM o un utente root nel tuo Account AWS, attiva l'MFA per una maggiore sicurezza. Per richiedere la MFA quando vengono chiamate le operazioni API, aggiungere le condizioni MFA alle policy. Per maggiori informazioni, consultare [Protezione dell'accesso API con MFA](#) nella Guida per l'utente di IAM.

Per maggiori informazioni sulle best practice in IAM, consulta [Best practice di sicurezza in IAM](#) nella Guida per l'utente di IAM.

## Uso della console Lambda

Per accedere alla AWS Lambda console, devi disporre di un set minimo di autorizzazioni. Queste autorizzazioni devono consentirti di elencare e visualizzare i dettagli sulle risorse Lambda presenti nel tuo Account AWS. Se crei una policy basata sull'identità più restrittiva rispetto alle autorizzazioni

minime richieste, la console non funzionerà nel modo previsto per le entità (utenti o ruoli) associate a tale policy.

Non è necessario consentire autorizzazioni minime per la console per gli utenti che effettuano chiamate solo verso AWS CLI o l'API. AWS Al contrario, è opportuno concedere l'accesso solo alle azioni che corrispondono all'operazione API che stanno cercando di eseguire.

Per una policy di esempio che consente l'accesso minimo per lo sviluppo di funzioni, vedere [Concedere agli utenti l'accesso a una funzione Lambda](#). Oltre a Lambda APIs, la console Lambda utilizza altri servizi per visualizzare la configurazione dei trigger e consentire di aggiungere nuovi trigger. Se gli utenti utilizzano Lambda con altri servizi, devono anche accedere a tali servizi. Per informazioni dettagliate sulla configurazione di altri servizi con Lambda, consultare [Richiamare Lambda con eventi provenienti da altri AWS services](#).

Consentire agli utenti di visualizzare le loro autorizzazioni

Questo esempio mostra in che modo è possibile creare una policy che consente agli utenti IAM di visualizzare le policy inline e gestite che sono collegate alla relativa identità utente. Questa politica include le autorizzazioni per completare questa azione sulla console o utilizzando l'API o a livello di codice. AWS CLI AWS

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
```

```
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
    ],
    "Resource": "*"
}
]
```

## AWS politiche gestite per AWS Lambda

Una policy AWS gestita è una policy autonoma creata e amministrata da AWS. Le politiche gestite sono progettate per fornire autorizzazioni per molti casi d'uso comuni, in modo da poter iniziare ad assegnare autorizzazioni a utenti, gruppi e ruoli.

Tieni presente che le policy AWS gestite potrebbero non concedere le autorizzazioni con il privilegio minimo per i tuoi casi d'uso specifici, poiché sono disponibili per tutti i clienti. AWS si consiglia pertanto di ridurre ulteriormente le autorizzazioni definendo [policy gestite dal cliente](#) specifiche per i propri casi d'uso.

Non è possibile modificare le autorizzazioni definite nelle politiche gestite. Se AWS aggiorna le autorizzazioni definite in una politica AWS gestita, l'aggiornamento ha effetto su tutte le identità principali (utenti, gruppi e ruoli) a cui è associata la politica. AWS è più probabile che aggiorni una policy AWS gestita quando nel Servizio AWS viene lanciata una nuova o quando diventano disponibili nuove operazioni API per i servizi esistenti.

Per ulteriori informazioni, consultare [Policy gestite da AWS](#) nella Guida per l'utente di IAM.

### Argomenti

- [AWS policy gestita: AWS Lambda\\_FullAccess](#)
- [AWS policy gestita: AWS Lambda\\_ReadOnlyAccess](#)
- [AWS politica gestita: AWSLambdaBasicExecutionRole](#)
- [AWS politica gestita: AWSLambdaBasicDurableExecutionRolePolicy](#)
- [AWS politica gestita: AWSLambdaDynamoDBExecutionRole](#)

- [AWS politica gestita: AWSLambdaENIManagementAccess](#)
- [AWS politica gestita: AWSLambdaInvocation-DynamoDB](#)
- [AWS politica gestita: AWSLambdaKinesisExecutionRole](#)
- [AWS politica gestita: AWSLambdaMSKExecutionRole](#)
- [AWS politica gestita: AWSLambdaRole](#)
- [AWS politica gestita: AWSLambdaSQSQueueExecutionRole](#)
- [AWS politica gestita: AWSLambdaVPCLAccessExecutionRole](#)
- [AWS politica gestita: AWSLambdaManagedEC2ResourceOperator](#)
- [AWS politica gestita: AWSLambdaServiceRolePolicy](#)
- [AWS politica gestita: AWSLambdaNetworkConnectorOperatorPolicy](#)
- [Aggiornamenti Lambda a AWS policy gestite](#)

AWS policy gestita: AWS Lambda\_FullAccess

Questa policy concede l'accesso completo a tutte le operazioni di Lambda. Concede inoltre le autorizzazioni ad altri AWS servizi utilizzati per sviluppare e gestire le risorse Lambda.

Puoi associare la policy AWSLambda\_FullAccess ai tuoi utenti, gruppi e ruoli.

#### Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `lambda`: consente ai principali l'accesso completo a Lambda.
- `cloudformation`— Consente ai responsabili di descrivere gli AWS CloudFormation stack ed elencare le risorse in essi contenuti.
- `cloudwatch`— Consente ai responsabili di elencare i CloudWatch parametri di Amazon e ottenere dati sui parametri.
- `ec2`: consente ai principali di descrivere gruppi di sicurezza, sottoreti e VPC.
- `iam`: consente ai principali di ottenere policy, versioni delle policy, ruoli, policy dei ruoli, policy di ruolo allegate e l'elenco dei ruoli. Questa policy consente inoltre ai principali di trasferire ruoli a Lambda. L'autorizzazione `PassRole` è necessaria quando si assegna un ruolo di esecuzione a una funzione. L'`CreateServiceLinkedRole` autorizzazione viene utilizzata durante la creazione di un ruolo collegato al servizio.

- `kms`— Consente ai responsabili di elencare gli alias e descrivere la chiave per la crittografia dei volumi.
- `logs`— Consente ai responsabili di descrivere i flussi di log, ottenere eventi di registro, filtrare gli eventi di registro e avviare e interrompere le sessioni di Live Tail.
- `states`— Consente ai responsabili di descrivere ed AWS Step Functions elencare le macchine a stati.
- `tag`: consente ai principali di ottenere risorse in base ai loro tag.
- `xray`— Consente ai responsabili di ottenere riepiloghi delle AWS X-Ray tracce e recuperare un elenco di tracce specificato dall'ID.

Per ulteriori informazioni su questa policy, inclusi il documento sulla policy JSON e le versioni delle policy, consulta [AWS Lambda\\_FullAccess](#) nella AWS Managed Policy Reference Guide.

AWS policy gestita: `AWS Lambda_ReadOnlyAccess`

Questa policy garantisce l'accesso in sola lettura alle risorse Lambda e ad altri AWS servizi utilizzati per sviluppare e gestire le risorse Lambda.

Puoi associare la policy `AWSLambda_ReadOnlYAccess` ai tuoi utenti, gruppi e ruoli.

### Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `lambda`: consente ai principali di ottenere ed elencare tutte le risorse.
- `cloudformation`— Consente ai responsabili di descrivere ed elencare gli AWS CloudFormation stack e di elencare le risorse in essi contenuti.
- `cloudwatch`— Consente ai responsabili di elencare i CloudWatch parametri di Amazon e ottenere dati sui parametri.
- `ec2`: consente ai principali di descrivere gruppi di sicurezza, sottoreti e VPC.
- `iam`: consente ai principali di ottenere policy, versioni delle policy, ruoli, policy dei ruoli, policy di ruolo allegate e l'elenco dei ruoli.
- `kms`: consente ai principali di elencare gli alias.
- `logs`— Consente ai responsabili di descrivere i flussi di log, ottenere eventi di registro, filtrare gli eventi di registro e avviare e interrompere sessioni di Live Tail.

- `states`— Consente ai responsabili di descrivere ed AWS Step Functions elencare le macchine a stati.
- `tag`: consente ai principali di ottenere risorse in base ai loro tag.
- `xray`— Consente ai responsabili di ottenere riepiloghi delle AWS X-Ray tracce e recuperare un elenco di tracce specificato dall'ID.

Per ulteriori informazioni su questa policy, inclusi il documento sulla policy JSON e le versioni delle policy, consulta [AWS Lambda\\_ReadOnlyAccess](#) nella AWS Managed Policy Reference Guide.

AWS politica gestita: `AWSLambdaBasicExecutionRole`

Questa politica concede le autorizzazioni per caricare i log in Logs. CloudWatch

Puoi associare la policy `AWSLambdaBasicExecutionRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento sulla policy JSON e le versioni delle politiche, consulta la Managed Policy Reference [AWSLambdaBasicExecutionRole](#)Guide.AWS

AWS politica gestita: `AWSLambdaBasicDurableExecutionRolePolicy`

Questa policy fornisce autorizzazioni di scrittura per CloudWatch i log e read/write autorizzazioni per le API di esecuzione durevoli utilizzate dalle funzioni durevoli di Lambda. Questa policy fornisce le autorizzazioni essenziali richieste per le funzioni durevoli Lambda, che utilizzano API di esecuzione durevoli per mantenere l'avanzamento e mantenere lo stato tra le chiamate di funzione.

Puoi associare la policy `AWSLambdaBasicDurableExecutionRolePolicy` ai tuoi utenti, gruppi e ruoli.

Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `logs`— Consente ai responsabili di creare gruppi di log e flussi di log e di scrivere eventi di log in Logs. CloudWatch
- `lambda`— Consente ai responsabili di controllare lo stato di esecuzione durevole e recuperare lo stato di esecuzione durevole per le funzioni durevoli Lambda.

Per visualizzare ulteriori dettagli sulla policy, inclusa la versione più recente del documento di policy JSON, consulta [AWSLambdaBasicDurableExecutionRolePolicy](#) nella Guida di riferimento alle policy gestite da AWS .



### AWS politica gestita: AWSLambdaDynamoDBExecutionRole

Questa policy concede le autorizzazioni per leggere i record da uno stream Amazon DynamoDB e scrivere su Logs. CloudWatch

Puoi associare la policy `AWSLambdaDynamoDBExecutionRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento e le versioni della policy JSON, consulta la Managed Policy Reference Guide [AWSLambdaDynamoDBExecutionRole](#).AWS

### AWS politica gestita: AWSLambdaENIManagementAccess

Questa politica concede le autorizzazioni per creare, descrivere ed eliminare interfacce di rete elastiche utilizzate da una funzione VPC-enabled Lambda.

Puoi associare la policy `AWSLambdaENIManagementAccess` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento e le versioni delle policy JSON, consulta la Managed Policy Reference [AWSLambdaENIManagementAccess](#)Guide.AWS

### AWS politica gestita: AWSLambdaInvocation-DynamoDB

Questa policy concede l'accesso in lettura ai flussi Amazon DynamoDB.

Puoi associare la policy `AWSLambdaInvocation-DynamoDB` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento sulla policy JSON e le versioni delle policy, consulta [AWSLambdaInvocation-DynamoDB](#)la AWS Managed Policy Reference Guide.

### AWS politica gestita: AWSLambdaKinesisExecutionRole

Questa policy concede le autorizzazioni per leggere gli eventi da un flusso di dati di Amazon Kinesis e scriverli nei log. CloudWatch

Puoi associare la policy `AWSLambdaKinesisExecutionRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, incluso il documento e le versioni della policy JSON, consulta la Managed Policy Reference [AWSLambdaKinesisExecutionRole](#)Guide.AWS

### AWS politica gestita: AWSLambdaMSKExecutionRole

Questa policy concede le autorizzazioni per leggere e accedere ai record da un cluster Amazon Managed Streaming for Apache Kafka, gestire interfacce di rete elastiche e scrivere su Logs. CloudWatch

Puoi associare la policy `AWSLambdaMSKExecutionRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa policy, inclusi il documento sulla policy JSON e le versioni delle policy, consulta la Managed Policy Reference Guide. [AWSLambdaMSKExecutionRole](#)AWS

AWS politica gestita: `AWSLambdaRole`

Questa policy concede le autorizzazioni per invocare le funzioni Lambda.

Puoi associare la policy `AWSLambdaRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento sulla policy JSON e le versioni delle policy, consulta [AWSLambdaRole](#)la AWS Managed Policy Reference Guide.

AWS politica gestita: `AWSLambdaSQSQueueExecutionRole`

Questa politica concede le autorizzazioni per leggere ed eliminare i messaggi da una coda di Amazon Simple Queue Service e concede le autorizzazioni di scrittura ai log. CloudWatch

Puoi associare la policy `AWSLambdaSQSQueueExecutionRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento e le versioni della policy JSON, consulta la Managed Policy Reference Guide. [AWSLambdaSQSQueueExecutionRole](#)AWS

AWS politica gestita: `AWSLambdaVPCLocalAccessExecutionRole`

Questa politica concede le autorizzazioni per gestire interfacce di rete elastiche all'interno di un Amazon Virtual Private Cloud e scrivere su Logs. CloudWatch

Puoi associare la policy `AWSLambdaVPCLocalAccessExecutionRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento e le versioni della policy JSON, consulta la Managed Policy Reference [AWSLambdaVPCLocalAccessExecutionRole](#)Guide.AWS

AWS politica gestita: `AWSLambdaManagedEC2ResourceOperator`

Questa policy consente la gestione automatizzata delle istanze di Amazon Elastic Compute Cloud e la CloudWatch registrazione di Amazon per i provider di capacità Lambda. Concede a Lambda le autorizzazioni per eseguire operazioni sul ciclo di vita delle istanze e scrivere log per tuo conto.

Puoi associare la policy `AWSLambdaManagedEC2ResourceOperator` ai tuoi utenti, gruppi e ruoli.

## Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `ec2:RunInstances`— Consente a Lambda di lanciare nuove istanze Amazon EC2 a condizione che `ec2:` sia `ManagedResourceOperator` uguale a `scaler.lambda.amazonaws.com` e limiti l'utilizzo dell'AMI alle sole immagini. Amazon-owned
- `ec2:DescribeInstances` e `ec2:DescribeInstanceStatus` — Consente a Lambda di monitorare lo stato dell'istanza e recuperare le informazioni sull'istanza.
- `ec2:CreateTags`— Consente a Lambda di etichettare le risorse Amazon EC2 per scopi di gestione e identificazione.
- `ec2:DescribeAvailabilityZones`— Consente a Lambda di visualizzare le zone disponibili, ad esempio le decisioni di posizionamento.
- `ec2:DescribeCapacityReservations`— Consente a Lambda di verificare le prenotazioni di capacità per un posizionamento ottimale delle istanze.
- `ec2:DescribeInstanceTypes` e `ec2:DescribeInstanceTypeOfferings`: consente a Lambda di esaminare i tipi di istanze disponibili e le relative offerte.
- `ec2:DescribeSubnets`— Consente a Lambda di esaminare le configurazioni delle sottoreti per la pianificazione della rete.
- `ec2:DescribeVpcEncryptionControls`— Consente a Lambda di esaminare le impostazioni di controllo della crittografia VPC, ad esempio la configurazione.
- `ec2:DescribeSecurityGroups`— Consente a Lambda di recuperare le informazioni sul gruppo di sicurezza per la configurazione dell'interfaccia di rete.
- `ec2:CreateNetworkInterface`— Consente a Lambda di creare interfacce di rete e gestire associazioni di sottoreti e gruppi di sicurezza.
- `ec2:AttachNetworkInterface`— [Consente a Lambda di collegare interfacce di rete alle istanze Amazon EC2 a condizione che sia uguale a `scaler.lambda.amazonaws.com`. `ec2:ManagedResourceOperator`](#)
- `logs:CreateLogGroup`— Consente a Lambda di creare gruppi di CloudWatch log Logs.
- `logs:CreateLogStream` e `logs:PutLogEvents` — Consente a Lambda di creare flussi di log e scrivere eventi di log in Logs. CloudWatch

Per ulteriori informazioni su questa politica, inclusi il documento e le versioni delle policy JSON, consulta la AWS Managed Policy [AWSLambdaManagedEC2ResourceOperator](#) Reference Guide.

## AWS politica gestita: AWSLambdaServiceRolePolicy

Questa policy è associata al ruolo collegato al servizio denominato per consentire AWSServiceRoleForLambda a Lambda di 1) terminare le istanze gestite come parte dei provider di capacità Lambda e 2) eliminare le interfacce di rete elastiche (ENI) gestite come parte dei connettori di rete Lambda.

### Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `ec2:TerminateInstances`— Consente a Lambda di terminare le istanze EC2 a condizione che `ec2:` sia uguale a `scaler.lambda.amazonaws.com`. `ManagedResourceOperator`
- `ec2:DescribeInstanceStatus` `ec2:DescribeInstances` — Consente a Lambda di descrivere le istanze EC2.
- `ec2:DescribeNetworkInterfaces` `ec2:DescribeSubnets`, e `ec2:DescribeSecurityGroups` — Consente a Lambda di descrivere interfacce di rete, sottoreti e gruppi di sicurezza.
- `ec2:DeleteNetworkInterface`— Consente a Lambda di eliminare ENI a condizione che `ec2:` equals `network-connectors.lambda.amazonaws.com`. `ManagedResourceOperator`

Per ulteriori informazioni su questa politica, consulta [Using service-linked roles for Lambda](#).

## AWS politica gestita: AWSLambdaNetworkConnectorOperatorPolicy

Questa politica concede le autorizzazioni per creare e amministrare risorse ENI (elastic network interface) gestite dal Lambda Network Connector.

Puoi associare la policy `AWSLambdaNetworkConnectorOperatorPolicy` ai tuoi utenti, gruppi e ruoli.

### Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `ec2:CreateNetworkInterface`— Concede l'autorizzazione a creare interfacce di rete in qualsiasi sottorete con gruppi di sicurezza associati. Per le risorse dell'interfaccia di rete, sono consentiti solo i tag con `aws:lambda:networkConnectorName` e `aws:lambda:networkConnectorId` chiavi.

- `ec2:CreateTags`— Concede l'autorizzazione a contrassegnare le interfacce di rete al momento della creazione. Questa autorizzazione si applica solo quando l'azione di creazione è `CreateNetworkInterface` e l'operatore della risorsa gestita lo è. `network-connectors.lambda.amazonaws.com`

Per ulteriori informazioni su questa politica, inclusi il documento e le versioni della policy JSON, consulta [AWSLambdaNetworkConnectorOperatorPolicy](#) la AWS Managed Policy Reference Guide.

#### Aggiornamenti Lambda a AWS policy gestite

| Modifica                                                               | Descrizione                                                                                                                                                                                                                                                                                      | Data           |
|------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| <a href="#">AWSLambdaNetworkConnectorOperatorPolicy</a> : nuova policy | Lambda ha aggiunto una nuova politica gestita per concedere le autorizzazioni per creare e amministrare le risorse ENI gestite dal Lambda Network Connector.                                                                                                                                     | 19 giugno 2026 |
| <a href="#">AWSLambdaManagedEC2ResourceOperator</a> — Cambia           | Lambda ha aggiornato la <code>AWSLambdaManagedEC2ResourceOperator</code> policy per consentire le azioni <code>logs:CreateLogGroup</code> , <code>logs:CreateLogStream</code> , e <code>logs:PutLogEvents</code> . Queste autorizzazioni consentono a Lambda di scrivere log in CloudWatch Logs. | 10 giugno 2026 |
| <a href="#">AWSLambdaManagedEC2ResourceOperator</a> — Cambia           | Lambda ha aggiornato la <code>AWSLambdaManagedEC2ResourceOperator</code> policy per consentire l'azione <code>ec2:DescribeVpcEncryptionCo</code>                                                                                                                                                 | 23 aprile 2026 |

| Modifica                                                           | Descrizione                                                                                                                                                                                                                            | Data             |
|--------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
|                                                                    | ntrols . Questa autorizzazione consente a Lambda di esaminare le impostazioni di controllo della crittografia VPC, ad esempio la configurazione.                                                                                       |                  |
| <a href="#">AWSLambdaManagedEC2ResourceOperator</a> : nuova policy | Lambda ha aggiunto una nuova policy gestita per abilitare la gestione automatizzata delle istanze Amazon EC2 per i provider di capacità Lambda, consentendo al servizio scaler di eseguire operazioni sul ciclo di vita delle istanze. | 30 novembre 2025 |
| <a href="#">AWSLambdaServiceRolePolicy</a> : nuova policy          | Lambda ha aggiunto una nuova policy gestita per il ruolo collegato ai servizi per consentire a Lambda di terminare le istanze gestite come parte dei provider di capacità Lambda.                                                      | 30 novembre 2025 |
| <a href="#">AWS_Lambda_FullAccess</a> — Modifica                   | Lambda ha aggiornato la <code>AWSLambda_FullAccess</code> policy per consentire le azioni <code>kms:DescribeKey</code> e <code>iam:CreateServiceLinkedRole</code> .                                                                    | 30 novembre 2025 |

| Modifica                                                                                        | Descrizione                                                                                                                                                                                                                                                                     | Data            |
|-------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">AWSLambdaBasicDurableExecutionRolePolicy</a> — Nuova politica gestita               | Lambda ha rilasciato una nuova policy gestita <code>AWSLambdaBasicDurableExecutionRolePolicy</code> che fornisce autorizzazioni di scrittura per CloudWatch i log e read/write e autorizzazioni per le API di esecuzione durevoli utilizzate dalle funzioni durevoli di Lambda. | 1 dicembre 2025 |
| <a href="#">AWS Lambda_ReadOnlyAccess</a> e <a href="#">AWS Lambda_FullAccess</a> — Cambiamento | Lambda ha aggiornato le policy <code>AWSLambda_ReadOnlyAccess</code> e <code>AWSLambda_FullAccess</code> per consentire le azioni <code>logs:StartLiveTail</code> e <code>logs:StopLiveTail</code> .                                                                            | 17 marzo 2025   |
| <a href="#">AWSLambdaVPCAccessExecutionRole</a> — Cambia                                        | Lambda ha aggiornato la policy <code>AWSLambdaVPCAccessExecutionRole</code> per consentire l'azione <code>ec2:DescribeSubnets</code> .                                                                                                                                          | 5 gennaio 2024  |
| <a href="#">AWS Lambda_ReadOnlyAccess</a> — Modifica                                            | Lambda ha aggiornato la <code>AWSLambda_ReadOnlyAccess</code> policy per consentire ai principali di elencare gli stack. CloudFormation                                                                                                                                         | 27 luglio 2023  |

| Modifica                                                | Descrizione                                                                               | Data           |
|---------------------------------------------------------|-------------------------------------------------------------------------------------------|----------------|
| AWS Lambda ha iniziato a tenere traccia delle modifiche | AWS Lambda ha iniziato a tenere traccia delle modifiche per le sue politiche AWS gestite. | 27 luglio 2023 |

## Risoluzione dei problemi AWS Lambda di identità e accesso

Utilizza le seguenti informazioni per diagnosticare e risolvere i problemi comuni che possono verificarsi durante l'uso di Lambda e IAM.

### Argomenti

- [Non sono autorizzato/a a eseguire un'operazione in Lambda](#)
- [Non sono autorizzato a eseguire iam: PassRole](#)
- [Voglio consentire a persone esterne a me di accedere Account AWS alle mie risorse Lambda](#)

### Non sono autorizzato/a a eseguire un'operazione in Lambda

Se ricevi un errore che indica che non sei autorizzato a eseguire un'operazione, le tue policy devono essere aggiornate per poter eseguire l'operazione.

L'errore di esempio seguente si verifica quando l'utente IAM `mateojackson` prova a utilizzare la console per visualizzare i dettagli relativi a una risorsa `my-example-widget` fittizia ma non dispone di autorizzazioni `lambda:GetWidget` fittizie.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
lambda:GetWidget on resource: my-example-widget
```

In questo caso, la policy per l'utente `mateojackson` deve essere aggiornata per consentire l'accesso alla risorsa `my-example-widget` utilizzando l'azione `lambda:GetWidget`.

Se hai bisogno di assistenza, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.



## Non sono autorizzato a eseguire iam: PassRole

Se ricevi un errore che indica che non sei autorizzato a eseguire l'operazione `iam:PassRole`, devi aggiornare le policy per poter passare un ruolo a Lambda.

Alcuni Servizi AWS consentono di passare un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per passare il ruolo al servizio.

L'errore di esempio riportato di seguito si verifica quando un utente IAM denominato `marymajor` tenta di utilizzare la console per eseguire un'operazione in Lambda. Tuttavia, l'azione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per trasmettere il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Voglio consentire a persone esterne a me di accedere Account AWS alle mie risorse Lambda

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo. Per i servizi che supportano politiche basate sulle risorse o liste di controllo degli accessi (ACLs), puoi utilizzare tali politiche per concedere alle persone l'accesso alle tue risorse.

Per ulteriori informazioni, consultare gli argomenti seguenti:

- Per sapere se Lambda supporta queste funzionalità, consultare [Come AWS Lambda funziona con IAM](#).
- Per scoprire come fornire l'accesso alle tue risorse attraverso Account AWS le risorse di tua proprietà, consulta [Fornire l'accesso a un utente IAM in un altro Account AWS di tua proprietà](#) nella IAM User Guide.
- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta [Fornire l'accesso a soggetti Account AWS di proprietà di terze parti](#) nella Guida per l'utente IAM.

- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta [Fornire l'accesso a utenti autenticati esternamente \(federazione delle identità\)](#) nella Guida per l'utente IAM.
- Per informazioni sulle differenze di utilizzo tra ruoli e policy basate su risorse per l'accesso multi-account, consulta [Accesso a risorse multi-account in IAM](#) nella Guida per l'utente IAM.

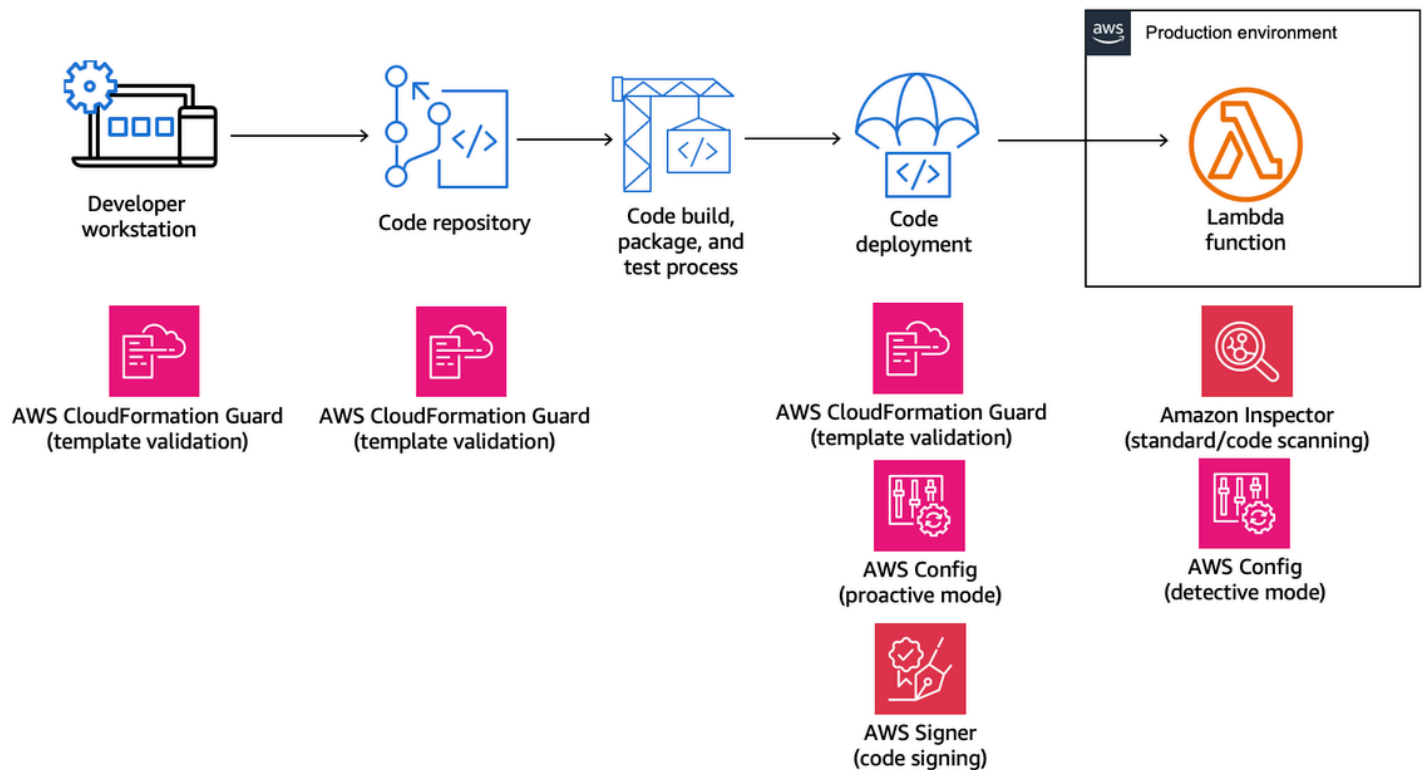
## Creare una strategia di governance per le funzioni e i livelli Lambda

Per creare e implementare applicazioni serverless native del cloud, è necessario garantire agilità e velocità di immissione sul mercato con una governance e guardrail adeguati. Stabilisci delle priorità a livello aziendale, magari sottolineando l'agilità come priorità assoluta o, in alternativa, enfatizzando l'avversione al rischio attraverso governance, guardrail e controlli. Realisticamente, non avrai una strategia either/or "» ma una strategia «e» che bilanci agilità e barriere nel ciclo di vita dello sviluppo del software. A prescindere dal punto del ciclo di vita dell'azienda in cui tali requisiti rientrano, è probabile che le funzionalità di governance diventino un requisito di implementazione nei processi e nelle toolchain.

Di seguito sono riportati alcuni esempi di controlli di governance che un'organizzazione potrebbe implementare per Lambda:

- Le funzioni Lambda non devono essere accessibili pubblicamente.
- Le funzioni Lambda devono essere associate a un VPC.
- Le funzioni Lambda non dovrebbero utilizzare runtime ritirati.
- Le funzioni Lambda devono essere etichettate con un set di tag obbligatori.
- I livelli Lambda non devono essere accessibili all'esterno dell'organizzazione.
- Le funzioni Lambda con un gruppo di sicurezza associato devono disporre di corrispondenti tra la funzione e il gruppo di sicurezza.
- Le funzioni Lambda con un livello associato devono utilizzare una versione approvata.
- Le variabili di ambiente Lambda devono essere crittografate a riposo con una chiave gestita dal cliente.

Il diagramma seguente è un esempio di una strategia di governance approfondita che implementa controlli e policy durante tutto il processo di sviluppo e implementazione del software:



I seguenti argomenti spiegano come implementare i controlli per lo sviluppo e l'implementazione di funzioni Lambda nell'organizzazione, sia per le startup che per le imprese. L'organizzazione potrebbe già disporre di alcuni strumenti. Gli argomenti seguenti adottano un approccio modulare a tali controlli, affinché si possano scegliere i componenti effettivamente necessari. [the section called “Controlli proattivi con Guard”](#) [the section called “Scansione del codice”](#) [the section called “Firma del codice”](#) Gli [the section called “Osservabilità”](#) argomenti includono,, e.

## Argomenti

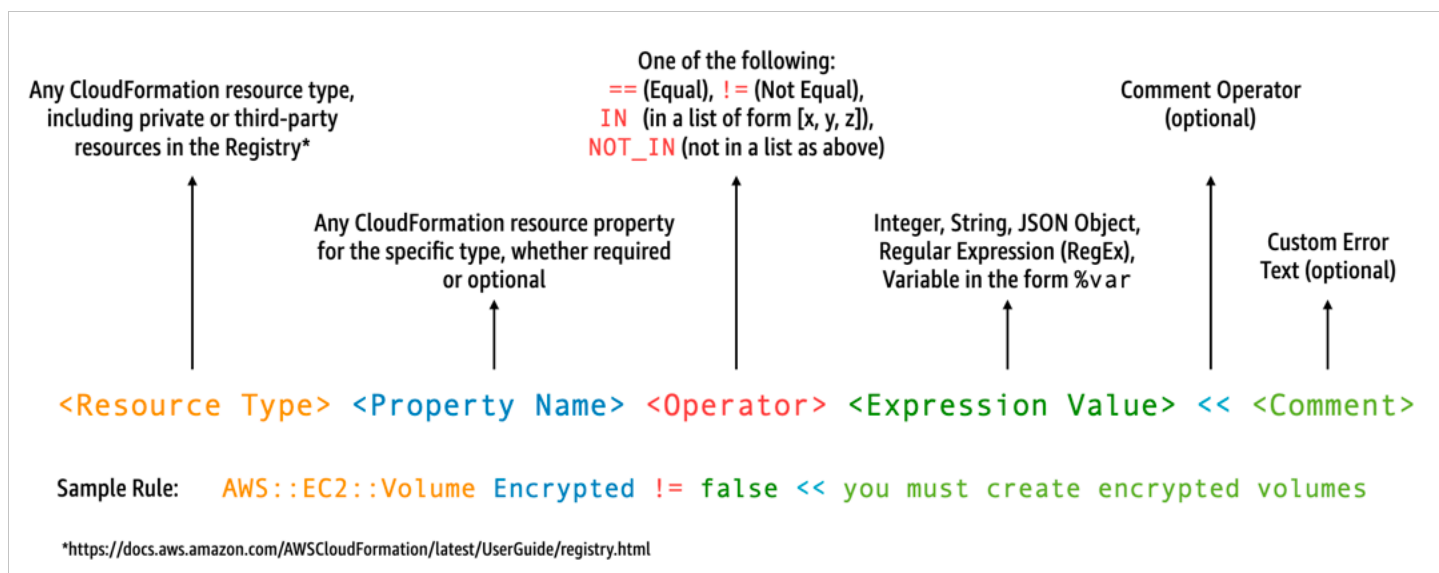
- [Controlli proattivi per Lambda con AWS CloudFormation Guard](#)
- [Implementa controlli preventivi per Lambda con AWS Config](#)
- [Rileva implementazioni e configurazioni Lambda non conformi con AWS Config](#)
- [Firma del codice Lambda con AWS Signer](#)
- [Automatizzare le valutazioni di sicurezza per Lambda con Amazon Inspector](#)
- [Implementazione dell'osservabilità per la sicurezza e la conformità Lambda](#)

## Controlli proattivi per Lambda con AWS CloudFormation Guard

[AWS CloudFormation Guard](#) è uno strumento di valutazione policy as code open source per uso generico. Può essere utilizzato per la governance preventiva e la conformità attraverso la convalida dei modelli di infrastructure as code (IaC) e le composizioni dei servizi rispetto alle regole delle policy. Queste regole possono essere personalizzate in base ai requisiti del team o dell'organizzazione. Per le funzioni Lambda, è possibile utilizzare le regole Guard per controllare la creazione di risorse e gli aggiornamenti della configurazione definendo le impostazioni di proprietà richieste necessarie durante la creazione o l'aggiornamento di una funzione Lambda.

Gli amministratori addetti alla conformità definiscono l'elenco dei controlli e delle policy di governance necessari per l'implementazione e l'aggiornamento delle funzioni Lambda. Gli amministratori della piattaforma implementano i controlli in CI/CD pipeline, come webhook di convalida pre-commit con repository di codice, e forniscono agli sviluppatori strumenti a riga di comando per la convalida di modelli e codice sulle workstation locali. Gli sviluppatori creano il codice, convalidano i modelli con strumenti a riga di comando e quindi eseguono il commit del codice nei repository, che vengono quindi convalidati automaticamente tramite le pipeline prima della distribuzione in un ambiente. CI/CD AWS

Con Guard, puoi [scrivere le tue regole](#) e implementare i controlli con un linguaggio specifico del dominio come segue.



Ad esempio, supponi di volerti assicurare che gli sviluppatori scelgano solo i runtime più recenti. Potresti specificare due policy diverse, una per identificare i [runtime](#) già ritirati e l'altra per identificare

i runtime che verranno ritirati a breve. A tale scopo, potresti scrivere il file `etc/rules.guard` seguente:

```
let lambda_functions = Resources.*[
  Type == "AWS::Lambda::Function"
]

rule lambda_already_deprecated_runtime when %lambda_functions !empty {
  %lambda_functions {
    Properties {
      when Runtime exists {
        Runtime !in ["dotnetcore3.1", "nodejs12.x", "python3.6", "python2.7",
"dotnet5.0", "dotnetcore2.1", "ruby2.5", "nodejs10.x", "nodejs8.10", "nodejs4.3",
"nodejs6.10", "dotnetcore1.0", "dotnetcore2.0", "nodejs4.3-edge", "nodejs"] <<Lambda
function is using a deprecated runtime.>>
      }
    }
  }
}

rule lambda_soon_to_be_deprecated_runtime when %lambda_functions !empty {
  %lambda_functions {
    Properties {
      when Runtime exists {
        Runtime !in ["nodejs16.x", "nodejs14.x", "python3.7", "java8",
"dotnet7", "go1.x", "ruby2.7", "provided"] <<Lambda function is using a runtime that
is targeted for deprecation.>>
      }
    }
  }
}
```

Supponiamo ora di scrivere il seguente `iac/lambda.yaml` CloudFormation modello che definisce una funzione Lambda:

```
Fn:
  Type: AWS::Lambda::Function
  Properties:
    Runtime: python3.7
    CodeUri: src
    Handler: fn.handler
    Role: !GetAtt FnRole.Arn
    Layers:
```

```
- arn:aws:lambda:us-east-1:111122223333:layer:LambdaInsightsExtension:35
```

Dopo aver eseguito l'[installazione](#) della funzionalità Guard, convalida il modello:

```
cfn-guard validate --rules etc/rules.guard --data iac/lambda.yaml
```

L'output sarà il seguente:

```
lambda.yaml Status = FAIL
FAILED rules
rules.guard/lambda_soon_to_be_deprecated_runtime
---
Evaluating data lambda.yaml against rules rules.guard
Number of non-compliant resources 1
Resource = Fn {
  Type      = AWS::Lambda::Function
  Rule = lambda_soon_to_be_deprecated_runtime {
    ALL {
      Check = Runtime not IN
["nodejs16.x","nodejs14.x","python3.7","java8","dotnet7","go1.x","ruby2.7","provided"]
{
  ComparisonError {
    Message      = Lambda function is using a runtime that is targeted for
deprecation.
    Error         = Check was not compliant as property [/Resources/
Fn/Properties/Runtime[L:88,C:15]] was not present in [(resolved, Path=[L:0,C:0]
Value=["nodejs16.x","nodejs14.x","python3.7","java8","dotnet7","go1.x","ruby2.7","provided"])]
  }
  PropertyPath   = /Resources/Fn/Properties/Runtime[L:88,C:15]
  Operator       = NOT IN
  Value         = "python3.7"
  ComparedWith  =
[["nodejs16.x","nodejs14.x","python3.7","java8","dotnet7","go1.x","ruby2.7","provided"]]
  Code:
    86. Fn:
    87.   Type: AWS::Lambda::Function
    88.   Properties:
    89.     Runtime: python3.7
    90.     CodeUri: src
    91.     Handler: fn.handler
  }
}
```

```
}  
}
```

Con Guard, i tuoi sviluppatori possono vedere dalle loro workstation di sviluppo locali che devono aggiornare il modello per utilizzare un runtime consentito dall'organizzazione. Ciò avviene prima di eseguire il commit in un repository di codice e successivamente di non superare i controlli all'interno di una pipeline. CI/CD Di conseguenza, gli sviluppatori ricevono questo feedback su come sviluppare modelli conformi e dedicare più tempo alla scrittura di codice che offra valore aziendale. Questo controllo può essere applicato sulla workstation locale degli sviluppatori, in un webhook di convalida prima del commit, nella pipeline prima della distribuzione. and/or CI/CD

## Avvertenze

Se stai usando modelli AWS Serverless Application Model (AWS SAM) per definire le funzioni Lambda, tieni presente che devi aggiornare la regola Guard per cercare il tipo di risorsa come segue. `AWS::Serverless::Function`

```
let lambda_functions = Resources.*[  
    Type == "AWS::Serverless::Function"  
]
```

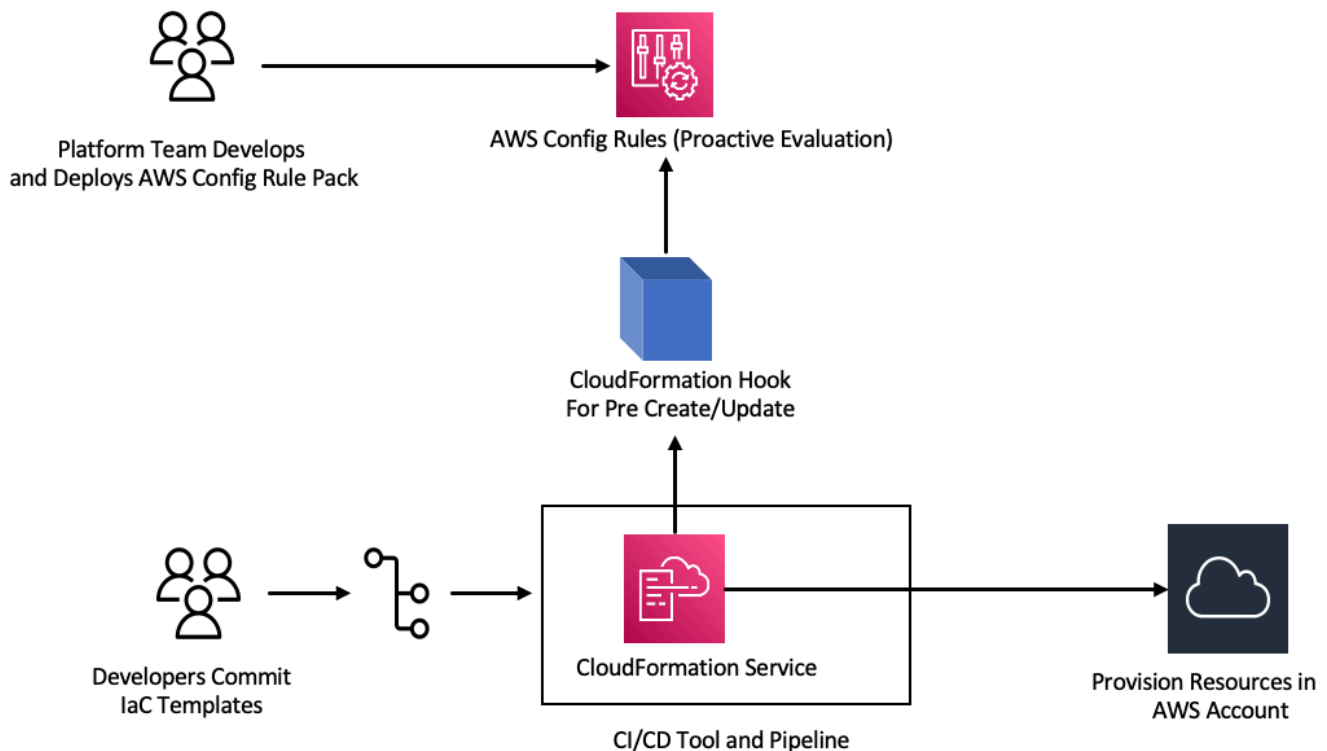
Guard si aspetta inoltre che le proprietà vengano incluse nella definizione della risorsa. Nel frattempo, AWS SAM i modelli consentono di specificare le proprietà in una sezione [Globals](#) separata. Le proprietà definite nella sezione Globali non vengono convalidate con le regole Guard.

Come indicato nella [documentazione](#) per la risoluzione dei problemi di Guard, tieni presente che Guard non supporta funzioni intrinseche in formato breve come `!GetAtt` o `!Sub` e richiede invece l'utilizzo dei formati espansi: `Fn::GetAtt` e `Fn::Sub`. (L'[esempio precedente](#) non valuta la proprietà Ruolo, quindi per semplicità è stata utilizzata la funzione intrinseca in formato breve.)

## Implementa controlli preventivi per Lambda con AWS Config

È essenziale garantire la conformità delle applicazioni serverless il più presto possibile nel processo di sviluppo. In questo argomento, spieghiamo come implementare controlli preventivi utilizzando [AWS Config](#). Con AWS Config, puoi implementare i controlli di conformità nelle prime fasi del processo di sviluppo e implementare gli stessi controlli nelle tue pipeline. CI/CD Ciò consente inoltre di standardizzare i controlli in un archivio di regole gestito centralmente in modo da poterli applicare in modo coerente su tutti gli account. AWS

Ad esempio, supponiamo che gli amministratori della conformità abbiano definito un requisito per garantire che tutte le funzioni Lambda includano il tracciamento. AWS X-Ray Con AWS Config la modalità proattiva, puoi eseguire controlli di conformità sulle risorse delle tue funzioni Lambda prima della distribuzione, riducendo il rischio di implementare funzioni Lambda configurate in modo errato e facendo risparmiare tempo agli sviluppatori fornendo loro un feedback più rapido sull'infrastruttura sotto forma di modelli di codice. Di seguito è riportata una visualizzazione del flusso per i controlli preventivi con: AWS Config



Immagina che vi sia un requisito per cui il tracciamento deve essere abilitato in tutte le funzioni Lambda. In risposta, il team della piattaforma identifica la necessità che una AWS Config regola



specifica venga eseguita in modo proattivo su tutti gli account. Questa regola contrassegna qualsiasi funzione Lambda priva di una configurazione di X-Ray tracciamento configurata come risorsa non conforme. Il team sviluppa una regola, la impacchetta in un pacchetto [di](#) conformità e lo distribuisce in tutti gli account per garantire che tutti gli AWS account dell'organizzazione applichino questi controlli in modo uniforme. Puoi scrivere la regola nella sintassi 2.x.x di AWS CloudFormation Guard , che assume la forma seguente:

```
rule name when condition { assertion }
```

Di seguito è riportato un esempio di regola Guard che verifica l'abilitazione del tracciamento nelle funzioni Lambda:

```
rule lambda_tracing_check {  
  when configuration.tracingConfig exists {  
    configuration.tracingConfig.mode == "Active"  
  }  
}
```

Il team della piattaforma intraprende ulteriori azioni imponendo che ogni AWS CloudFormation distribuzione richieda un pre-hook. create/update <https://docs.aws.amazon.com/cloudformation-cli/latest/userguide/hooks-structure.html> Il team si assume la piena responsabilità dello sviluppo di tale hook e di configurare la pipeline, rafforzando il controllo centralizzato delle regole di conformità e sostenendo l'applicazione coerente in tutte le implementazioni. Per sviluppare, impacchettare e registrare un hook, vedi Developing AWS CloudFormation Hooks [nella](#) documentazione di CloudFormation Command Line Interface (). CFN-CLI Puoi usare la [CloudFormation CLI](#) per creare il progetto hook:

```
cfn init
```

Questo comando richiede alcune informazioni di base sul progetto hook e crea un progetto contenente i file seguenti:

```
README.md  
<hook-name>.json  
rpd.log  
src/handler.py  
template.yml  
hook-role.yaml
```

Come sviluppatore di hook, devi aggiungere il tipo di risorsa di destinazione desiderato nel file di configurazione `<hook-name>.json`. Nella configurazione seguente, un hook è configurato per l'esecuzione prima che venga creata qualsiasi funzione Lambda utilizzando CloudFormation. Puoi anche aggiungere gestori simili per le azioni `preUpdate` e `preDelete`.

```
"handlers": {
  "preCreate": {
    "targetNames": [
      "AWS::Lambda::Function"
    ],
    "permissions": []
  }
}
```

È inoltre necessario assicurarsi che l'CloudFormation hook disponga delle autorizzazioni appropriate per chiamare le AWS Config API. Puoi farlo aggiornando il file di definizione del ruolo denominato `hook-role.yaml`. Per impostazione predefinita, il file di definizione del ruolo ha la seguente politica di fiducia, che consente di CloudFormation assumere il ruolo.

```
AssumeRolePolicyDocument:
  Version: '2012-10-17'
  Statement:
    - Effect: Allow
      Principal:
        Service:
          - hooks.cloudformation.amazonaws.com
          - resources.cloudformation.amazonaws.com
```

Per consentire all'hook di effettuare la chiamata alle API di configurazione, devi aggiungere le seguenti autorizzazioni all'istruzione Policy. Quindi invii il progetto hook utilizzando il `cfn submit` comando, dove CloudFormation crea un ruolo per te con le autorizzazioni richieste.

```
Policies:
  - PolicyName: HookTypePolicy
    PolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Action:
            - "config:Describe*"
            - "config:Get"
```

```

- "config:List*"
- "config:SelectResourceConfig"
Resource: "*"

```

Successivamente, devi scrivere una funzione Lambda in un file `src/handler.py`. All'interno di tale file, troverai i metodi denominati `preCreate`, `preUpdate` e `preDelete` già creati all'avvio del progetto. Il tuo obiettivo è scrivere una funzione comune e riutilizzabile che richiami l' `AWS Config StartResourceEvaluationAPI` in modalità proattiva utilizzando il `AWS SDK per Python (Boto3)`. Questa chiamata API accetta le proprietà della risorsa come input e valuta la risorsa rispetto alla definizione della regola.

```

def validate_lambda_tracing_config(resource_type, function_properties:
MutableMapping[str, Any]) -> ProgressEvent:
    LOG.info("Fetching proactive data")
    config_client = boto3.client('config')
    resource_specs = {
        'ResourceId': 'MyFunction',
        'ResourceType': resource_type,
        'ResourceConfiguration': json.dumps(function_properties),
        'ResourceConfigurationSchemaType': 'CFN_RESOURCE_SCHEMA'
    }
    LOG.info("Resource Specifications:", resource_specs)
    eval_response = config_client.start_resource_evaluation(EvaluationMode='PROACTIVE',
ResourceDetails=resource_specs, EvaluationTimeout=60)
    ResourceEvaluationId = eval_response.ResourceEvaluationId
    compliance_response =
config_client.get_compliance_details_by_resource(ResourceEvaluationId=ResourceEvaluationId)
    LOG.info("Compliance Verification:",
compliance_response.EvaluationResults[0].ComplianceType)
    if "NON_COMPLIANT" == compliance_response.EvaluationResults[0].ComplianceType:
        return ProgressEvent(status=OperationStatus.FAILED, message="Lambda function
found with no tracing enabled : FAILED", errorCode=HandlerErrorCode.NonCompliant)
    else:
        return ProgressEvent(status=OperationStatus.SUCCESS, message="Lambda function
found with tracing enabled : PASS.")

```

Ora puoi effettuare la chiamata alla funzione comune dal gestore per l'hook di pre-creazione. Di seguito è riportato un esempio del gestore:

```

@hook.handler(HookInvocationPoint.CREATE_PRE_PROVISION)
def pre_create_handler(
    session: Optional[SessionProxy],

```

```

        request: HookHandlerRequest,
        callback_context: MutableMapping[str, Any],
        type_configuration: TypeConfigurationModel
    ) -> ProgressEvent:
        LOG.info("Starting execution of the hook")
        target_name = request.hookContext.targetName
        LOG.info("Target Name:", target_name)
        if "AWS::Lambda::Function" == target_name:
            return validate_lambda_tracing_config(target_name,
                                                  request.hookContext.targetModel.get("resourceProperties"))
        )
    else:
        raise exceptions.InvalidRequest(f"Unknown target type: {target_name}")

```

Dopo questo passaggio è possibile registrare l'hook e configurarlo per ascoltare tutti gli eventi di creazione AWS Lambda della funzione.

Uno sviluppatore prepara il modello di infrastructure as code (IaC) per un microservizio serverless utilizzando Lambda. La preparazione prevede il rispetto degli standard interni, a cui seguono il test e il commit locale del modello nel repository. Ecco un esempio di modello IaC:

```

MyLambdaFunction:
  Type: 'AWS::Lambda::Function'
  Properties:
    Handler: index.handler
    Role: !GetAtt LambdaExecutionRole.Arn
    FunctionName: MyLambdaFunction
    Code:
      ZipFile: |
        import json

        def handler(event, context):
            return {
                'statusCode': 200,
                'body': json.dumps('Hello World!')
            }
    Runtime: python3.14
    TracingConfig:
      Mode: PassThrough
    MemorySize: 256
    Timeout: 10

```

Come parte del CI/CD processo, quando il CloudFormation modello viene distribuito, il CloudFormation servizio richiama il create/update pre-hook subito prima di assegnare il tipo di risorsa. `AWS::Lambda::Function` L'hook utilizza AWS Config regole eseguite in modalità proattiva per verificare che la configurazione della funzione Lambda includa la configurazione di tracciamento obbligatoria. La risposta dell'hook determina la fase successiva.

Se conforme, l'hook segnala l'esito positivo e CloudFormation procede al provisioning delle risorse. In caso contrario, l'implementazione CloudFormation dello stack fallisce, la pipeline si interrompe immediatamente e il sistema registra i dettagli per la successiva revisione. Le notifiche di conformità vengono inviate ai soggetti interessati.

Puoi trovare le success/fail informazioni sugli hook nella console: CloudFormation

| Stack info                                 | Events              | Resources            | Outputs                                                                                     | Parameters                    | Template | Change sets |
|--------------------------------------------|---------------------|----------------------|---------------------------------------------------------------------------------------------|-------------------------------|----------|-------------|
| <b>Events (19)</b>                         |                     |                      |                                                                                             |                               |          |             |
| <input type="text" value="Search events"/> |                     |                      |                                                                                             |                               |          |             |
| Timestamp                                  | Logical ID          | Status               | Status reason                                                                               | Hook invocations              |          |             |
| 2023-08-29 23:50:23 UTC-0500               | HookTestStack       | ROLLBACK_COMPLETE    | -                                                                                           | -                             |          |             |
| 2023-08-29 23:50:22 UTC-0500               | LambdaExecutionRole | DELETE_COMPLETE      | -                                                                                           | -                             |          |             |
| 2023-08-29 23:50:21 UTC-0500               | MyApi               | DELETE_COMPLETE      | -                                                                                           | -                             |          |             |
| 2023-08-29 23:50:20 UTC-0500               | LambdaExecutionRole | DELETE_IN_PROGRESS   | -                                                                                           | -                             |          |             |
| 2023-08-29 23:50:20 UTC-0500               | MyLambdaFunction    | DELETE_COMPLETE      | -                                                                                           | -                             |          |             |
| 2023-08-29 23:50:20 UTC-0500               | MyApi               | DELETE_IN_PROGRESS   | -                                                                                           | -                             |          |             |
| 2023-08-29 23:50:18 UTC-0500               | HookTestStack       | ROLLBACK_IN_PROGRESS | The following resource(s) failed to create: [MyLambdaFunction]. Rollback requested by user. | -                             |          |             |
| 2023-08-29 23:50:17 UTC-0500               | MyLambdaFunction    | CREATE_FAILED        | The following hook(s) failed: [AWS::LambdaTracingCheck::Hook]                               | -                             |          |             |
| 2023-08-29 23:50:17 UTC-0500               | MyLambdaFunction    | CREATE_IN_PROGRESS   | -                                                                                           | AWS::LambdaTracingCheck::Hook |          |             |
| 2023-08-29 23:50:16 UTC-0500               | MyLambdaFunction    | CREATE_IN_PROGRESS   | -                                                                                           | AWS::LambdaTracingCheck::Hook |          |             |
| 2023-08-29 23:50:15 UTC-0500               | MyLambdaFunction    | CREATE_IN_PROGRESS   | -                                                                                           | -                             |          |             |
| 2023-08-29 23:50:14 UTC-0500               | LambdaExecutionRole | CREATE_COMPLETE      | -                                                                                           | -                             |          |             |
| 2023-08-29 23:49:59 UTC-0500               | MyApi               | CREATE_COMPLETE      | -                                                                                           | -                             |          |             |
| 2023-08-29 23:49:59 UTC-0500               | MyApi               | CREATE_IN_PROGRESS   | Resource creation Initiated                                                                 | -                             |          |             |
| 2023-08-29 23:49:58 UTC-0500               | LambdaExecutionRole | CREATE_IN_PROGRESS   | Resource creation Initiated                                                                 | -                             |          |             |
| 2023-08-29 23:49:58 UTC-0500               | LambdaExecutionRole | CREATE_IN_PROGRESS   | -                                                                                           | -                             |          |             |
| 2023-08-29 23:49:58 UTC-0500               | MyApi               | CREATE_IN_PROGRESS   | -                                                                                           | -                             |          |             |
| 2023-08-29 23:49:55 UTC-0500               | HookTestStack       | CREATE_IN_PROGRESS   | User Initiated                                                                              | -                             |          |             |
| 2023-08-29 23:49:50 UTC-0500               | HookTestStack       | REVIEW_IN_PROGRESS   | User Initiated                                                                              | -                             |          |             |

Se hai abilitato i log per il tuo CloudFormation hook, puoi acquisire il risultato della valutazione dell'hook. Ecco un esempio di log per un hook con stato fallito, che indica che la funzione Lambda non è X-Ray abilitata:

▼

2023-08-29T23:50:17.574-05:00

ProgressEvent(status=<OperationStatus.FAILED: 'FAILED'>, errorCode=<HandlerErrorCode.NonCompliant: 'NonCompliant'...

ProgressEvent(status=<OperationStatus.FAILED: 'FAILED'>, errorCode=<HandlerErrorCode.NonCompliant: 'NonCompliant'>, message='Lambda function found with no tracing enabled : FAILED', result=None, callbackContext=None, callbackDelaySeconds=0, resourceModel=None, resourceModels=None, nextToken=None)

Copy

No newer events at this moment. Auto retry paused. [Resume](#)

Se lo sviluppatore sceglie di modificare l'IaC per aggiornare il valore `TracingConfig Mode` in `Active` ed eseguire nuovamente l'implementazione, l'hook viene eseguito correttamente e lo stack procede alla creazione della risorsa Lambda.

| Events (21)                                |                        |                    |                                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |
|--------------------------------------------|------------------------|--------------------|--------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <input type="text" value="Search events"/> |                        |                    |                                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |
| Timestamp                                  | Logical ID             | Status             | Status reason                                          | Hook invocations                                                                                                                                                                                                                                                                                                                                                                                                                                                      |  |
| 2023-08-29 23:56:52 UTC-0500               | LambdaApiGatewayInvoke | CREATE_IN_PROGRESS | -                                                      | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |
| 2023-08-29 23:56:52 UTC-0500               | MyLambdaFunction       | CREATE_COMPLETE    | -                                                      | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |
| 2023-08-29 23:56:44 UTC-0500               | MyLambdaFunction       | CREATE_IN_PROGRESS | Resource creation Initiated                            | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |
| 2023-08-29 23:56:44 UTC-0500               | MyLambdaFunction       | CREATE_IN_PROGRESS | Hook invocations complete. Resource creation initiated | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |
| 2023-08-29 23:56:43 UTC-0500               | MyLambdaFunction       | CREATE_IN_PROGRESS | -                                                      | AWS::Lambda::Function::Hook                                                                                                                                                                                                                                                                                                                                                                                                                                           |  |
| 2023-08-29 23:56:41 UTC-0500               | MyLambdaFunction       | CREATE_IN_PROGRESS | -                                                      | <div> <div>Hook invocation details</div> <div> <div>Hook name</div> <div>AWS::Lambda::Function::Hook</div> </div> <div> <div>Hook status</div> <div>HOOK_COMPLETE_SUCCEEDED</div> </div> <div> <div>Hook failure mode</div> <div>Fail</div> </div> <div> <div>Hook invocation point</div> <div>PRE_PROVISION</div> </div> <div> <div>Hook status reason</div> <div>Hook succeeded with message: Lambda function found with tracing enabled : PASS</div> </div> </div> |  |
| 2023-08-29 23:56:41 UTC-0500               | MyLambdaFunction       | CREATE_IN_PROGRESS | -                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |
| 2023-08-29 23:56:40 UTC-0500               | LambdaExecutionRole    | CREATE_COMPLETE    | -                                                      | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |
| 2023-08-29 23:56:25 UTC-0500               | MyApi                  | CREATE_COMPLETE    | -                                                      | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |
| 2023-08-29 23:56:25 UTC-0500               | MyApi                  | CREATE_IN_PROGRESS | Resource creation Initiated                            | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |
| 2023-08-29 23:56:24 UTC-0500               | LambdaExecutionRole    | CREATE_IN_PROGRESS | Resource creation Initiated                            | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |
| 2023-08-29 23:56:23 UTC-0500               | LambdaExecutionRole    | CREATE_IN_PROGRESS | -                                                      | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |

In questo modo, puoi implementare controlli preventivi AWS Config in modalità proattiva durante lo sviluppo e l'implementazione di risorse serverless nei tuoi account. AWS Integrando AWS Config le regole nella CI/CD pipeline, puoi identificare e opzionalmente bloccare le distribuzioni di risorse non

conformi, come le funzioni Lambda prive di una configurazione di tracciamento attiva. Ciò garantisce che solo le risorse conformi alle politiche di governance più recenti vengano implementate nei tuoi ambienti. AWS

## Rileva implementazioni e configurazioni Lambda non conformi con AWS Config

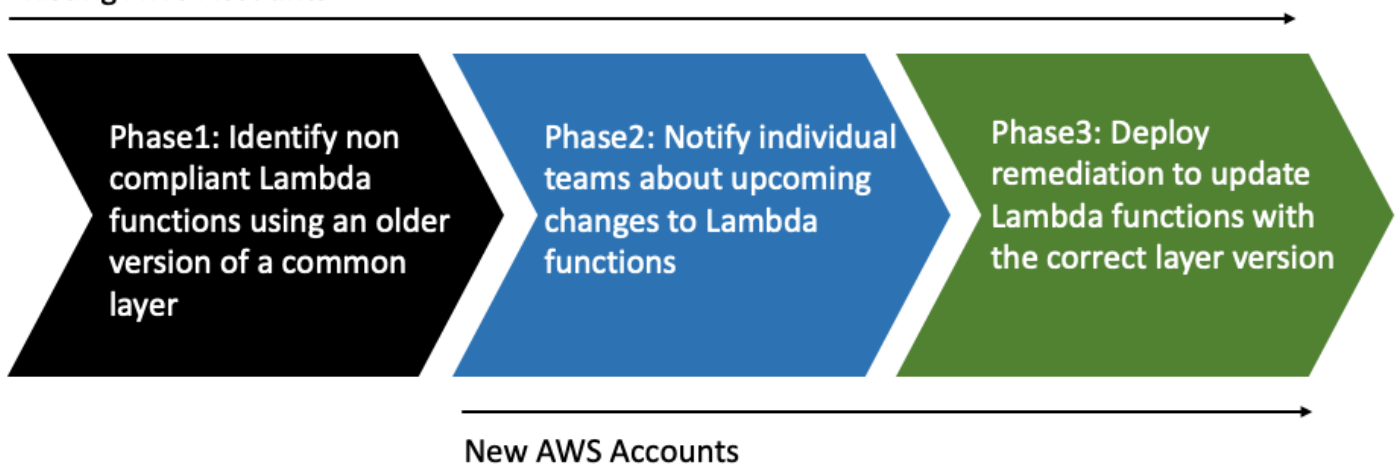
Oltre alla valutazione [proattiva](#), AWS Config può anche rilevare in modo reattivo le implementazioni e le configurazioni delle risorse che non sono conformi alle tue politiche di governance. Si tratta di una funzione importante, perché le policy di governance si evolvono man mano che l'organizzazione apprende e implementa nuove best practice.

Immagina uno scenario in cui imposti una nuova policy durante l'implementazione o l'aggiornamento delle funzioni Lambda: tutte le funzioni Lambda devono sempre utilizzare una versione di livello Lambda specifica e approvata. Puoi configurare AWS Config per monitorare le funzioni nuove o aggiornate per le configurazioni di livello. Se AWS Config rileva una funzione che non utilizza una versione di livello approvata, contrassegna la funzione come risorsa non conforme.

Facoltativamente, è possibile AWS Config configurare la correzione automatica della risorsa specificando un'azione di riparazione utilizzando un documento di automazione. AWS Systems Manager Ad esempio, è possibile scrivere un documento di automazione in Python utilizzando il AWS SDK per Python (Boto3), che aggiorna la funzione non conforme in modo che punti alla versione del livello approvata. Pertanto, AWS Config funge sia da controllo investigativo che correttivo, automatizzando la gestione della conformità.

Suddividiamo questo processo in tre importanti fasi di implementazione:

### Existing AWS Accounts



### Fase 1: identificazione delle risorse di accesso

Inizia attivandolo AWS Config su tutti i tuoi account e configurandolo per registrare le funzioni Lambda. AWS Config consente di osservare quando le funzioni Lambda vengono create



o aggiornate. Puoi quindi configurare [regole di policy personalizzate](#) per verificare la presenza di eventuali violazioni di policy specifiche, che utilizzano la sintassi AWS CloudFormation Guard . Le regole di Guard assumono la seguente forma generale:

```
rule name when condition { assertion }
```

Di seguito è riportato un esempio di regola che verifica che un livello non sia impostato su una versione di livello vecchia:

```
rule desiredlayer when configuration.layers !empty {  
    some configuration.layers[*].arn != CONFIG_RULE_PARAMETERS.OldLayerArn  
}
```

Analizziamo la sintassi e la struttura della regola:

- Nome della regola: il nome della regola nell'esempio presentato è `desiredlayer`.
- Condizione: questa clausola specifica la condizione in base alla quale la regola deve essere verificata. Nell'esempio fornito, la condizione è `configuration.layers !empty`. Ciò significa che la risorsa deve essere valutata solo quando la proprietà `layers` nella configurazione non è vuota.
- Asserzione: dopo la clausola `when`, un'asserzione determina che cosa verifica la regola. L'asserzione `some configuration.layers[*].arn != CONFIG_RULE_PARAMETERS.OldLayerArn` verifica se uno qualsiasi degli ARN di livello Lambda non corrisponde al valore `OldLayerArn`. Se non corrispondono, l'asserzione è vera e la regola viene rispettata; in caso contrario, ha esito negativo.

`CONFIG_RULE_PARAMETERS` è un set speciale di parametri configurato con la AWS Config regola. In questo caso, `OldLayerArn` è un parametro all'interno di `CONFIG_RULE_PARAMETERS`. Ciò consente agli utenti di fornire un valore di ARN specifico che ritengono vecchio o ritirato, quindi la regola verifica se alcune funzioni Lambda utilizzano l'ARN vecchio in questione.

## Fase 2: visualizzazione e progettazione

AWS Config raccoglie i dati di configurazione e li archivia in bucket Amazon Simple Storage Service (Amazon S3). Puoi utilizzare [Amazon Athena](#) per inviare query a tali dati direttamente dai bucket S3. Con Athena, puoi aggregare questi dati a livello dell'organizzazione, generando una visione onnicomprensiva delle configurazioni delle risorse in tutti gli account. Per configurare l'aggregazione

dei dati di configurazione delle risorse, consulta [Visualizing AWS Config data using Athena and Amazon Quick](#) nel blog Cloud Operations and Management. AWS

Di seguito è riportato un esempio di query Athena per identificare tutte le funzioni Lambda che utilizzano un particolare ARN di livello:

```
WITH unnested AS (
  SELECT
    item.awsaccountid AS account_id,
    item.awsregion AS region,
    item.configuration AS lambda_configuration,
    item.resourceid AS resourceid,
    item.resourcename AS resourcename,
    item.configuration AS configuration,
    json_parse(item.configuration) AS lambda_json
  FROM
    default.aws_config_configuration_snapshot,
    UNNEST(configurationitems) as t(item)
  WHERE
    "dt" = 'latest'
    AND item.resourcetype = 'AWS::Lambda::Function'
)

SELECT DISTINCT
  region as Region,
  resourcename as FunctionName,
  json_extract_scalar(lambda_json, '$.memorySize') AS memory_size,
  json_extract_scalar(lambda_json, '$.timeout') AS timeout,
  json_extract_scalar(lambda_json, '$.version') AS version
FROM
  unnested
WHERE
  lambda_configuration LIKE '%arn:aws:lambda:us-east-1:111122223333:layer:AnyGovernanceLayer:24%'
```

Ecco i risultati della query:

Query results

Query stats

Completed

Time in queue: 127 msRun time: 1.803 secData scanned: 239.40 KB

Results (27)

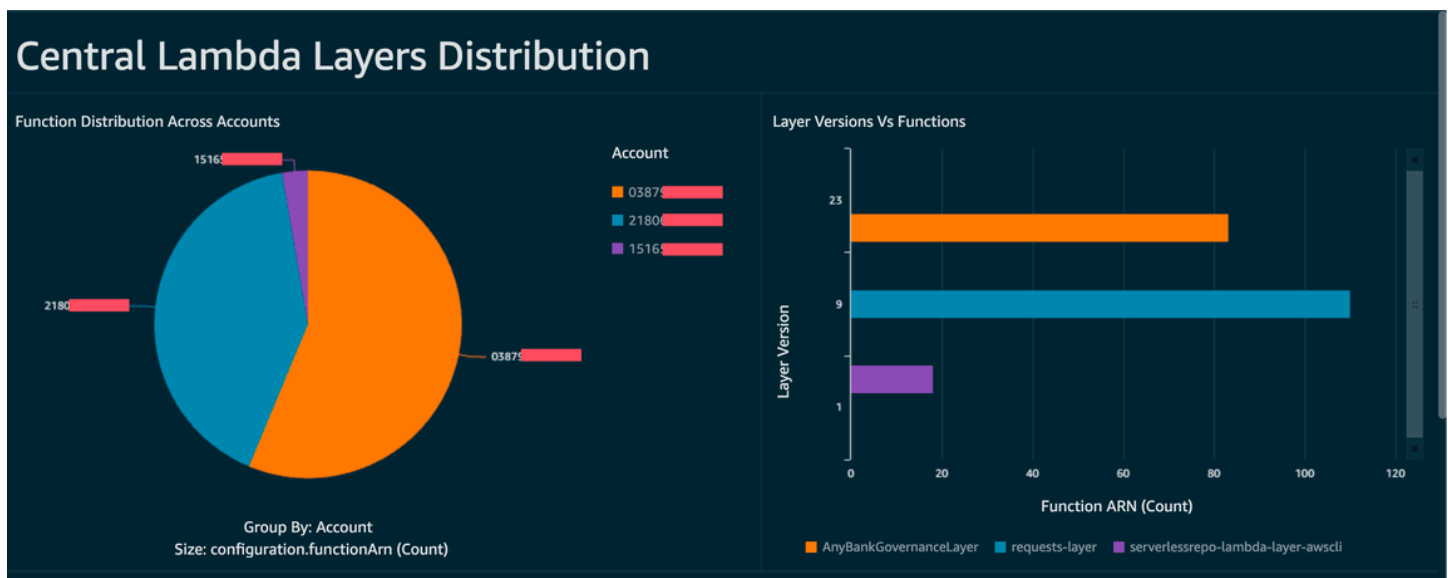
Copy

Download results

Search rows

| # | Region    | FunctionName                                                     | memory_size | timeout | version  |
|---|-----------|------------------------------------------------------------------|-------------|---------|----------|
| 1 | us-east-1 | UpdateUIForPublishEvents                                         | 128         | 18      | \$LATEST |
| 2 | us-east-1 | SchedulerCLI-InstanceSchedulerMain                               | 128         | 300     | \$LATEST |
| 3 | us-east-1 | my_functions_function10                                          | 128         | 3       | \$LATEST |
| 4 | us-east-1 | lex-web-ui-CognitoidentityP-CleanStackNameFunction-1TSORSH6L6YXQ | 128         | 300     | \$LATEST |
| 5 | us-east-1 | GetLatestArn                                                     | 128         | 3       | \$LATEST |
| 6 | us-east-1 | aws-python-http-api-project-dev-hello                            | 1024        | 6       | \$LATEST |
| 7 | us-east-1 | cloud9-MyTest-MyTest-688JGPVYP37L                                | 128         | 15      | \$LATEST |
| 8 | us-east-1 | my_functions_function1                                           | 128         | 3       | \$LATEST |
| 9 | us-east-1 | my_functions_function25                                          | 128         | 3       | \$LATEST |

Con i AWS Config dati aggregati in tutta l'organizzazione, puoi quindi creare una dashboard utilizzando Amazon Quick. <https://aws.amazon.com/quicksight/> Importando i risultati di Athena in Quick, puoi visualizzare in che misura le tue funzioni Lambda aderiscono alla regola della versione dei livelli. Questa dashboard può evidenziare le risorse conformi e non conformi. Ciò ti aiuta a determinare la policy di applicazione, come indicato nella [sezione successiva](#). L'immagine seguente è un esempio di dashboard che riporta la distribuzione delle versioni di livello applicate alle funzioni all'interno dell'organizzazione.



### Fase 3: implementazione e applicazione

Ora puoi opzionalmente abbinare la regola della versione del livello che hai creato nella [fase 1](#) a un'azione di correzione tramite un documento di automazione di Systems Manager, che crei come script Python scritto con AWS SDK per Python (Boto3). Lo script richiama l'azione

[UpdateFunctionConfiguration](#) API per ogni funzione Lambda, aggiornando la configurazione della funzione con il nuovo livello ARN. In alternativa, potresti fare in modo che lo script invii una richiesta pull al repository del codice per aggiornare l'ARN del livello. In questo modo anche le future implementazioni di codice vengono aggiornate con l'ARN di livello corretto.

## Firma del codice Lambda con AWS Signer

[AWS Signer](#) è un servizio di firma del codice completamente gestito che puoi utilizzare per convalidare il codice rispetto a una firma digitale per confermare che il codice sia inalterato e proveniente da un editore affidabile. AWS Signer può essere utilizzato insieme per verificare che le funzioni e AWS Lambda i livelli siano inalterati prima della distribuzione negli ambienti. AWS Ciò protegge l'organizzazione da malintenzionati che potrebbero aver ottenuto credenziali per creare nuove funzioni o aggiornare quelle esistenti.

Per impostare la firma del codice per le funzioni Lambda, comincia creando un bucket S3 con il controllo delle versioni abilitato. Successivamente, crea un profilo di firma con AWS Signer, specifica Lambda come piattaforma e quindi specifica un periodo di giorni in cui il profilo di firma è valido.

Esempio:

```
Signer:
  Type: AWS::Signer::SigningProfile
  Properties:
    PlatformId: AWSLambda-SHA384-ECDSA
    SignatureValidityPeriod:
      Type: DAYS
      Value: !Ref pValidDays
```

Quindi utilizza il profilo di firma e crea una configurazione di firma con Lambda. Devi specificare che cosa fare quando la configurazione di firma rileva un artefatto che non corrisponde alla firma digitale prevista: avvisare (ma consentire l'implementazione) o imporre (e bloccare l'implementazione). L'esempio seguente è configurato per imporre e bloccare le implementazioni.

```
SigningConfig:
  Type: AWS::Lambda::CodeSigningConfig
  Properties:
    AllowedPublishers:
      SigningProfileVersionArns:
        - !GetAtt Signer.ProfileVersionArn
    CodeSigningPolicies:
      UntrustedArtifactOnDeployment: Enforce
```

Ora hai AWS Signer configurato Lambda per bloccare le distribuzioni non attendibili. Supponiamo che tu abbia finito di codificare una richiesta di funzionalità e che ora stia aspettando di implementare la funzione. La prima fase consiste nel comprimere in formato zip il codice con le dipendenze

appropriate e quindi nel firmare l'artefatto utilizzando il profilo di firma che hai creato. Puoi farlo caricando l'artefatto zip nel bucket S3 e quindi avviando un processo di firma.

```
aws signer start-signing-job \
--source 's3={bucketName=your-versioned-bucket,key=your-prefix/your-zip-artifact.zip,version=QyaJ3c4qa50LXV.9VaZgXHlsGbvCXxpT}' \
--destination 's3={bucketName=your-versioned-bucket,prefix=your-prefix/}' \
--profile-name your-signer-id
```

Si ottiene un output come segue, dove `jobId` è l'oggetto creato nel bucket e nel prefisso di destinazione ed è l'ID a 12 cifre in cui `jobOwner` è stato eseguito Account AWS il lavoro.

```
{
  "jobId": "87a3522b-5c0b-4d7d-b4e0-4255a8e05388",
  "jobOwner": "111122223333"
}
```

Ora puoi implementare la funzione utilizzando l'oggetto S3 firmato e la configurazione di firma del codice che hai creato.

```
Fn:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: s3://your-versioned-bucket/your-prefix/87a3522b-5c0b-4d7d-b4e0-4255a8e05388.zip
    Handler: fn.handler
    Role: !GetAtt FnRole.Arn
    CodeSigningConfigArn: !Ref pSigningConfigArn
```

In alternativa, puoi testare l'implementazione di una funzione con l'artefatto zip di origine non firmato autentico. L'implementazione dovrebbe avere esito negativo con il seguente messaggio di errore:

```
Lambda cannot deploy the function. The function or layer might be signed using a
signature that the client is not configured to accept. Check the provided signature
for unsigned.
```

Se state creando e distribuendo le vostre funzioni utilizzando AWS Serverless Application Model (AWS SAM), il comando `package` gestisce il caricamento dell'artefatto zip su S3 e avvia anche il processo di firma e ottiene l'artefatto firmato. Puoi eseguire questa operazione con il comando e i parametri riportati di seguito:

```
sam package -t your-template.yaml \  
--output-template-file your-output.yaml \  
--s3-bucket your-versioned-bucket \  
--s3-prefix your-prefix \  
--signing-profiles your-signer-id
```

AWS Signer ti aiuta a verificare che gli artefatti zip distribuiti nei tuoi account siano affidabili per la distribuzione. Puoi includere il processo precedente nelle tue CI/CD pipeline e richiedere che a tutte le funzioni sia associata una configurazione di firma del codice utilizzando le tecniche descritte negli argomenti precedenti. Utilizzando la firma del codice con le implementazioni delle funzioni Lambda, impedisce a malintenzionati che potrebbero aver ottenuto le credenziali per creare o aggiornare le funzioni di iniettare codice dannoso nelle funzioni.

## Automatizzare le valutazioni di sicurezza per Lambda con Amazon Inspector

[Amazon Inspector](#) è un servizio di gestione delle vulnerabilità che scansiona continuamente i carichi di lavoro alla ricerca di vulnerabilità del software ed esposizione alla rete non intenzionale. Amazon Inspector crea un esito che descrive la vulnerabilità, identifica la risorsa interessata, valuta la gravità della vulnerabilità e fornisce indicazioni per la correzione.

Il supporto di Amazon Inspector fornisce valutazioni continue e automatizzate delle vulnerabilità di sicurezza per le funzioni e i livelli Lambda. Amazon Inspector offre due tipi di scansione per Lambda:

- Scansione standard Lambda (impostazione predefinita): esegue la scansione delle dipendenze dell'applicazione all'interno di una funzione Lambda e dei relativi livelli alla ricerca di [vulnerabilità dei pacchetti](#).
- Scansione del codice Lambda: esegue la scansione del codice dell'applicazione personalizzato nelle funzioni e nei livelli alla ricerca di [vulnerabilità del codice](#). Puoi attivare la scansione standard Lambda da sola o insieme alla scansione del codice Lambda.

Per abilitare Amazon Inspector, accedi alla console [Amazon Inspector](#), espandi la sezione Impostazioni e scegli Gestione dell'account. Nella scheda Account, scegli Attiva, quindi seleziona una delle opzioni di scansione.

Puoi abilitare Amazon Inspector per più account e delegare le autorizzazioni a gestire Amazon Inspector per l'organizzazione ad account specifici durante la configurazione di Amazon Inspector. Durante l'abilitazione, devi concedere le autorizzazioni ad Amazon Inspector creando il ruolo: `AWSServiceRoleForAmazonInspector2`. Nella console di Amazon Inspector, puoi creare questo ruolo utilizzando un'opzione in un solo passaggio.

Per la scansione standard Lambda, Amazon Inspector avvia scansioni di vulnerabilità delle funzioni Lambda nelle situazioni seguenti:

- Non appena Amazon Inspector rileva una funzione Lambda esistente.
- Quando si implementa una nuova funzione Lambda.
- Quando si implementa un aggiornamento al codice dell'applicazione o alle dipendenze di una funzione Lambda esistente o dei relativi livelli.
- Ogni volta che Amazon Inspector aggiunge un nuovo elemento di vulnerabilità ed esposizioni comuni (common vulnerabilities and exposures, CVE) al suo database e tale CVE è pertinente alla funzione.



Per la scansione del codice Lambda, Amazon Inspector valuta il codice dell'applicazione della funzione Lambda utilizzando ragionamento automatico e machine learning che analizzano il codice dell'applicazione per quanto riguarda la conformità di sicurezza generale. Se rileva una vulnerabilità nel codice dell'applicazione della funzione Lambda, Amazon Inspector produce un esito di Vulnerabilità del codice dettagliato. Per un elenco dei possibili rilevamenti, consulta la [Amazon CodeGuru Detector Library](#).

Per visualizzare gli esiti, accedi alla [console Amazon Inspector](#). Nel menu Esiti, seleziona Per funzione Lambda per visualizzare i risultati della scansione di sicurezza eseguita sulle funzioni Lambda.

Per escludere una funzione Lambda dalla scansione standard, etichetta la funzione con la seguente coppia chiave-valore:

- Key:InspectorExclusion
- Value:LambdaStandardScanning

Per escludere una funzione Lambda dalle scansioni del codice, etichetta la funzione con la seguente coppia chiave-valore:

- Key:InspectorCodeExclusion
- Value:LambdaCodeScanning

Ad esempio, come mostrato nell'immagine seguente, Amazon Inspector rileva in automatico le vulnerabilità e classifica gli esiti di tipo Vulnerabilità del codice, il che indica che la vulnerabilità si trova nel codice della funzione e non in una delle librerie dipendenti dal codice. Puoi controllare questi dettagli per una funzione specifica o per più funzioni contemporaneamente.

Findings (2)

Choose a row to view the finding details. All findings are related to this instance.

Active ▾

Q

Add filter

Resource ID EQUALS

arn:aws:lambda:us-east-1: function:code\_scanning\_python:\$LATEST

Clear filters

< 1 > ⚙

|                       | Severity ▾ | Title                                         | Type ▾             | Age ▾      | Status |
|-----------------------|------------|-----------------------------------------------|--------------------|------------|--------|
| <input type="radio"/> | High       | CWE-200 - Insecure Socket Bind                | Code Vulnerability | 10 minutes | Active |
| <input type="radio"/> | High       | Overriding environment variables that are res | Code Vulnerability | 10 minutes | Active |

Puoi approfondire ciascuno degli esiti e scoprire come risolvere il problema.

## Overriding environment variables that are reserved by AWS Lambda might lead to unexpected behavior.



Finding ID: [arn:aws:inspector2:us-east-1: \[REDACTED\]:finding/\[REDACTED\]](#)

Overriding environment variables that are reserved by AWS Lambda might lead to unexpected behavior or failure of the Lambda function.

### Finding overview

|                                 |                                                                                                                       |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| AWS account ID                  | [REDACTED]                                                                                                            |
| Severity                        | High                                                                                                                  |
| Type                            | Code Vulnerability                                                                                                    |
| Detector name <a href="#">↗</a> | <a href="#">Override of reserved variable names in a Lambda function</a>                                              |
| Relevant CWE <a href="#">↗</a>  | --                                                                                                                    |
| Rule ID <a href="#">↗</a>       | <a href="#">Rule-434311</a>                                                                                           |
| Detector tags                   | #availability, #aws-python-sdk, #aws-lambda, #data-integrity, #maintainability, #security, #security-context, #python |
| Fix available                   | Yes                                                                                                                   |
| Created at                      | March 29, 2023 10:08 AM (UTC-04:00)                                                                                   |

### Vulnerability details

File path `lambda_function.py`

### Vulnerability location

```
3 import socket
4
5 def lambda_handler(event, context):
6
7     # print("Scenario 1");
8     os.environ['_HANDLER'] = 'hello'
9     # print("Scenario 1 ends")
10
11     # print("Scenario 2");
12     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13     s.bind(('',0))
```

### Suggested remediation

Your code attempts to override an environment variable that is reserved by the Lambda runtime environment. This can lead to unexpected behavior and might break the execution of your Lambda function.

Mentre lavori con le funzioni Lambda, assicurati di rispettare le convenzioni di denominazione delle funzioni Lambda. Per ulteriori informazioni, consulta [Utilizzo delle variabili di ambiente Lambda](#).

La responsabilità sui suggerimenti di riparazione che accetti è tua. Esamina sempre i suggerimenti di riparazione prima di accettarli. Per garantire che il codice funzioni come previsto, potrebbe essere necessario apportare modifiche ai suggerimenti di riparazione.

## Implementazione dell'osservabilità per la sicurezza e la conformità Lambda

AWS Config è uno strumento utile per trovare e correggere risorse Serverless non conformi AWS . Ogni modifica apportata alle risorse serverless viene registrata in AWS Config. Inoltre, con AWS Config, è possibile archiviare i dati delle istantanee di configurazione su S3.

Puoi utilizzare Amazon Athena e Amazon Quick per creare dashboard e visualizzare i dati. AWS Config In [Rileva implementazioni e configurazioni Lambda non conformi con AWS Config](#), abbiamo discusso come visualizzare una determinata configurazione quali i livelli Lambda. Il presente argomento approfondisce tali concetti.

### Visibilità sulle configurazioni Lambda

Puoi utilizzare le query per estrarre configurazioni importanti come Account AWS ID, regione, configurazione di AWS X-Ray tracciamento, configurazione VPC, dimensione della memoria, runtime e tag. Di seguito è riportato un esempio di query che puoi utilizzare per ottenere queste informazioni da Athena:

```
WITH unnested AS (
  SELECT
    item.awsaccountid AS account_id,
    item.awsregion AS region,
    item.configuration AS lambda_configuration,
    item.resourceid AS resourceid,
    item.resourcename AS resourcename,
    item.configuration AS configuration,
    json_parse(item.configuration) AS lambda_json
  FROM
    default.aws_config_configuration_snapshot,
    UNNEST(configurationitems) as t(item)
  WHERE
    "dt" = 'latest'
    AND item.resourcetype = 'AWS::Lambda::Function'
)

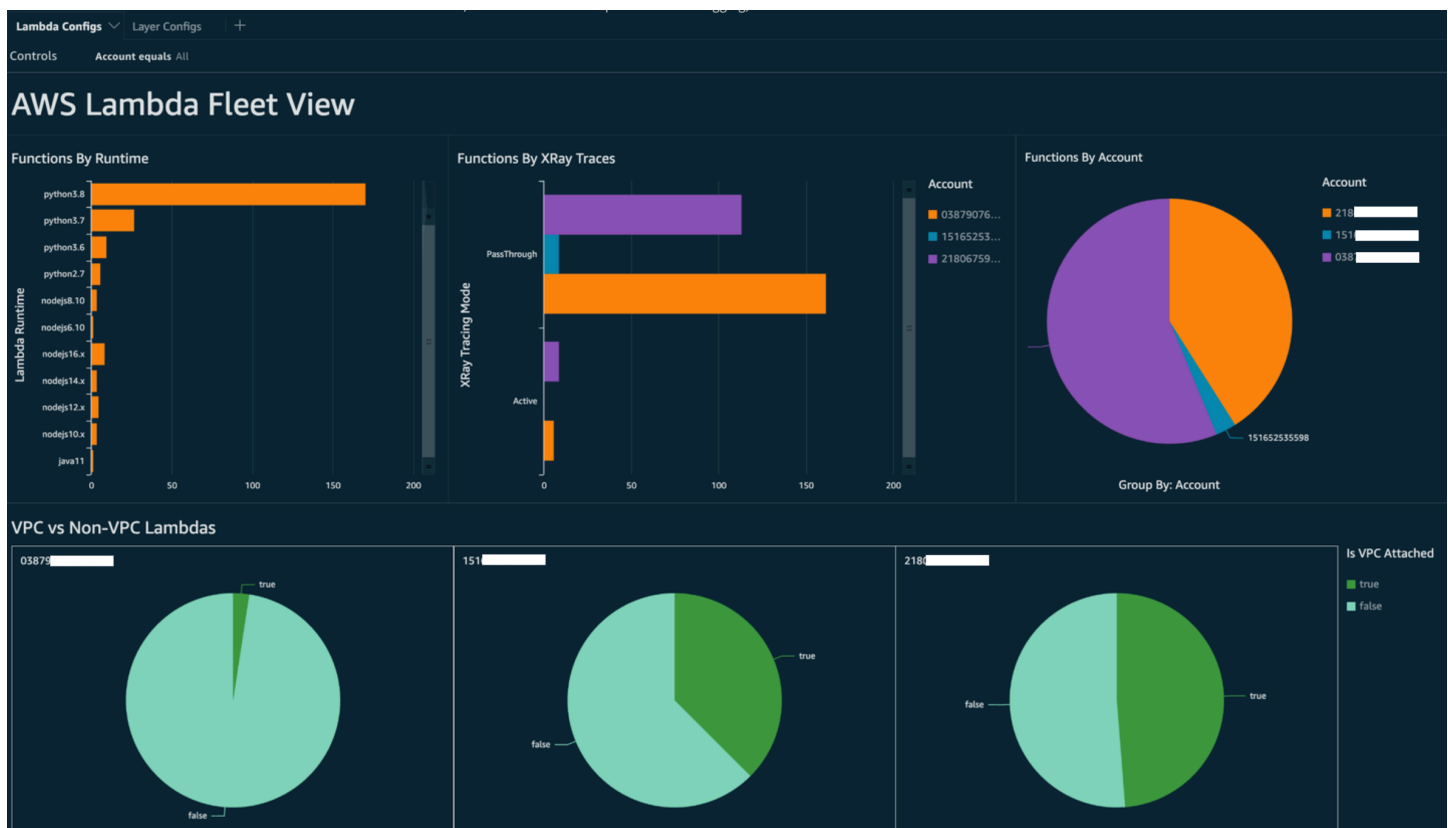
SELECT DISTINCT
  account_id,
  tags,
  region as Region,
  resourcename as FunctionName,
  json_extract_scalar(lambda_json, '$.memorySize') AS memory_size,
  json_extract_scalar(lambda_json, '$.timeout') AS timeout,
```

```

json_extract_scalar(lambda_json, '$.runtime') AS version
json_extract_scalar(lambda_json, '$.vpcConfig.SubnetIds') AS vpcConfig
json_extract_scalar(lambda_json, '$.tracingConfig.mode') AS tracingConfig
FROM
  unnested

```

Puoi utilizzare la query per creare una dashboard rapida e visualizzare i dati. Per aggregare i dati di configurazione AWS delle risorse, creare tabelle in Athena e creare dashboard rapide sui dati di Athena, consulta [Visualizing AWS Config data using Athena and Amazon Quick](#) sul blog Cloud Operations and Management. AWS In particolare, questa query recupera anche le informazioni dei tag per le funzioni. Ciò consente di ottenere maggiori approfondimenti sui carichi di lavoro e sugli ambienti, soprattutto se impieghi tag personalizzati.



Per ulteriori informazioni sulle azioni che puoi intraprendere, consulta la sezione [Risoluzione degli esiti di osservabilità](#) più avanti in questo argomento.

## Visibilità sulla conformità Lambda

Con i dati generati da AWS Config, puoi creare dashboard a livello di organizzazione per monitorare la conformità. Ciò consente il tracciamento e il monitoraggio coerenti di:

- Pacchetti di conformità per punteggio di conformità
- Regole per risorse non conformi
- Compliance status (Stato di conformità)

AWS Config

Dashboard

Conformance packs

Rules

Resources

▼ Aggregators

Conformance packs

Rules

Resources

Authorizations

Advanced queries

Settings

What's new

Documentation

Partners

FAQs

Pricing

AWS Config

> Dashboard

Dashboard

Conformance Packs by Compliance Score

| Conformance pack     | Compliance score           |
|----------------------|----------------------------|
| MyNewConformancePack | <div><div></div></div> 37% |

Compliance status

Rules

⚠️ 6 Noncompliant rule(s)

✅ 7 Compliant rule(s)

Resources

⚠️ 100+ Noncompliant resource(s)

✅ 82 Compliant resource(s)

Noncompliant rules by noncompliant resource count

| Name                           | Compliance                      |
|--------------------------------|---------------------------------|
| lambda-function-settings-ch... | ⚠️ 25+ Noncompliant resource(s) |
| lambda-dlq-check-conforma...   | ⚠️ 25+ Noncompliant resource(s) |
| lambda-inside-vpc-conforma...  | ⚠️ 25+ Noncompliant resource(s) |
| lambda-vpc-multi-az-check-...  | ⚠️ 25+ Noncompliant resource(s) |
| lambda-function-settings-ch... | ⚠️ 14 Noncompliant resource(s)  |

View all noncompliant rules

Controlla ogni regola per identificare le risorse non conformi alla stessa. Ad esempio, se la tua organizzazione impone che tutte le funzioni Lambda debbano essere associate a un VPC e se

Governance

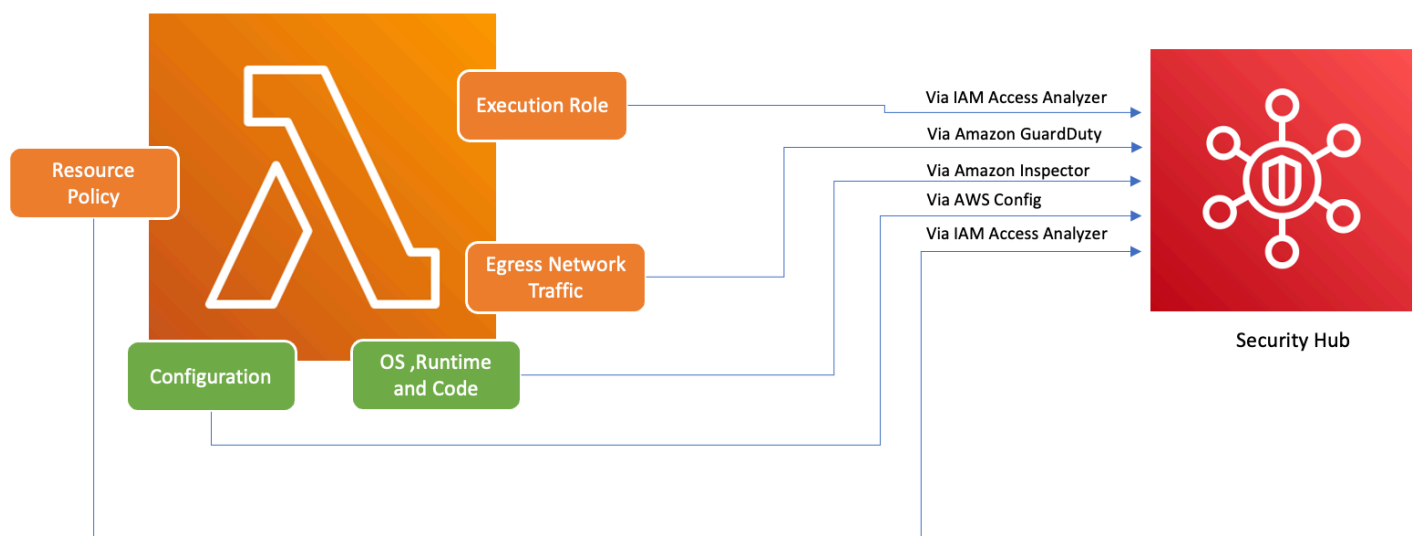
2405

hai implementato una AWS Config regola per identificare la conformità, puoi selezionare la regola nell'elenco precedente. `lambda-inside-vpc`

| Resources in scope                                                                      |                                        |                |  |
|-----------------------------------------------------------------------------------------|----------------------------------------|----------------|--|
| <div> <div>All</div> <div>All</div> <div>Compliant</div> <div>Noncompliant</div> </div> |                                        |                |  |
| Type                                                                                    | Annotation                             | Compliance     |  |
| Lambda Function                                                                         | -                                      | ✔ Compliant    |  |
| my_functions_function46                                                                 | -                                      | ✔ Compliant    |  |
| my_functions_function47                                                                 | -                                      | ✔ Compliant    |  |
| my_functions_function49                                                                 | -                                      | ✔ Compliant    |  |
| my_functions_function50                                                                 | -                                      | ✔ Compliant    |  |
| my_functions_function6                                                                  | -                                      | ✔ Compliant    |  |
| my_functions_function7                                                                  | -                                      | ✔ Compliant    |  |
| my_functions_function8                                                                  | -                                      | ✔ Compliant    |  |
| ConfigQueryLambda                                                                       | This AWS Lambda function is not in ... | ⚠ Noncompliant |  |
| DormamuLambda                                                                           | This AWS Lambda function is not in ... | ⚠ Noncompliant |  |

Per ulteriori informazioni sulle azioni che puoi intraprendere, consulta la sezione [Risoluzione degli esiti di osservabilità](#) di seguito.

Visibilità nei limiti delle funzioni Lambda utilizzando Security Hub CSPM



Per garantire che AWS i servizi, tra cui Lambda, vengano utilizzati in modo sicuro, sono state AWS introdotte le Foundational Security Best Practices v1.0.0. Questa serie di best practice fornisce



linee guida chiare per proteggere risorse e dati nell' AWS ambiente, sottolineando l'importanza di mantenere un elevato livello di sicurezza. A questo AWS Security Hub CSPM si aggiunge l'offerta di un centro unificato per la sicurezza e la conformità. Aggrega, organizza e dà priorità ai risultati di sicurezza di più AWS servizi come Amazon Inspector e Amazon. AWS Identity and Access Management Access Analyzer GuardDuty

Se disponi di Security Hub CSPM, Amazon Inspector e IAM Access Analyzer GuardDuty abilitati all'interno della tua AWS organizzazione, Security Hub CSPM aggrega automaticamente i risultati di questi servizi. Prendiamo ad esempio Amazon Inspector. Utilizzando Security Hub CSPM, puoi identificare in modo efficiente le vulnerabilità del codice e dei pacchetti nelle funzioni Lambda. Nella console Security Hub CSPM, vai alla sezione inferiore denominata Ultime scoperte dalle integrazioni. AWS Qui puoi visualizzare e analizzare i risultati provenienti da vari servizi integrati. AWS

| Latest findings from AWS integrations                                                                      |  |                                                |
|------------------------------------------------------------------------------------------------------------|--|------------------------------------------------|
| <b>Amazon GuardDuty</b><br><a href="#">Open the GuardDuty console</a>                                      |  | No findings                                    |
| <b>Amazon Inspector</b><br><a href="#">Open the Inspector console</a>                                      |  | 23 minutes ago<br><a href="#">See findings</a> |
| <b>Amazon Macie</b><br><a href="#">Open the Macie console</a>                                              |  | No findings                                    |
| <b>AWS Health</b><br><a href="#">Open the Personal Health Dashboard</a>                                    |  | No findings                                    |
| <b>AWS IAM Access Analyzer</b><br><a href="#">Open the IAM Access Analyzer console</a>                     |  | No findings                                    |
| <b>AWS Systems Manager Patch Manager</b><br><a href="#">Open the Systems Manager Patch Manager console</a> |  | No findings                                    |
| <b>AWS Firewall Manager</b><br><a href="#">Open the Firewall Manager console</a>                           |  | No findings                                    |

Per visualizzare i dettagli, scegli il link Vedi risultati nella seconda colonna. Viene visualizzato un elenco di risultati filtrati per prodotto, ad esempio Amazon Inspector. Per limitare la ricerca alle funzioni Lambda, imposta ResourceType su AwsLambdaFunction. Vengono visualizzati gli esiti di Amazon Inspector relativi alle funzioni Lambda.

Security Hub > Findings

**Findings (20+)** Actions Workflow status Create insight

A finding is a security issue or a failed security check.

Q Add filter

Product name is Inspector X Resource type is AwsLambdaFunction X Workflow status is NEW X Workflow status is NOTIFIED X Record state is ACTIVE X Clear filters

< 1 ... >

| <input type="checkbox"/> | Severity                                | Workflow status | Record State | Region    | Account Id | Company | Product   | Title                                   | Resource                    | Compliance Status | Updated at     |
|--------------------------|-----------------------------------------|-----------------|--------------|-----------|------------|---------|-----------|-----------------------------------------|-----------------------------|-------------------|----------------|
| <input type="checkbox"/> | <span style="color: red;">■</span> HIGH | NEW             | ACTIVE       | us-east-1 | 218        | Amazon  | Inspector | <a href="#">CWE-117 - Log injection</a> | Lambda Function<br>\$LATEST |                   | 27 minutes ago |
| <input type="checkbox"/> | <span style="color: red;">■</span> HIGH | NEW             | ACTIVE       | us-east-1 | 218        | Amazon  | Inspector | <a href="#">CWE-117 - Log injection</a> | Lambda Function<br>\$LATEST |                   | 27 minutes ago |
| <input type="checkbox"/> | <span style="color: red;">■</span> HIGH | NEW             | ACTIVE       | us-east-1 | 218        | Amazon  | Inspector | <a href="#">CWE-117 - Log injection</a> | Lambda Function<br>\$LATEST |                   | 27 minutes ago |
| <input type="checkbox"/> | <span style="color: red;">■</span> HIGH | NEW             | ACTIVE       | us-east-1 | 218        | Amazon  | Inspector | <a href="#">CWE-117 - Log injection</a> | Lambda Function<br>\$LATEST |                   | 27 minutes ago |

Perché GuardDuty, è possibile identificare modelli di traffico di rete sospetti. Tali anomalie potrebbero suggerire l'esistenza di codice potenzialmente dannoso all'interno della funzione Lambda.

Con Sistema di analisi degli accessi IAM, puoi controllare le policy, in particolare quelle con istruzioni condizionali che concedono l'accesso alle funzioni a entità esterne. Inoltre, IAM Access Analyzer valuta le autorizzazioni impostate quando si utilizza l'[AddPermission](#) operazione nell'API Lambda insieme a un. EventSourceToken

## Risoluzione degli esiti di osservabilità

Data l'ampia gamma di configurazioni possibili per le funzioni Lambda e i relativi requisiti distinti, una soluzione di automazione standardizzata per la riparazione potrebbe non essere adatta a ogni situazione. Inoltre, le modifiche vengono implementate in modo diverso nei vari ambienti. Se riscontri una configurazione che sembra non conforme, tieni presente le seguenti linee guida:

### 1. Strategia di assegnazione tag

Consigliamo di implementare una strategia di assegnazione tag completa. A ogni funzione Lambda si dovrebbero assegnare tag con informazioni chiave come le seguenti:

- Proprietario: la persona o il team responsabile della funzione.
- Ambiente: produzione, staging, sviluppo o ambiente di sperimentazione (sandbox).
- Applicazione: il contesto più ampio a cui appartiene la funzione, se applicabile.

## 2. Contatto del proprietario

Anziché automatizzare le modifiche sostanziali (come la regolazione della configurazione del VPC), contatta in modo proattivo i proprietari delle funzioni non conformi (identificati dal tag del proprietario) fornendo loro il tempo sufficiente per adottare una delle misure seguenti:

- Regolare le configurazioni non conformi sulle funzioni Lambda.
- Fornire una spiegazione e richiedere un'eccezione oppure perfezionare gli standard di conformità.

## 3. Mantenimento di un database di gestione della configurazione (CMDB)

Sebbene i tag possano fornire un contesto immediato, mantenere un CMDB centralizzato può fornire maggiori approfondimenti. Il database può contenere informazioni più granulari in merito a ogni funzione Lambda, alle relative dipendenze e altri metadati critici. Un CMDB è una risorsa di valore inestimabile per l'audit, i controlli di conformità e l'identificazione dei proprietari delle funzioni.

Poiché il panorama dell'infrastruttura serverless è in continua evoluzione, è essenziale adottare un atteggiamento proattivo nei confronti del monitoraggio. Con strumenti come AWS Config Security Hub CSPM e Amazon Inspector, è possibile identificare rapidamente potenziali anomalie o configurazioni non conformi. Tuttavia, gli strumenti da soli non possono garantire la conformità totale o configurazioni ottimali. È fondamentale abbinare tali strumenti a processi e best practice ben documentati.

- Ciclo di feedback: una volta intraprese le misure correttive, assicurati che vi sia un ciclo di feedback. Ciò significa riesaminare su base periodica le risorse non conformi per confermare se sono state aggiornate o se presentano ancora gli stessi problemi.
- Documentazione: documenta sempre le osservazioni, le azioni intraprese e le eventuali eccezioni concesse. Una documentazione adeguata non solo aiuta durante gli audit, ma contribuisce anche a migliorare il processo per ottenere una maggiore conformità e sicurezza in futuro.
- Formazione e consapevolezza: assicurati che tutte le parti interessate, in particolare i responsabili delle funzioni Lambda, ricevano una formazione periodica e siano informate su best practice, policy dell'organizzazione e obblighi di conformità. Workshop, webinar o sessioni di formazione regolari possono dare un contributo notevole per garantire che tutti siano sulla stessa lunghezza d'onda in materia di sicurezza e conformità.

In conclusione, mentre gli strumenti e le tecnologie forniscono funzionalità solide per rilevare e segnalare potenziali problemi, l'elemento umano (comprensione, comunicazione, formazione e documentazione) rimane di cruciale importanza. Insieme, questi aspetti formano una combinazione molto efficace per garantire che le funzioni Lambda e l'infrastruttura in senso lato rimangano conformi, sicure e ottimizzate per le esigenze aziendali.

## Convalida della conformità per AWS Lambda

Third-party gli auditor valutano la sicurezza e la conformità di nell' AWS Lambda ambito di più programmi di AWS conformità. Questi includono SOC, PCI, FedRAMP, HIPAA e altri.

Per un elenco dei AWS servizi che rientrano nell'ambito di specifici programmi di conformità, consulta [AWS i servizi compresi in base al programma](#) di conformità. Per informazioni generali, consultare [Programmi per la conformità di AWS](#).

È possibile scaricare i report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta [Download dei report in AWS Artifact](#).

La tua responsabilità di conformità durante l'utilizzo di Lambda è determinata dalla riservatezza dei dati, dagli obiettivi dell'azienda e dalle leggi e normative vigenti in materia. Puoi implementare controlli di governance per garantire che le funzioni Lambda della tua azienda soddisfino i requisiti di conformità. Per ulteriori informazioni, consulta [Creare una strategia di governance per le funzioni e i livelli Lambda](#).

Per informazioni su come AWS protegge l'infrastruttura globale che esegue Lambda, consulta [the section called "Sicurezza dell'infrastruttura"](#).

## Resilienza in AWS Lambda

L'infrastruttura AWS globale è costruita attorno a AWS regioni e zone di disponibilità. AWS Le regioni forniscono più zone di disponibilità fisicamente separate e isolate, collegate a reti a bassa latenza, alta produttività e altamente ridondanti. Con le zone di disponibilità, è possibile progettare e gestire applicazioni e database che eseguono il failover automatico tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture tradizionali a data center singolo o multiplo.

Per ulteriori informazioni su AWS regioni e zone di disponibilità, vedi infrastruttura globale. [AWS](#)

Oltre all'infrastruttura AWS globale, Lambda offre diverse funzionalità per aiutare a supportare la resilienza dei dati e le esigenze di backup.

- Gestione delle versioni – È possibile usare la gestione delle versioni in Lambda per salvare il codice e la configurazione in fase di sviluppo. Insieme agli alias, puoi utilizzare il controllo delle versioni per eseguire blue/green distribuzioni continue. Per informazioni dettagliate, vedi [Gestire le versioni delle funzioni Lambda](#).

- **Dimensionamento** – Quando la funzione riceve una richiesta durante l'elaborazione di una richiesta precedente, Lambda avvia un'altra istanza della funzione per gestire l'aumento del carico. Lambda dimensiona automaticamente per gestire 1.000 esecuzioni simultanee per regione, un [quota](#) che può essere aumentata se necessario. Per informazioni dettagliate, vedi [Informazioni sulla scalabilità della funzione Lambda](#).
- **Elevata disponibilità** – Lambda esegue la funzione in più zone di disponibilità per assicurare che sia disponibile per elaborare eventi in caso di interruzione del servizio in una specifica zona. Se si configura la tua funzione affinché si connetta a un cloud privato virtuale (VPC) nel proprio account, specificare le sottoreti in più zone di disponibilità per garantire un'elevata disponibilità. Per informazioni dettagliate, vedi [Consentire alle funzioni Lambda l'accesso alle risorse in un Amazon VPC](#).
- **Simultaneità riservata** – Per garantire che la funzione sia sempre in grado di ridimensionare le risorse per gestire le richieste aggiuntive, è possibile riservare la simultaneità. Impostare la simultaneità riservata per una funzione garantisce che questa possa ridimensionarsi fino a un dato numero di chiamate simultanee, senza tuttavia superarlo. In questo modo non andranno perse delle richieste a causa di altre funzioni onerose in termini di disponibilità simultanea. Per informazioni dettagliate, vedi [Configurazione della simultaneità riservata per una funzione](#).
- **Tentativi ripetuti** – Per le invocazioni asincrone e un sottoinsieme di invocazioni attivate da altri servizi, Lambda esegue automaticamente dei nuovi tentativi in caso di errore con specifici ritardi tra i tentativi successivi. Altri client e Servizi AWS che invocano le funzioni in modo sincrono sono responsabili dell'esecuzione di tentativi ripetuti. Per informazioni dettagliate, vedi [Informazioni sul comportamento relativo ai nuovi tentativi in Lambda](#).
- **Dead-letter coda**: per le chiamate asincrone, puoi configurare Lambda in modo che invii richieste a una coda di lettere morte se tutti i tentativi falliscono. Una coda DLQ è un argomento Amazon SNS o una coda Amazon SQS che riceve gli eventi a scopo di risoluzione dei problemi o rielaborazione. Per informazioni dettagliate, vedi [Aggiunta di una coda DLQ](#).

Per ulteriori informazioni sulla rete e sull'infrastruttura fisica che protegge Lambda, consulta [the section called “Sicurezza dell'infrastruttura”](#)

## Sicurezza dell'infrastruttura in AWS Lambda

Come servizio gestito, AWS Lambda è protetto dalla sicurezza di rete globale AWS. Per informazioni sui servizi di sicurezza AWS e su come AWS protegge l'infrastruttura, consulta la pagina [Sicurezza del cloud AWS](#). Per progettare l'ambiente AWS utilizzando le best practice per la sicurezza

dell'infrastruttura, consulta la pagina [Protezione dell'infrastruttura](#) nel Pilastro della sicurezza di AWS Well-Architected Framework.

Utilizza le chiamate API pubblicate da AWS per accedere a Lambda tramite la rete. I client devono supportare quanto segue:

- Transport Layer Security (TLS). È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

## Protezione dei carichi di lavoro con endpoint pubblici

Per i carichi di lavoro accessibili pubblicamente, AWS fornisce una serie di funzionalità e servizi che possono aiutare a mitigare determinati rischi. In questa sezione viene trattata l'autenticazione e l'autorizzazione degli utenti delle applicazioni e la protezione degli endpoint delle API.

### Autenticazione e autorizzazione

L'autenticazione si riferisce all'identità mentre l'autorizzazione si riferisce alle operazioni. Usa l'autenticazione per controllare chi può richiamare una funzione Lambda, quindi usa l'autorizzazione per controllare cosa può fare. Per molte applicazioni, IAM è sufficiente per gestire entrambi i meccanismi di controllo.

Per le applicazioni con utenti esterni, come le applicazioni Web o per dispositivi mobili, è comune utilizzare [JSON Web Tokens](#) (JWT) per gestire l'autenticazione e l'autorizzazione. A differenza della tradizionale gestione delle password basata su server, i JWT vengono trasmessi dal client a ogni richiesta. Sono un modo sicuro dal punto di vista della crittografia per verificare l'identità e le affermazioni utilizzando i dati trasmessi dal client. Per Lambda-based le applicazioni, puoi proteggere ogni chiamata a ciascun endpoint API senza affidarti a un server centrale per l'autenticazione.

Puoi [implementare JWT con Amazon Cognito](#), un servizio di elenchi utenti in grado di gestire la registrazione, l'autenticazione, il ripristino dell'account e altre comuni operazioni di gestione degli account. [Framework Amplify](#) fornisce librerie per semplificare l'integrazione di questo servizio nell'applicazione frontend. Puoi anche prendere in considerazione servizi di partner di terze parti come Auth0. <https://auth0.com/>

Dato il ruolo fondamentale di sicurezza di un servizio di identity provider, è importante utilizzare strumenti professionali per salvaguardare l'applicazione. Non è consigliabile scrivere servizi propri per

gestire l'autenticazione o l'autorizzazione. Qualsiasi vulnerabilità nelle librerie personalizzate potrebbe avere implicazioni significative per la sicurezza del carico di lavoro e dei relativi dati.

## Protezione degli endpoint API

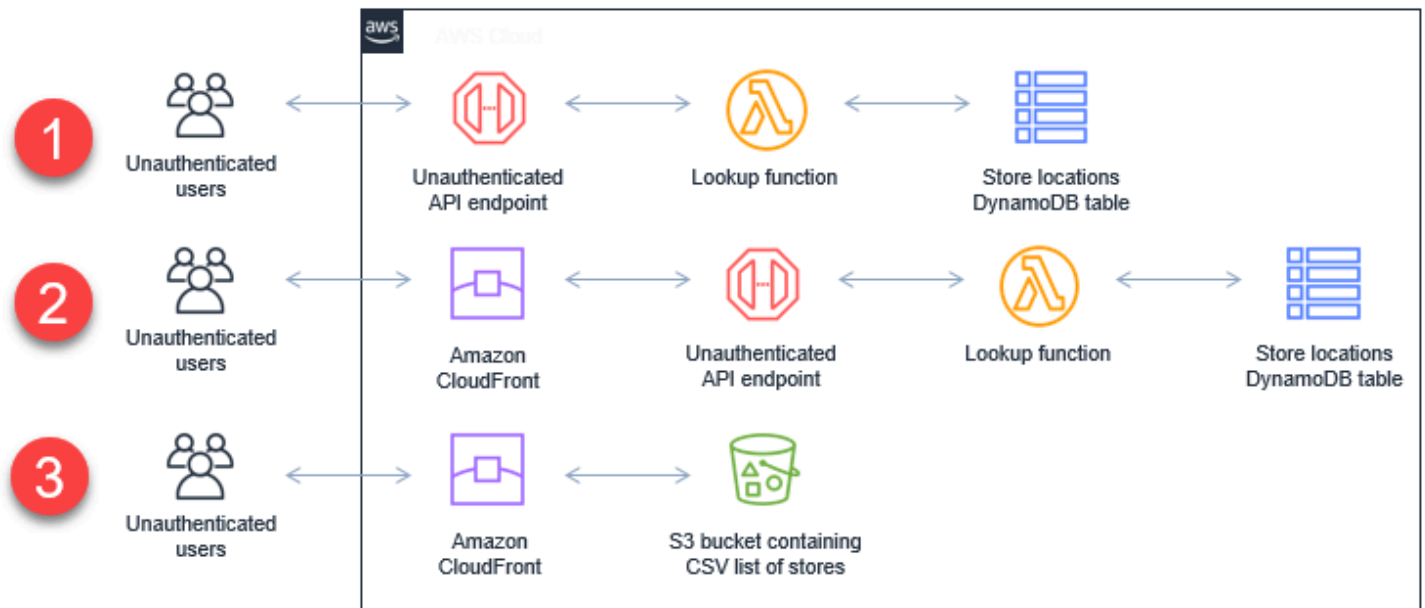
Per le applicazioni serverless, il modo preferito per servire pubblicamente un'applicazione di backend consiste nell'utilizzare Gateway Amazon API. Ciò può aiutarti a proteggere un'API da utenti malintenzionati o da picchi di traffico.

API Gateway offre due tipi di endpoint per gli sviluppatori serverless: [REST API](#) e [API HTTP](#). Entrambi supportano [l'autorizzazione tramite AWS Lambda IAM](#) o Amazon Cognito. Quando si utilizza IAM o Amazon Cognito, le richieste in entrata vengono valutate e se mancano di un token richiesto o contengono un'autenticazione non valida, la richiesta viene rifiutata. Queste richieste non ti vengono addebitate e non vengono conteggiate ai fini delle quote di limitazione.

Le route API non autenticate sono accessibili da chiunque sulla rete Internet pubblica, quindi è consigliabile limitare l'uso di API non autenticate. Se devi utilizzare API non autenticate, è importante proteggerle dai rischi comuni, come gli attacchi denial-of-service (DoS). [https://en.wikipedia.org/wiki/Denial-of-service\\_attack](https://en.wikipedia.org/wiki/Denial-of-service_attack) [L'applicazione di AWS WAF](#) a queste API può aiutare a proteggere l'applicazione da attacchi di iniezione SQL e cross-site scripting (XSS). API Gateway implementa anche la [limitazione](#) a AWS livello di account e per client quando vengono utilizzate le chiavi API.

In molti casi, la funzionalità fornita da un'API non autenticata può essere ottenuta con un approccio alternativo. Ad esempio, un'applicazione web potrebbe fornire un elenco dei negozi al dettaglio dei clienti da una tabella DynamoDB agli utenti che non hanno effettuato l'accesso. Questa richiesta potrebbe provenire da un'applicazione web frontend o da qualsiasi altra fonte che richiama l'endpoint URL. Questo diagramma mette a confronto tre soluzioni:





1. Questa API non autenticata può essere chiamata da chiunque su Internet. In un attacco denial of service, è possibile esaurire i limiti di limitazione delle API, la concorrenza Lambda o la capacità di lettura fornita da DynamoDB su una tabella sottostante.
2. Una CloudFront distribuzione davanti all'endpoint API con una configurazione [time-to-live](#) (TTL) appropriata assorbirebbe la maggior parte del traffico in un attacco DoS, senza modificare la soluzione sottostante per il recupero dei dati.
3. In alternativa, per i dati statici che cambiano raramente, la CloudFront distribuzione potrebbe fornire i «dati» contenuti in un bucket Amazon S3.

## Utilizzo della firma del codice per verificare l'integrità del codice con Lambda

La firma del codice consente di garantire che solo il codice attendibile venga eseguito nelle funzioni Lambda. Usando AWS Signer, puoi creare pacchetti di codice con firma digitale per le tue funzioni. Quando [aggiungi una configurazione di firma del codice a una funzione](#), Lambda verifica che tutte le nuove implementazioni di codice siano firmate da una fonte attendibile. Poiché i controlli di convalida della firma del codice vengono eseguiti al momento dell'implementazione, non vi è alcun impatto sull'esecuzione delle funzioni. Per informazioni sulla gestione delle politiche di firma del codice, vedere [the section called "Permissions"](#).

### Important

Le configurazioni di firma del codice impediscono solo nuove implementazioni di codice non firmato. Se aggiungi una configurazione di firma del codice a una funzione esistente con codice non firmato, tale codice continua a funzionare finché non distribuisce un nuovo pacchetto di codice.

Quando attivi la firma del codice per una funzione, tutti i [livelli](#) aggiunti alla funzione devono essere firmati anche da uno dei profili di firma consentiti.

Non sono previsti costi aggiuntivi per l'utilizzo AWS Signer o la firma del codice AWS Lambda.

## Convalida della firma

Lambda esegue i seguenti controlli di convalida quando si distribuisce un pacchetto di codice firmato alla funzione:

1. **Integrità:** convalida che il pacchetto di codice non sia stato modificato dopo la firma. Lambda confronta l'hash del pacchetto con l'hash della firma.
2. **Scadenza:** convalida che la firma del pacchetto di codice non sia scaduta.
3. **Mancata corrispondenza:** convalida che il pacchetto di codice sia firmato con un profilo di firma consentiti.
4. **Revoca:** convalida che la firma del pacchetto di codice non sia stata revocata.

Quando crei una configurazione di firma del codice, puoi utilizzare il [UntrustedArtifactOnDeployment](#) parametro per specificare come deve rispondere Lambda in caso di esito negativo dei controlli di scadenza, mancata corrispondenza o revoca. Puoi scegliere una di queste azioni:

- **Warn:** è l'impostazione predefinita. Lambda consente l'implementazione del pacchetto di codice, ma emette un avviso. Lambda emette una nuova CloudWatch metrica Amazon (`SignatureValidationErrors`) e memorizza anche l'avviso nel registro. CloudTrail
- **Enforce:** Lambda emette un avviso (lo stesso dell'azione Warn) e blocca l'implementazione del pacchetto di codice.

## Argomenti

- [Creazione di configurazioni di firma del codice per Lambda](#)
- [Configurazione delle policy IAM per le configurazioni di firma del codice Lambda](#)
- [Utilizzo di tag nelle configurazioni di firma del codice](#)

## Creazione di configurazioni di firma del codice per Lambda

Per abilitare la firma del codice per una funzione, creare una configurazione di firma del codice e collegarla alla funzione. Una configurazione di firma del codice definisce un elenco di profili di firma consentiti e le operazioni delle policy da eseguire in caso di esito negativo dei controlli di convalida.

### Note

Le funzioni definite come immagini di container non supportano la firma del codice.

## Sezioni

- [Prerequisiti di configurazione](#)
- [Creazione delle configurazioni di firma del codice](#)
- [Abilitazione della firma del codice per una funzione](#)

## Prerequisiti di configurazione

Prima di configurare la firma del codice per una funzione Lambda, utilizzare AWS Signer per effettuare le seguenti operazioni:

- Creare uno o più [profili di firma](#).
- Utilizzare un profilo di firma per [creare un pacchetto di codice firmato per la funzione](#).

## Creazione delle configurazioni di firma del codice

Una configurazione di firma del codice definisce un elenco di profili di firma consentiti e la policy di convalida della firma.

Per creare una configurazione di firma del codice (console)

1. Aprire la pagina [Configurazioni di firma del codice](#) della console Lambda.

2. Scegli Crea configurazione.
3. In Description (Descrizione), immettere un nome descrittivo per la configurazione.
4. In Signing profiles (Profili di firma) aggiungere fino a 20 profili di firma alla configurazione.
  - a. Per l'ARN della versione del profilo di firma, scegliere l'ARN (Amazon Resource Name) di una versione del profilo oppure inserire l'ARN.
  - b. Per aggiungere un profilo di firma aggiuntivo, scegliere Add signing profiles (Aggiungi profili di firma).
5. In Signature validation policy (Policy di convalida della firma), scegliere Warn (Avvisa) o Enforce (Applica).
6. Scegli Crea configurazione.

### Abilitazione della firma del codice per una funzione

Per abilitare la firma del codice per una funzione, aggiungi una configurazione di firma del codice alla funzione.

#### Important

Le configurazioni di firma del codice impediscono solo nuove implementazioni di codice non firmato. Se aggiungi una configurazione di firma del codice a una funzione esistente con codice non firmato, tale codice continua a funzionare finché non distribuisce un nuovo pacchetto di codice.

Per associare una configurazione di firma del codice a una funzione (console)

1. Aprire la pagina [Funzioni](#) della console Lambda.
2. Scegliere la funzione per la quale si desidera abilitare la firma del codice.
3. Apri la scheda Configurazione.
4. Scorri verso il basso e scegli Firma del codice.
5. Scegli Modifica.
6. In Edit code signing (Modifica della firma del codice), scegliere una configurazione di firma del codice per questa funzione.
7. Selezionare Salva.

## Configurazione delle policy IAM per le configurazioni di firma del codice Lambda

Per concedere l'autorizzazione a un utente di accedere alle operazioni dell'API di firma del codice Lambda, allegare una o più istruzioni delle policy alla policy dell'utente. Per ulteriori informazioni sulle policy utente, consulta [Identity-based Politiche IAM per Lambda](#).

La seguente istruzione di policy di esempio concede l'autorizzazione per creare, aggiornare e recuperare le configurazioni di firma del codice.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:CreateCodeSigningConfig",
        "lambda:UpdateCodeSigningConfig",
        "lambda:GetCodeSigningConfig"
      ],
      "Resource": "*"
    }
  ]
}
```

Gli amministratori possono utilizzare la chiave di condizione `CodeSigningConfigArn` per specificare le configurazioni di firma del codice che gli sviluppatori devono utilizzare per creare o aggiornare le funzioni.

La seguente istruzione di policy di esempio concede l'autorizzazione per creare una funzione. La dichiarazione di policy include una condizione `lambda:CodeSigningConfigArn` per specificare la configurazione di firma del codice consentita. Lambda blocca le richieste `CreateFunction` API se il [CodeSigningConfigArn](#) parametro manca o non corrisponde al valore nella condizione.

JSON

```
{
  "Version": "2012-10-17",
```

```
"Statement": [
  {
    "Sid": "AllowReferencingCodeSigningConfig",
    "Effect": "Allow",
    "Action": [
      "lambda:CreateFunction"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "lambda:CodeSigningConfigArn": "arn:aws:lambda:us-east-1:111122223333:code-signing-config:csc-0d4518bd353a0a7c6"
      }
    }
  }
]
```

## Utilizzo di tag nelle configurazioni di firma del codice

Puoi contrassegnare le configurazioni di firma del codice per organizzare e gestire le risorse. I tag sono coppie chiave-valore a forma libera associate alle risorse supportate su Servizi AWS. Per ulteriori informazioni sui casi d'uso dei tag, consulta Strategie di tagging [comuni](#) nella Guida alle AWS risorse sui tag e all'editor di tag.

Puoi utilizzare l' AWS Lambda API per visualizzare e aggiornare i tag. Puoi anche visualizzare e aggiornare i tag mentre gestisci una configurazione specifica di firma del codice nella console Lambda.

### Sections

- [Autorizzazioni necessarie per lavorare con i tag](#)
- [Utilizzo di tag con la console Lambda](#)
- [Utilizzo dei tag con AWS CLI](#)

### Autorizzazioni necessarie per lavorare con i tag

Per consentire a un'identità AWS Identity and Access Management (IAM) (utente, gruppo o ruolo) di leggere o impostare tag su una risorsa, concedile le autorizzazioni corrispondenti:

- `lambda: ListTags` —Quando una risorsa ha dei tag, concedi questa autorizzazione a chiunque debba richiamarla `ListTags`. Per le funzioni con tag, questa autorizzazione è necessaria anche per `GetFunction`.
- `lambda: TagResource` —Concedi questa autorizzazione a chiunque debba chiamare `TagResource` o eseguire un tag durante la creazione.

Facoltativamente, valuta la possibilità di concedere anche l'`UntagResource` autorizzazione `lambda:` per consentire `UntagResource` le chiamate alla risorsa.

Per ulteriori informazioni, consulta [Identity-based Politiche IAM per Lambda](#).

### Utilizzo di tag con la console Lambda

Puoi utilizzare la console Lambda per creare configurazioni di firma del codice che hanno tag, aggiungere tag a configurazioni di firma del codice esistenti e filtrare le configurazioni di firma del codice per tag.

Per aggiungere un tag durante la creazione di una configurazione di firma del codice

1. Apri la pagina [Configurazioni di firma del codice](#) della console Lambda.
2. Dall'intestazione del riquadro dei contenuti, scegli Crea configurazione.
3. Nella sezione nella parte superiore del riquadro dei contenuti, configura la configurazione di firma del codice. Per ulteriori informazioni sull'impostazione delle configurazioni di firma del codice, consulta [the section called “Firma del codice”](#).
4. Nella sezione Tag, seleziona Aggiungi nuovo tag.
5. Nel campo Chiave, inserisci la chiave del tag. Per informazioni sulle restrizioni relative ai tag, consulta Limiti e requisiti per la denominazione dei [tag](#) nella [AWS Tagging Resources and Tag Editor Guide](#).
6. Scegli Crea configurazione.

Per aggiungere un tag a una configurazione di firma del codice esistente

1. Apri la pagina [Configurazioni di firma del codice](#) della console Lambda.
2. Scegli il nome della configurazione di firma del codice.
3. Nelle schede sotto il riquadro Dettagli, scegli Tag.
4. Scegliere Gestisci tag.

5. Scegliere Aggiungi nuovo tag.
6. Nel campo Chiave, inserisci la chiave del tag. Per informazioni sulle restrizioni di etichettatura, consulta Limiti e requisiti per la denominazione dei [tag](#) nella AWS Tagging Resources and Tag Editor Guide.
7. Selezionare Salva.

Per filtrare le configurazioni di firma del codice per tag

1. Apri la pagina [Configurazioni di firma del codice](#) della console Lambda.
2. Scegli la barra di ricerca.
3. Dall'elenco a discesa, seleziona il tag sotto il sottotitolo Tag.
4. Seleziona Usa: "nome-tag" per vedere tutte le configurazioni di firma del codice etichettate con questa chiave, oppure scegli un operatore per filtrare ulteriormente in base al valore.
5. Seleziona il valore del tag da filtrare in base a una combinazione di chiave e valore del tag.

La barra di ricerca supporta anche la ricerca di chiavi di tag. Immetti il nome di una chiave per trovarla nell'elenco.

## Utilizzo dei tag con AWS CLI

Puoi aggiungere e rimuovere tag sulle risorse Lambda esistenti, incluse le configurazioni di firma del codice, con l'API Lambda. È inoltre possibile aggiungere tag durante la creazione di una configurazione di firma del codice per mantenere una risorsa contrassegnata durante l'intero ciclo di vita.

## Aggiornamento dei tag con le API dei tag Lambda

Puoi aggiungere e rimuovere tag per le risorse Lambda supportate tramite le operazioni [TagResource](#) e [UntagResource](#) API.

Puoi chiamare queste operazioni tramite la AWS CLI. Per aggiungere i tag a una risorsa esistente, utilizza il comando `tag-resource`. Questo esempio aggiunge due tag, uno con la chiave *Department* e uno con la chiave *CostCenter*.

```
aws lambda tag-resource \  
--resource arn:aws:lambda:us-east-2:123456789012:resource-type:my-resource \  
--tags Department=Marketing, CostCenter=1234ABCD
```



Per rimuovere i tag, utilizza il comando `untag-resource`. Questo esempio rimuove il tag con la chiave *Department*.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:resource-type:resource-identifier \
--tag-keys Department
```

Aggiunta di tag durante la creazione di una configurazione di firma del codice

Per creare una nuova configurazione di firma del codice Lambda con tag, utilizza l'operazione [CreateCodeSigningConfig](#) API. Specifica il parametro `Tags`. Puoi chiamare questa operazione con il `create-code-signing-config` AWS CLI comando e l' `--tags` opzione. Per ulteriori informazioni sul comando della CLI, consulta [create-code-signing-config](#) in Riferimento ai comandi della AWS CLI.

Prima di utilizzare il parametro `Tags` con `CreateCodeSigningConfig`, assicurati che il tuo ruolo disponga dell'autorizzazione per etichettare le risorse oltre alle normali autorizzazioni necessarie per questa operazione. Per ulteriori informazioni sulle autorizzazioni per il tagging, consulta [the section called "Autorizzazioni necessarie per lavorare con i tag"](#).

Visualizzazione dei tag con le API dei tag Lambda

Per visualizzare i tag applicati a una risorsa Lambda specifica, utilizza l'operazione API `ListTags`. Per ulteriori informazioni, consulta [ListTags](#).

Puoi chiamare questa operazione con il `list-tags` AWS CLI comando fornendo un ARN (Amazon Resource Name).

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:resource-type:resource-identifier
```

Filtro delle risorse per tag

Puoi utilizzare l'operazione AWS Resource Groups Tagging API [GetResources](#) API per filtrare le tue risorse per tag. L'operazione `GetResources` riceve fino a 10 filtri, ognuno dei quali contenente una chiave di tag e un massimo di 10 valori di tag. Fornisci `GetResources` con un `ResourceType` per filtrare in base a tipi di risorse specifiche.

Puoi chiamare questa operazione usando il `get-resources` AWS CLI comando. Per esempi di utilizzo di `get-resources`, consulta [get-resources](#) nella Riferimento ai comandi CLI di AWS.

# AWS Lambda MicroVM

Se stai creando un'applicazione in cui più utenti o agenti di intelligenza artificiale si connettono a un ambiente di elaborazione ed eseguono codice (ad esempio, ambienti di sviluppo interattivi, CI/CD sistemi o sandbox per l'IA), hai bisogno di ambienti di elaborazione che si avviino rapidamente, rimangano isolati tra i tenant e non richiedano la gestione di server o reti.

AWS Lambda Le MicroVM sono progettate appositamente per questi casi d'uso. Sono ambienti di elaborazione serverless che forniscono VM-level isolamento con funzionalità complete del sistema operativo (ad esempio, installazione di pacchetti di sistema o montaggio di file system), velocità di avvio rapide basate su istantanee e controllo granulare sulle reti in ingresso (accesso alle porte, supporto per HTTP/2, gRPC) e sulle reti in uscita (accesso pubblico a Internet e accesso VPC). WebSockets

Per i casi d'uso interattivi come gli ambienti di sviluppo interattivi e le sandbox di intelligenza artificiale, gli ambienti possono subire lunghi periodi di inattività quando il contesto degli utenti finali passa ad altre attività o quando attendono l'intelligenza artificiale. Le MicroVM possono essere sospese quando sono inattive, preservando la memoria e lo stato del disco e riducendo i costi. Quando il traffico arriva, viene ripreso. Puoi abilitare la sospensione automatica o programmatica della ripresa per ridurre i costi di inattività, mantenendo al contempo una disponibilità quasi istantanea al ritorno degli utenti finali.

AWS Lambda Le MicroVM offrono queste funzionalità di base tramite la virtualizzazione Firecracker, che alimenta gli oltre 15 trilioni di chiamate mensili di Lambda Functions.

Per iniziare, consulta [Crea la tua prima microVM](#). Per informazioni sui prezzi, consultare [Prezzi di AWS Lambda](#). Per sicurezza [the section called “Sicurezza”](#) e autorizzazioni, consulta. Per l'osservabilità, vedere. [the section called “Monitoraggio”](#) Per indicazioni operative, vedere [the section called “Best practice”](#).

## Come funziona Lambda MicroVMS

1. Inserisci il codice dell'applicazione e a Dockerfile in un archivio zip e lo carichi su Amazon S3.
2. Chiami l'API Lambda per creare un'immagine MicroVM. Lambda esegue l'utenteDockerfile, avvia l'applicazione e acquisisce un'istanza dell'ambiente completamente inizializzato.

3. Quando la tua applicazione ha bisogno di un ambiente isolato, per una sessione utente, un lavoro o una sandbox, chiami `run-microvm` Lambda avvia una MicroVM dall'istantanea con tempi di avvio rapidi.
4. I client si connettono alla MicroVM in esecuzione tramite il relativo endpoint HTTPS dedicato. Non sono necessari sistemi di bilanciamento del carico o infrastruttura di ingresso.
5. Quando è inattiva, la microVM si sospende, preservando la memoria e lo stato del disco e riducendo i costi. Riprende con la memoria e lo stato del disco intatti quando il traffico ritorna. Suspend-resume il comportamento può essere configurato automaticamente tramite politiche del ciclo di vita o attivato direttamente tramite chiamate API a `e. suspend-microvm resume-microvm`
6. Al termine della sessione, si chiude la MicroVM per rilasciare tutte le risorse.

## Quando usare Lambda MicroVMS

Lambda MicroVMS è ideale per i casi d'uso che eseguono utente o AI-generated codice e richiedono ambienti di esecuzione che offrano un forte isolamento, una latenza di avvio e ripresa rapida e il controllo degli sviluppatori sul ciclo di vita dell'ambiente e sulla conservazione dello stato. I casi d'uso tipici includono:

- Ambienti di codice interattivi: ambienti di sviluppo in cui gli utenti scrivono ed eseguono codice in tempo reale.
- Sandbox per l'esecuzione del codice AI: sandbox temporanei per l'esecuzione sicura del codice. AI-generated
- Applicazioni di analisi dei dati: notebook Jupyter e carichi di lavoro temporanei di elaborazione dati che eseguono script forniti dall'utente.
- Scansione della sicurezza: strumenti di valutazione delle vulnerabilità che richiedono ambienti di esecuzione isolati.
- Ambienti di reinforcement learning: sandbox isolate per la valutazione e la formazione degli agenti di intelligenza artificiale, in cui vengono creati nuovi ambienti per ogni esecuzione.
- Multi-tenant CI/CD— Esecutori di attività che richiedono l'isolamento tra gli inquilini.
- Server di gioco: ambienti di hosting che eseguono script forniti dall'utente con un forte isolamento.

## Funzionalità principali

- **Avvio rapido:** i MicroVMS riprendono le istantanee preinizializzate, saltando l'inizializzazione dell'applicazione. Per informazioni, consulta [Immagini MicroVM](#).
- **Controllo del ciclo di vita:** sospendi, riprendi e termina MicroVMS in modo programmatico o automatico tramite policy di inattività configurabili. Per informazioni, consulta [Esecuzione di MicroVMS](#).
- **Rete flessibile:** le microVMS supportano il traffico HTTPS in entrata su porte configurabili, con autenticazione JWE fornita dal servizio. L'accesso in uscita è configurabile. Accedi alla rete Internet pubblica o al tuo VPC. Per i dettagli, consulta il capitolo su [Rete](#).
- **Allocazione flessibile delle risorse:** assegna a ciascuna MicroVM un utilizzo di base o medio, con la flessibilità di scalare verticalmente fino a 4 volte la linea di base configurata durante i picchi di attività. Paga la tariffa di base mentre la tua MicroVM è in funzione e paga solo per un utilizzo attivo superiore alla linea di base.

## AWS Lambda Concetti fondamentali di MicroVMS

AWS Lambda MicroVMS utilizza diversi tipi di risorse che puoi creare e gestire. Questa pagina descrive ogni tipo di risorsa, come Lambda costruisce l'immagine MicroVM in un'istananea e gli stati del ciclo di vita che una microVM passa in fase di esecuzione, la base per la creazione di applicazioni con MicroVMS.

### Concetti chiave

#### MicroVM

Una microVM è una risorsa che rappresenta un ambiente di elaborazione isolato per un singolo tenant, sessione utente o processo. Ogni MicroVM esegue un sistema operativo Amazon Linux 2023 con funzionalità di sistema operativo, garantendo avvio e ripristino quasi istantanei. Le MicroVM ricevono richieste tramite connessioni HTTPS in entrata e possono essere sospese quando sono inattive, preservando la memoria e lo stato del disco. Una MicroVM sospesa riprende quando il traffico ritorna.

#### Immagine MicroVM

Un'immagine MicroVM è una risorsa che definisce l'ambiente applicativo di una MicroVM. Quando si crea un'immagine, Lambda la crea in un'istananea che consente un avvio quasi istantaneo (vedi sotto). [Come Lambda costruisce la tua immagine](#)

Per creare un'immagine MicroVM, fornisci un pacchetto zip contenente gli artefatti dell'applicazione, caricati su Dockerfile Amazon S3. È necessario utilizzare un'immagine di base Lambda-published gestita come base: specificala con il parametro `base-image-arn`. Il tuo Dockerfile definisce i livelli applicativi che Lambda crea sulla base gestita.

Le immagini MicroVM hanno una versione. Ogni versione rappresenta una singola build prodotta da un artefatto di codice e un'immagine di base specifici. Una versione procede attraverso gli stati di compilazione (PENDING → IN\_PROGRESS → SUCCESSFUL o FAILED) e le versioni riuscite possono essere impostate su o. ACTIVE INACTIVE Per informazioni dettagliate sullo stato e sulla gestione delle immagini, vedere [Immagini MicroVM](#).

## Connettori di rete

I connettori di rete sono risorse che controllano il modo in cui il traffico raggiunge la microVM e il modo in cui la microVM raggiunge i servizi esterni. È possibile associare i connettori a una MicroVM in fase di esecuzione per configurare l'accesso in entrata e in uscita in modo indipendente.

Build-time e i connettori in fase di esecuzione possono differire, consentendo alla microVM di raggiungere ambienti diversi durante la creazione dell'immagine rispetto al runtime.

Usa le Lambda-provided impostazioni predefinite per l'accesso alla porta in entrata (con autenticazione JWE), l'accesso alla shell e l'uscita pubblica da Internet. Crea il tuo connettore di rete per indirizzare il traffico in uscita attraverso il tuo VPC.

## Come Lambda costruisce la tua immagine

Quando crei o aggiorni un'immagine MicroVM, Lambda esegue un processo di compilazione che produce un'istantanea Firecracker. Questa istantanea cattura lo stato completamente inizializzato dell'applicazione, consentendo l'avvio e il ripristino quasi istantanei delle MicroVM che vengono eseguite da essa.

Il processo di compilazione:

1. Lambda esegue il provisioning di una nuova MicroVM utilizzando l'immagine di base gestita specificata.
2. Lambda esegue Dockerfile le istruzioni per installare le dipendenze e configurare l'ambiente.
3. Lambda avvia l'applicazione utilizzando il comando `or. ENTRYPOINT CMD`

4. Se hai abilitato `/readyhook`, Lambda attende che l'applicazione segnali di disponibilità (HTTP 200).
5. Lambda acquisisce un'istantanea dello stato del disco e della memoria, inclusi tutti i processi in esecuzione.

Quando si esegue una microVM, Lambda la ripristina da questa istantanea. L'applicazione riprende dallo stato preinizializzato senza ripetere l'avvio.

Se l'applicazione genera contenuti univoci durante la compilazione (ad esempio ID univoci, segreti o connessioni di rete), tali contenuti vengono condivisi tra tutte le MicroVM eseguite dalla stessa versione dell'immagine. Per evitare ciò, genera contenuti univoci dopo che la MicroVM inizia a utilizzare il lifecycle hook. `/run` Per i dettagli, consulta la sezione sulla compatibilità delle istantanee in [Immagini MicroVM](#)

## Ciclo di vita di MicroVM

In fase di esecuzione, una microVM passa attraverso le seguenti fasi:

1. Esegui: chiami. `run-microvm` Lambda ripristina la microVM dall'istantanea dell'immagine, assegna un ID univoco e crea un endpoint. La microVM passa PENDING da RUNNING a.
2. In esecuzione: l'applicazione riceve ed elabora le richieste tramite l'URL dell'endpoint.
3. Sospensione: dopo un periodo di inattività configurabile (o tramite `suspend-microvmAPI`), la microVM passa a. SUSPENDING SUSPENDED La memoria e lo stato del disco vengono preservati.
4. Riprendi: la microVM passa SUSPENDED direttamente a RUNNING quando arriva il traffico (`seautoResumeEnabled=true`) o chiami. `resume-microvm`
5. Termina: la microVM passa a TERMINATED quando si chiama `terminate-microvm` o viene superata TERMINATING la durata massima.

## Stati

La tabella seguente descrive ogni stato di MicroVM. Questi stati consentono di creare applicazioni affidabili e implementare una corretta gestione degli errori.

| Stato       | Description                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------------------|
| PENDING     | È in corso il provisioning di MicroVM. Le risorse vengono allocate e l'istanza è in fase di caricamento.                 |
| RUNNING     | MicroVM è attiva e accetta il traffico tramite l'URL dell'endpoint. L'/runhook è stato completato.                       |
| SUSPENDING  | MicroVM è in fase di sospensione. L'/suspendhook è in esecuzione. Il disco e la memoria vengono controllati.             |
| SUSPENDED   | MicroVM è sospeso. Lo stato è preservato. Non vengono addebitati costi di elaborazione. Può essere ripreso o interrotto. |
| TERMINATING | MicroVM è in fase di chiusura. L'/terminate hook è in esecuzione. Le risorse vengono rilasciate.                         |
| TERMINATED  | MicroVM è stato terminato. Si tratta di uno stato terminale. La MicroVM non può essere ripresa o riavviata.              |

## Transizioni di stato

La tabella seguente mostra le transizioni valide tra gli stati di MicroVM e ciò che attiva ciascuna transizione.

| Initial state (Stato iniziale) | Stato di destinazione | Trigger                                                                              |
|--------------------------------|-----------------------|--------------------------------------------------------------------------------------|
| PENDING                        | RUNNING               | Fornitura completata, /run hook riuscito.                                            |
| RUNNING                        | SUSPENDING            | Durata di inattività superata o chiamata API esplicita. <code>suspend-microvm</code> |
| SUSPENDING                     | SUSPENDED             | /suspendhook completato, memoria e stato del disco controllati.                      |

| Initial state (Stato iniziale) | Stato di destinazione | Trigger                                                                                                          |
|--------------------------------|-----------------------|------------------------------------------------------------------------------------------------------------------|
| SUSPENDED                      | RUNNING               | Traffico in arrivo ( <code>autoResumeEnabled=true</code> ) o chiamata <code>resume-microvm</code> API esplicita. |
| RUNNING                        | TERMINATING           | Chiamata <code>terminate-microvm</code> API esplicita o <code>maximumDurationInSeconds</code> superata.          |
| SUSPENDED                      | TERMINATING           | <code>suspendedDurationSeconds</code> chiamata API superata o esplicita <code>terminate-microvm</code> .         |
| TERMINATING                    | TERMINATED            | <code>/terminate</code> hook completato, tutte le risorse rilasciate.                                            |

#### Important

Se l'hook `/runhook` fallisce o scade, la microVM può passare direttamente a `TERMINATING` senza mai raggiungerla. `RUNNING` Implementa il timeout e la gestione degli errori negli hook per evitare guasti silenziosi.

## Crea la tua prima microVM Lambda

Questo tutorial illustra la creazione di un'immagine AWS Lambda MicroVM e l'esecuzione di una MicroVM da essa. Alla fine, avrai un'applicazione in esecuzione accessibile tramite HTTPS.

### Prerequisiti

Hai bisogno di due cose prima di iniziare:

1. Un bucket Amazon S3 nella tua AWS regione preferita per archiviare gli artefatti dell'applicazione (il file zip che creerai nella fase 2).



2. Un ruolo di build IAM che Lambda assume durante la creazione dell'immagine. Lambda utilizza questo ruolo per scaricare l'artefatto di codice da Amazon S3 e scrivere i log di compilazione in CloudWatch

Crea il ruolo IAM con la seguente policy di fiducia. Ciò consente al servizio Lambda di assumere il ruolo:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": { "Service": "lambda.amazonaws.com" },
    "Action": ["sts:AssumeRole", "sts:TagSession"]
  }]
}
```

Allega la seguente politica di autorizzazioni al ruolo. Sostituiscilo <your-bucket-name> con il nome del tuo bucket Amazon S3:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": "arn:aws:s3:::<your-bucket-name>/*"
    },
    {
      "Effect": "Allow",
      "Action": ["logs:CreateLogGroup", "logs:CreateLogStream", "logs:PutLogEvents"],
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

#### Note

Se esegui i Dockerfile prelievi da un repository AWS ECR privato, aggiungi anche `ecr:GetAuthorizationToken` and `ecr:BatchGetImage` alla policy delle autorizzazioni.

# Creazione della tua prima immagine MicroVM

Un'immagine MicroVM cattura l'applicazione in uno stato completamente inizializzato. Quando si esegue una microVM da questa immagine, l'applicazione viene avviata immediatamente, senza ritardi nell'avvio o nell'inizializzazione.

## Fase 1: Scrivere l'applicazione e il Dockerfile

Crea un semplice server HTTP che funzioni all'interno della tua MicroVM. Questo esempio utilizza Node.js senza dipendenze esterne:

### app.js

```
// Minimal HTTP server – listens on port 8080
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'application/json' });
  res.end(JSON.stringify({ status: 'ok', path: req.url }));
});

server.listen(8080, () => {
  console.log('Listening on port 8080');
});
```

Quindi, crea un Dockerfile pacchetto e avvia la tua applicazione:

### Dockerfile

```
# Use a lightweight Node.js runtime for your application layers
FROM node:24-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy your application code
COPY app.js .

# Declare the port your app listens on
EXPOSE 8080
```

```
# Start the application – Lambda snapshots the running state
CMD ["node", "app.js"]
```

### Note

L'FROMistruzione imposta l'immagine del contenitore per i livelli dell'applicazione. È possibile utilizzare qualsiasi immagine del contenitore compatibile. L'immagine di base gestita Lambda (che fornisce il sistema operativo MicroVM e i componenti di servizio) viene specificata separatamente --base-image-arn nel passaggio 3.

Se la tua applicazione genera valori univoci (ID, segreti o materiale crittografico), utilizza la libreria CSPRNG (Pseudorandom Number Generator) crittograficamente sicura (CSPRNG) standard del tuo linguaggio per garantire l'unicità tra le microVMS. Se la tua applicazione utilizza OpenSSL, usa l'immagine del contenitore di base Lambda che include una versione compatibile con le istantanee. Per i dettagli, consulta la sezione sulla compatibilità delle istantanee in [Immagini MicroVM](#)

## Passaggio 2: impacchettamento e caricamento su Amazon S3

Imballa app.js i tuoi dati Dockerfile in un archivio zip, quindi caricali nel tuo bucket Amazon S3. Esegui i seguenti comandi nel tuo terminale:

```
zip app.zip app.js Dockerfile
aws s3 cp app.zip s3://your-bucket-name/app.zip
```

## Passaggio 3: creare l'immagine MicroVM

Chiama create-microvm-image per iniziare la compilazione. Lambda scarica il file zip da Amazon S3, lo esegue Dockerfile, avvia l'applicazione e acquisisce un'istantanea Firecracker dello stato completamente inizializzato:

```
aws lambda-microvms create-microvm-image \
  --name my-first-microvm-image \
  --code-artifact uri=s3://your-bucket-name/app.zip \
  --base-image-arn arn:aws:lambda:us-east-1:aws:microvm-image:al2023-1 \
  --build-role-arn arn:aws:iam::123456789012:role/MicrovmBuildRole
```

L'immagine inizia nello stato. CREATING Controlla lo stato della build con:

```
aws lambda-microvms get-microvm-image \  
  --image-identifier my-first-microvm-image
```

Al termine della compilazione, il state campo diventa: **CREATED**

```
{  
  "imageName": "my-first-microvm-image",  
  "imageArn": "arn:aws:lambda:us-east-1:123456789012:microvm-image:my-first-microvm-image",  
  "state": "CREATED",  
  "imageVersion": "1.0",  
  ...  
}
```

Se lo stato è **CREATE\_FAILED**, controlla i log di compilazione CloudWatch sotto `/aws/lambda/microvms/my-first-microvm-image`.

## Esecuzione della prima MicroVM

Una volta raggiunto **CREATED** lo stato dell'immagine MicroVM, è possibile eseguire MicroVMS da essa. Ogni immagine può avviare più microVM, una per tenant, sessione utente o processo.

Esegui una microVM con il seguente comando:

```
aws lambda-microvms run-microvm \  
  --image-identifier my-first-microvm-image \  
  --ingress-network-connectors "arn:aws:lambda:us-east-1:aws:network-connector:aws-network-connector:ALL_INGRESS" \  
  --egress-network-connectors "arn:aws:lambda:us-east-1:aws:network-connector:aws-network-connector:INTERNET_EGRESS" \  
  --idle-policy  
  '{"autoResumeEnabled":true,"maxIdleDurationSeconds":900,"suspendedDurationSeconds":300}'
```

Parametri spiegati:

- `--ingress-network-connectors`— Abilita il traffico HTTPS in entrata verso la tua MicroVM su tutte le porte. Si tratta di un connettore. Lambda-managed
- `--egress-network-connectors`— Consente l'accesso a Internet in uscita dalla tua MicroVM. Si tratta di un connettore. Lambda-managed

- `--idle-policy`— Configura il comportamento automatico di sospensione-ripresa. Questa policy sospende la MicroVM dopo 15 minuti di inattività, la mantiene sospesa per un massimo di 5 minuti e riprende automaticamente quando arriva il traffico.

La risposta include l'ID MicroVM e l'URL dell'endpoint:

```
{
  "microvmId": "mvm-01234567-abcd-ef01-2345-6789abcdef01",
  "state": "PENDING",
  "endpoint": "mvm-01234567-abcd-ef01-2345-6789abcdef01.lambda-microvm.us-east-1.on.aws",
  ...
}
```

Attendi che lo stato raggiunga: `RUNNING`

```
aws lambda-microvms get-microvm \
  --microvm-identifier mvm-01234567-abcd-ef01-2345-6789abcdef01
```

## Connessione alla tua MicroVM

Tutte le richieste a un endpoint MicroVM richiedono un token di autenticazione. Generane uno con:

```
aws lambda-microvms create-microvm-auth-token \
  --microvm-identifier mvm-01234567-abcd-ef01-2345-6789abcdef01 \
  --expiration-in-minutes 30 \
  --allowed-ports '[{"allPorts":{}}]'
```

La risposta include un token nel `authToken` campo. Usalo per inviare una richiesta all'applicazione in esecuzione:

```
curl https://mvm-01234567-abcd-ef01-2345-6789abcdef01.lambda-microvm.us-east-1.on.aws/ \
  -H "X-aws-proxy-auth: <token-value>"
```

Dovresti vedere la risposta della tua applicazione:

```
{"status":"ok","path":"/"}
```

La tua MicroVM è in esecuzione e serve traffico. L'applicazione che hai scritto nel passaggio 1 è disponibile nell'URL dell'endpoint.

## Eliminazione

Per evitare addebiti continui, interrompi la MicroVM quando hai finito:

```
aws lambda-microvms terminate-microvm \  
  --microvm-identifier mvm-01234567-abcd-ef01-2345-6789abcdef01
```

## Fasi successive

- Scopri come [Concetti principali](#) comprendere il processo di istantanea, gli stati del ciclo di vita e le immagini di base.
- Scopri [Immagini MicroVM](#) di più sugli hook per la creazione delle immagini, sul controllo delle versioni e sulla compatibilità delle istantanee.
- Consulta [Esecuzione di MicroVMS](#) gli esempi di SDK, gli hook del ciclo di vita e le strategie di scalabilità.
- Consulta le integrazioni di servizi supportate [the section called “Integrazioni”](#).

## Immagini MicroVM

Questa sezione descrive come creare, configurare, aggiornare e gestire le immagini MicroVM.

Un'immagine MicroVM è una risorsa che definisce il file system e l'ambiente applicativo di una MicroVM. L'immagine MicroVM include l'ambiente di runtime, il codice dell'applicazione e i programmi di supporto come i processi in background e gli agenti di osservabilità. Per creare un'immagine MicroVM, fornisci un pacchetto zip contenente gli artefatti dell'applicazione, che caricherai su Amazon S3. `Dockerfile`

Tu `Dockerfile` definisci come viene impacchettata la tua applicazione. Lambda crea l'immagine del contenitore dell'applicazione eseguendola `Dockerfile` su un ambiente del sistema operativo fornito da un'immagine di base Lambda-managed MicroVM. Le immagini di base MicroVM sono descritte di seguito, nella sezione intitolata —. [Immagini base MicroVM](#)

È possibile aggiornare le immagini di base MicroVM per aggiornare il codice dell'applicazione o la configurazione del MicroVMS. Ogni aggiornamento attivato crea una nuova versione dell'immagine MicroVM.

## Come Lambda crea un'immagine MicroVM

Quando si crea un'immagine MicroVM, Lambda:

- Recupera gli artefatti confezionati da Amazon S3.
- Avvia una nuova MicroVM dall'immagine di base. Lambda-managed
- Esegue le istruzioni nel tuo. `Dockerfile`
- Avvia l'applicazione utilizzando l'istruzione `ENTRYPOINT` o `CMD`.
- Attende il completamento dell'inizializzazione, segnalato dall'hook del ciclo di vita.
- Acquisisce un'istantanea del disco e dello stato della memoria.

Una volta completato il processo di istantanea, l'immagine MicroVM entra nello stato. `CREATED` È ora possibile utilizzare questa immagine MicroVM per creare una MicroVM e ogni immagine MicroVM può essere utilizzata per creare più MicroVM indipendenti. Una microVM eseguita dall'immagine MicroVM riprende direttamente dallo stato istantaneo, garantendo tempi di avvio rapidi. Ogni immagine MicroVM può essere utilizzata per eseguire più MicroVM, fino al limite disponibile per il tuo account.

Per una procedura dettagliata sulla creazione del codice e sulla creazione della prima immagine MicroVM, consulta. [Crea la tua prima microVM](#)

## Dimensionamento delle microVM

Lambda MicroVMS utilizza un modello baseline-peak che elimina la necessità di dimensionare correttamente ogni ambiente di elaborazione per i picchi di attività. Configura le risorse di elaborazione di base per la tua MicroVM. Durante i picchi di attività, la tua microVM può scalare automaticamente verticalmente fino a 4 volte la linea di base. Paghi la tariffa di base mentre la tua MicroVM è in funzione e paghi solo ciò che utilizzi attivamente al di sopra della linea di base, fatturata al secondo.

La linea di base viene impostata utilizzando il `memory` parametro durante la creazione dell'immagine microVM. La vCPU si ridimensiona proporzionalmente alla memoria (2 GB = 1 vCPU). La linea di base predefinita è 2 GB/1 vCPU.

La tabella seguente elenca le dimensioni disponibili:

| Valore di base                                        | Picco                     | Spazio massimo su disco |
|-------------------------------------------------------|---------------------------|-------------------------|
| 0,5 GB di memoria, 0,25 vCPU                          | 2 GB di memoria, 1 vCPU   | 8 GB                    |
| 1 GB di memoria, 0,5 vCPU                             | 4 GB di memoria, 2 vCPU   | 8 GB                    |
| 2 GB di memoria, 1 vCPU<br>(impostazione predefinita) | 8 GB di memoria, 4 vCPU   | 8 GB                    |
| 4 GB di memoria, 2 vCPU                               | 16 GB di memoria, 8 vCPU  | 16 GB                   |
| 8 GB di memoria, 4 vCPU                               | 32 GB di memoria, 16 vCPU | 32 GB                   |

## Immagini base MicroVM

Un'immagine di base MicroVM funge da base per le immagini MicroVM. Lambda pubblica un'immagine di base MicroVM che fornisce il sistema operativo Amazon Linux 2023 e i componenti di servizio necessari per eseguire MicroVMS. Quando crei o aggiorni un'immagine MicroVM, Lambda avvia una nuova MicroVM da questa immagine di base ed esegue le istruzioni all'interno di questo ambiente del sistema operativo. `Dockerfile`

Lambda rilascia periodicamente nuove versioni delle immagini di base MicroVM gestite dai servizi, ad esempio quando si applicano le patch di sicurezza, per aggiornare il sistema operativo o i componenti del servizio. Per impostazione predefinita, la versione più recente di un'immagine di base gestita dal servizio si applica quando si utilizzano le proprie immagini MicroVM. `creating/updating` Per la risoluzione dei problemi o il debug, puoi opzionalmente sovrascrivere la versione delle immagini basate su service managed quando crei la tua immagine MicroVM utilizzando il parametro. `base-image-version`

Le versioni delle immagini di base seguono un ciclo di vita obsoleto:

- **AVAILABLE**— Attuale, consigliato per l'uso.
- **DEPRECATED**(60 giorni): esiste una versione più recente. Puoi ancora compilare ed eseguire.
- **EXPIRING**(30 giorni): impossibile creare nuove immagini. Le immagini esistenti possono ancora essere eseguite.
- **EXPIRED**— Impossibile creare o eseguire. Ricostruisci l'immagine su una versione supportata.



- **RECALLED**— Immediatamente non disponibile a causa di problemi di sicurezza critici (rari).

Per rimanere aggiornato, monitora le notifiche di obsolescenza e ricostruisci le tue immagini MicroVM quando viene rilasciata una nuova versione dell'immagine di base.

Nota che l'immagine di base di MicroVM è distinta dall'immagine di base del contenitore che specifichi nei tuoi Dockerfile. Mentre la prima definisce l'ambiente del sistema operativo per la microVMS, la seconda definisce l'immagine del contenitore di base da utilizzare per impacchettare l'applicazione da utilizzare con Lambda MicroVMS. Per ulteriori informazioni, consulta la sezione su [Immagini di base del contenitore](#).

Utilizza le seguenti API per scoprire le immagini di base MicroVM gestite disponibili e le relative versioni:

```
# List all managed MicroVM base images
aws lambda-microvms list-managed-microvm-images

# List the versions of a specific managed MicroVM base image
aws lambda-microvms list-managed-microvm-image-versions \
  --image-identifier arn:aws:lambda:us-east-1:aws:microvm-image:al2023-1
```

## ganci per la creazione di immagini MicroVM

Lambda fornisce hook per la creazione di immagini MicroVM che consentono di verificare la correttezza delle applicazioni e ottimizzare le prestazioni durante la creazione di immagini MicroVM. Gli hook vengono eseguiti prima che Lambda acquisisca l'istantanea utilizzata per inizializzare ogni MicroVM. Ogni hook è un endpoint HTTP che l'applicazione espone e che Lambda chiama durante la compilazione. Rispondendo a queste richieste, controlli e convalidi il processo di creazione dell'immagine MicroVM. Lambda utilizza i codici di stato HTTP per determinare se gli hook sono stati completati correttamente.

### Important

Se configuri degli hook, devi specificare la porta su cui l'applicazione è in ascolto per le richieste di hook.

| Hook       | Path                                    | Informazioni                                                                                                                                                                             | Codici di stato HTTP                                                                                                            | Timeout                                                  |
|------------|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| /pronto    | /aws/lambda-microvm/runtime/v1/ready    | Richiamato durante la creazione dell'immagine MicroVM, dopo l'avvio dell'applicazione tramite <code>ENTRYPOINT CMD</code> . Segnala che l'applicazione è pronta per essere scattata.     | HTTP 503: non ancora pronto; Lambda riprova fino al timeout. HTTP 200: inizializzazione completata; Lambda scatta l'istantanea. | 1—3600 secondi ( <code>readyTimeoutInSeconds</code> )    |
| /convalida | /aws/lambda-microvm/runtime/v1/validate | Richiamato dopo il completamento della compilazione, su una nuova MicroVM è stata avviata dall'immagine creata. Conferma che l'applicazione funziona correttamente quando viene ripresa. | HTTP 503: la convalida richiede più tempo per essere completata; Lambda riprova fino al timeout. HTTP 200: convalida superata.  | 1—3600 secondi ( <code>validateTimeoutInSeconds</code> ) |

### Important

Quando restituisci HTTP 503, restituiscilo immediatamente anziché tenere aperta la richiesta mentre aspetti. Se il timeout scade mentre una richiesta è mantenuta aperta, Lambda termina la compilazione.

### Note

Puoi anche utilizzare l'hook `/validate` per ottimizzare i tempi di avvio. Per farlo, esegui dei payload fittizi durante la convalida. Ciò consente a Lambda di tracciare le aree a cui si accede dello snapshot e ottimizzarne il recupero durante l'avvio di MicroVM.

## Aggiornamento di un'immagine MicroVM

È possibile aggiornare un'immagine MicroVM esistente chiamando l'API. `update-microvm-image`. Ogni aggiornamento attiva una nuova versione dell'immagine MicroVM. In genere si aggiorna un'immagine MicroVM per:

- Implementa un nuovo codice applicativo: indica un nuovo elemento di codice (un nuovo file zip caricato su Amazon S3) per inviare una nuova versione dell'applicazione.
- Passa a un'immagine base MicroVM più recente: modifica l'ARN dell'immagine di base MicroVM per eseguire l'upgrade alle versioni più recenti dell'immagine base Lambda MicroVM. Per ulteriori informazioni, consulta [Applicazione di patch alle immagini MicroVM](#) e [Immagini base MicroVM](#).
- Modifica del ruolo di compilazione: aggiorna l'ARN del ruolo di compilazione quando vengono modificate le autorizzazioni necessarie a Lambda durante la compilazione, ad esempio quando l'artefatto del codice si sposta in un altro bucket Amazon S3 o inizi a eseguire l'estrazione da un repository ECR privato.
- Modifica la configurazione del runtime: modifica gli hook, le variabili di ambiente o le funzionalità per riconfigurare il modo in cui l'immagine MicroVM viene creata ed eseguita.
- Aggiorna la descrizione: modifica la descrizione dell'immagine MicroVM per registrare cosa è cambiato in questa versione.

Il seguente comando CLI mostra come aggiornare un'immagine MicroVM. I `--build-role-arn` parametri `--base-image-arn` and sono obbligatori per ogni `update-microvm-image` chiamata che attiva una nuova build, anche quando si modifica solo l'artefatto del codice: la loro omissione comporta: `ValidationException`

```
aws lambda-microvms update-microvm-image \
  --image-identifier arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-
image \
  --code-artifact uri=s3://my-bucket/deployments/app-v2.zip \
  --base-image-arn arn:aws:lambda:us-east-1:aws:microvm-image:al2023-1 \
  --build-role-arn arn:aws:iam::123456789012:role/MicrovmBuildRole \
  --description "Updated with v2 application code"
```

## Stati dell'immagine e stati di costruzione

Ogni volta che crei o aggiorni un'immagine MicroVM, Lambda produce una nuova versione creata a partire dall'artefatto del codice e dall'immagine di base. Un'immagine MicroVM può avere molte versioni nel tempo ed è possibile eseguire MicroVMS a partire da una versione specifica.

Tre stati indipendenti tengono traccia di diversi aspetti del ciclo di vita:

- **Stato dell'immagine:** il ciclo di vita complessivo della risorsa di immagine MicroVM (in fase di creazione, pronta per l'uso, in fase di aggiornamento, errore o eliminazione).
- **Stato della versione:** l'avanzamento della compilazione di una versione specifica (in sospeso, in fase di compilazione, completata o non riuscita). Controlla `stateReason` o CloudWatch logs (`/aws/lambda/microvms/<image-name>`) per i dettagli sugli errori.
- **Attivazione della versione:** se una versione creata correttamente è autorizzata a eseguire MicroVMS. Lambda imposta `ACTIVE` automaticamente le nuove versioni; puoi impostare una versione su `INACTIVE` per disabilitarla senza eliminarla.

| Stato                      | Valori possibili                                                                                                 | Transizione controllata da                               |
|----------------------------|------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| Stato dell'immagine        | CREATING, CREATED, CREATION_FAILED ,<br>UPDATING, UPDATED, UPDATE_FAILED ,<br>DELETING, DELETED, DELETION_FAILED | Lambda (automatico)                                      |
| Stato della versione       | PENDING, IN_PROGRESS , SUCCESSFUL ,<br>FAILED                                                                    | Lambda (automatico)                                      |
| Attivazione della versione | ACTIVE, INACTIVE                                                                                                 | Tu ( <code>update-microvm-image-version --state</code> ) |

Per eseguire una MicroVM da una versione, lo stato dell'immagine deve essere `CREATED` o `UPDATED`, lo stato della versione deve essere `SUCCESSFUL` e la versione deve essere `ACTIVE`.

### Note

Questi stati sono indipendenti. Un'immagine nello `CREATED` stato può contenere una versione il cui stato è `FAILED`.

```
# De-activate a version
aws lambda-microvms update-microvm-image-version \
  --image-identifier my-image \
  --image-version 1.0 \
  --state INACTIVE
```

## Variabili di ambiente

Le variabili di ambiente sono impostate al momento della creazione dell'immagine MicroVM tramite il `environmentVariables` campo (massimo 50 variabili). Queste vengono inserite nel contenitore durante il processo di creazione dell'istanza. È possibile trasferire payload impostati dinamicamente quando si esegue una nuova MicroVM. Per ulteriori informazioni, consulta la sezione sull'esecuzione della MicroVM.

## Utilizzo degli snapshot

Quando AWS Lambda MicroVMS crea un'immagine MicroVM, acquisisce un'istanza Firecracker dell'applicazione completamente inizializzata. Questa pagina descrive cosa contiene un'istanza, come gestire l'unicità tra le microVM che condividono la stessa istanza e le best practice per le applicazioni compatibili con le istanze.

## Cosa cattura un'istanza

Un'istanza di Firecracker acquisisce il seguente stato dalla microVM nel momento in cui viene completata la creazione dell'immagine:

- Stato della memoria dei processi in esecuzione: tutti i processi avviati da `ENTRYPOINT/CMD`, inclusi i demoni in background, i cron job e i processi secondari.
- Stato del disco: il filesystem principale che include tutti i file scritti durante la fase di compilazione (pacchetti installati, file binari compilati, file di configurazione).
- Connessioni di rete e descrittori di file: eventuali connessioni di rete iniziate, handle di file, pipe e altri descrittori.

Quando si esegue una microVM, Lambda la ripristina da questa istanza. L'applicazione riprende esattamente da dove era stata interrotta la creazione della versione dell'immagine di MicroVM.

## Considerazioni sulla compatibilità

Poiché tutte le MicroVM avviate dalla stessa versione di immagine condividono lo stesso stato iniziale, considera quanto segue quando crei applicazioni con Lambda MicroVMS:

- **Unicità:** se il codice genera contenuti univoci durante la fase di creazione della versione dell'immagine (ID univoci, segreti o entropia per pseudorandomness), tale contenuto viene condiviso tra tutte le MicroVM eseguite dalla stessa versione dell'immagine. Genera contenuti unici dopo l'avvio di una nuova MicroVM, non durante la creazione dell'immagine. Usa l'hook del `/run` ciclo di vita per reimpostare i contenuti unici generati in precedenza.
- **Connessioni di rete:** le connessioni stabilite durante la fase di creazione della versione dell'immagine potrebbero dover essere ristabilite quando una MicroVM viene eseguita dalla versione dell'immagine. Convalida lo stato della connessione e ristabilisci le connessioni se necessario. Nella maggior parte dei casi, le connessioni create dagli AWS SDK vengono ristabilite automaticamente.

## Gestire l'unicità con le istantanee

Tutte le MicroVM eseguite dalla stessa immagine condividono lo stesso stato iniziale. Per mantenere l'unicità tra le MicroVMS:

- Genera ID univoci, segreti e valori casuali dopo l'avvio di una microVM e non durante la creazione della versione dell'immagine. Usa l'hook del `/run` ciclo di vita per reimpostare qualsiasi stato univoco.
- Usa generatori di numeri pseudocasuali (CSPRNG) crittograficamente sicuri per il tuo linguaggio di programmazione. Verifica che la libreria crittografica standard del tuo linguaggio riceva l'entropia dal dispositivo o dal dispositivo. `/dev/random` `/dev/urandom` Questo è il comportamento predefinito per le librerie crittografiche per i linguaggi di programmazione più diffusi come Java 11+ (SecureRandom libreria), (libreria Crypto.randomBytes), Node.js Python 3.12+ (libreria) e Dotnet8+ (libreria). `Secrets.SystemRandom` `Cryptography.RandomNumberGenerator` Software che legge o mantiene la casualità se utilizzato con istantanee di immagini MicroVM. `/dev/random` `/dev/urandom`
- Se il codice dell'applicazione utilizza OpenSSL, utilizza l'immagine del contenitore di base AWS fornita da per creare l'immagine MicroVM. `public.ecr.aws/lambda/microvms:a12023-minimal` Questa immagine contiene una versione aggiornata di AWS OpenSSL compatibile con lo snapshotting. Se desideri utilizzare la tua immagine di base con OpenSSL, utilizza la versione

patchata di OpenSSL (openssl-snap-safe-libs) elencata nell'elenco dei pacchetti di Amazon Linux 2023. <https://docs.aws.amazon.com/linux/al2023/release-notes/all-packages.html>

## Applicazione di patch alle immagini MicroVM

Quando è disponibile una nuova immagine di base MicroVM, puoi inviare una `update-microvm-image` chiamata per attivare una build dell'immagine MicroVM con le patch più recenti, omettendo l'`base-image-version` argomento (per l'ultima) o specificando l'argomento con la versione più recente.

## Immagini di base del contenitore

Lambda MicroVMS esegue l'applicazione come contenitore all'interno dell'ambiente del sistema operativo MicroVM. Definisci quel contenitore con la tua e l'FROMistruzione in `Dockerfile` essa contenuta `Dockerfile` imposta l'immagine di base del contenitore per la tua applicazione.

Puoi iniziare con l'immagine del contenitore di base Lambda per Amazon Linux 2023 (`public.ecr.aws/lambda/microvms:al2023-minimal`) e aggiungere `Dockerfile` le istruzioni sopra, oppure utilizzare la tua immagine del contenitore di base. Quando usi le tue immagini del contenitore, convalida i seguenti requisiti:

### Requisiti

- L'immagine di base del contenitore deve essere compatibile con l'architettura della CPU di destinazione.
- Le immagini di base del contenitore provenienti da repository AWS ECR privati richiedono il ruolo di compilazione `ecr:GetAuthorizationToken` e `ecr:BatchGetImage` le autorizzazioni.
- L'immagine di base del contenitore deve essere basata su un sistema operativo Linux.
- Le immagini di base del contenitore devono essere accessibili dall'infrastruttura di compilazione Lambda (Internet pubblico o un repository ECR nello stesso AWS account).
- Le immagini di base del contenitore devono essere compatibili con le istantanee, vedi le istruzioni riportate di seguito.

## Snapshot-compatible immagini di base

Poiché Lambda MicroVMS avvia ogni microVM da un'istantanea preinizializzata, le immagini di base devono essere compatibili con le istantanee. Ti consigliamo di consultare la sezione dedicata all'utilizzo delle tue immagini di base con Lambda [Considerazioni sulla compatibilità](#) MicroVMS.

## Utilizzo di un'immagine ECR privata

Fai riferimento all'immagine di base del tuo contenitore ECR privato nelle FROM istruzioni di: `Dockerfile`

```
FROM 123456789012.dkr.ecr.us-east-1.amazonaws.com/my-base:latest
WORKDIR /app
COPY . .
CMD ["/my-app"]
```

Aggiungi le seguenti autorizzazioni al tuo ruolo di compilazione:

```
{
  "Effect": "Allow",
  "Action": [
    "ecr:GetAuthorizationToken",
    "ecr:BatchCheckLayerAvailability",
    "ecr:GetDownloadUrlForLayer",
    "ecr:BatchGetImage"
  ],
  "Resource": "*"
}
```

## Funzionalità del sistema operativo

Per impostazione predefinita, Lambda MicroVMS viene eseguito con un set standard di funzionalità Linux. È possibile concedere funzionalità Linux elevate utilizzando il `additionalOsCapabilities` campo durante la creazione o l'aggiornamento di un'immagine MicroVM. L'unico valore supportato è `["ALL"]`. Le funzionalità elevate consentono operazioni come il montaggio di file system, la creazione di namespace di rete o l'esecuzione di programmi eBPF. Le funzionalità vengono applicate entro il limite di isolamento delle VM e non influiscono sull'host o su altre microVM.

```
aws lambda-microvms create-microvm-image \
  --name my-network-tool \
```



```
--code-artifact uri=s3://my-bucket/app.zip \  
--base-image-arn arn:aws:lambda:us-east-1:aws:microvm-image:al2023-1 \  
--build-role-arn arn:aws:iam::123456789012:role/BuildRole \  
--additional-os-capabilities '["ALL"]'
```

## Esecuzione e utilizzo di MicroVMS

Questa sezione descrive come avviare MicroVMS, connettersi alle applicazioni in esecuzione, gestire il ciclo di vita di MicroVM e gestire la scalabilità.

### Avvio di una MicroVM

Usa il `run-microvm` comando per avviare una nuova MicroVM da un'immagine specificata. Lambda fornisce le risorse richieste, crea un endpoint HTTPS dedicato e avvia l'applicazione dall'istantanea dell'immagine.

```
aws lambda-microvms run-microvm \  
  --image-identifier arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-  
image \  
  --ingress-network-connectors "arn:aws:lambda:us-east-1:aws:network-connector:aws-  
network-connector:ALL_INGRESS" \  
  --egress-network-connectors "arn:aws:lambda:us-east-1:aws:network-connector:aws-  
network-connector:INTERNET_EGRESS" \  
  --idle-policy  
'{"autoResumeEnabled":true,"maxIdleDurationSeconds":900,"suspendedDurationSeconds":1800}' \  
  --maximum-duration-in-seconds 14400
```

Quando si chiama, viene creata una MicroVM. `run-microvm` Ogni MicroVM ha il proprio endpoint dedicato. Non esiste un bilanciamento del carico tra le microVM da un singolo endpoint: ogni endpoint è collegato a una singola microVM.

L'unico parametro richiesto è `--image-identifier` (che deve essere l'ARN dell'immagine MicroVM). Tutti gli altri parametri sono facoltativi.

### Parametri chiave

| Parametro                       | Description                                             |
|---------------------------------|---------------------------------------------------------|
| <code>--image-identifier</code> | (Obbligatorio) L'ARN dell'immagine MicroVM da eseguire. |

| Parametro                                  | Description                                                                                                                                      |
|--------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>--image-version</code>               | La versione dell'immagine MicroVM da eseguire. Il valore predefinito è la versione attiva più recente.                                           |
| <code>--execution-role-arn</code>          | Il ruolo IAM che fornisce le autorizzazioni di runtime per consentire alla microVM di interagire con altri servizi. AWS                          |
| <code>--idle-policy</code>                 | Controlla il comportamento di sospensione e ripresa automatica. Vedi la configurazione dei criteri di inattività nella sezione seguente.         |
| <code>--maximum-duration-in-seconds</code> | La durata massima in cui la microVM può rimanere in esecuzione e o sospesa prima che Lambda la interrompa. Intervallo: 1—28.800 secondi (8 ore). |
| <code>--run-hook-payload</code>            | Un payload di stringa (max 16 KB) inviato all'hook del <code>/run</code> ciclo di vita all'avvio della MicroVM.                                  |
| <code>--logging</code>                     | Configurazione della registrazione. Personalizza il gruppo e lo stream di CloudWatch log o disabilita completamente la registrazione.            |
| <code>--ingress-network-connectors</code>  | Gli ARN dei connettori di ingresso che abilitano la connettività HTTPS in entrata.                                                               |
| <code>--egress-network-connectors</code>   | Gli ARN dei connettori di uscita per la connettività in uscita (Internet o VPC).                                                                 |

 Note

Per disattivare la connettività in ingresso, usa il connettore. `Lambda-provided NO_INGRESS`  
Per ulteriori dettagli sui connettori di rete, vedere [the section called “Rete”](#).

## Configurazione dei criteri di inattività

Se abilitata, la politica di inattività controlla la sospensione e la ripresa automatiche. La presenza di traffico attraverso l'endpoint della MicroVM segnala l'attività. Se non arriva traffico per la durata di inattività configurata, la microVM viene considerata inattiva e sospesa.

| Campo                                 | Description                                                                                                             |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>autoResumeEnabled</code>        | Quando <code>true</code> , la MicroVM riprende automaticamente quando il traffico arriva all'endpoint mentre è sospeso. |
| <code>maxIdleDurationSeconds</code>   | Il numero di secondi senza traffico trascorsi i quali la MicroVM viene sospesa. Massimo: 28.800 (8 ore).                |
| <code>suspendedDurationSeconds</code> | Il numero di secondi in cui una microVM rimane sospesa prima che Lambda la interrompa.                                  |

### Note

Per le applicazioni asincrone che non inviano o ricevono attivamente traffico attraverso l'endpoint, disabilita la sospensione automatica o configura una durata di inattività adeguata.

## Payload in fase di esecuzione

Con il `runHookPayload` parametro, è possibile passare i dati di configurazione per microVM (stringa massima di 16 KB) in fase di esecuzione. Lambda invia questo payload come parte del corpo della richiesta al lifecycle hook. `/run` Lambda lo inserisce anche nel corpo della richiesta. `microvmId`

L' `/runhook` riceve un corpo JSON con la seguente struttura:

```
{
  "microvmId": "mvm-01234567-abcd-ef01-2345-6789abcdef01",
  "runHookPayload": "tenant-specific-string"
}
```

Utilizza i payload di runtime per fornire una configurazione che varia a seconda della microVM, ad esempio ID tenant, token di sessione, URL firmati o percorsi di Secrets Manager. A differenza

delle variabili di ambiente (che sono impostate a livello di immagine e condivise tra tutte le microVM presenti nell'immagine), il payload run hook è unico per ogni microVM.

```
aws lambda-microvms run-microvm \  
  --image-identifier arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-image \  
  --run-hook-payload 'tenant-specific-string'
```

Quando non hai più bisogno di una microVM, chiudila per interrompere tutti gli addebiti. Per istruzioni, consulta [the section called “Terminazione di una MicroVM”](#).

## Connessione a una microVM

Ogni microVM riceve un URL di endpoint HTTPS pubblico univoco, assegnato al momento della chiamata. `run-microvm` Ti connetti alla tua applicazione in esecuzione all'interno della MicroVM tramite questo URL.

### Autenticazione

Tutte le richieste a un endpoint MicroVM richiedono un token di autenticazione JWE. Non è disponibile alcuna opzione di accesso non autenticato. Genera un token con: `create-microvm-auth-token`

```
aws lambda-microvms create-microvm-auth-token \  
  --microvm-identifier microvm-id \  
  --expiration-in-minutes 30 \  
  --allowed-ports '[{"allPorts":{}}]'
```

I token sono assegnati a porte specifiche e hanno una scadenza configurabile. Puoi limitare l'accesso a una singola porta, a un intervallo di porte o a tutte le porte:

```
{ "port": number }  
{ "range": { "startPort": N, "endPort": N } }  
{ "allPorts": {} }
```

### Routing delle porte

Per impostazione predefinita, Lambda indirizza il traffico in entrata alla porta 8080 all'interno della microVM. Per indirizzare verso una porta diversa, includi l'intestazione nella `X-aws-proxy-port`

richiesta. La porta di destinazione deve rientrare in quella `allowedPorts` definita nel token di autenticazione.

## Protocolli

Lambda MicroVMS supporta HTTP/2 WebSockets, gRPC e SSE tramite l'URL dell'endpoint.

Per le WebSocket connessioni, passa il token di autenticazione e la porta di destinazione tramite i sottoprotocolli:

```
// JavaScript WebSocket example
const protocols = [
  "lambda-microvms", // Required base protocol
  "lambda-microvms.authentication.<auth-token>", // Auth token
  "lambda-microvms.port.9000" // Target port
];
const ws = new WebSocket('wss://<microvm-endpoint>/path', protocols);
```

Lambda rimuove i MicroVM-specific sottoprotocolli dalla richiesta prima di inoltrarla all'applicazione.

## Esempi di SDK

Gli esempi seguenti mostrano come eseguire una microVM e connettersi ad essa utilizzando gli SDK. AWS

### Python

Example Esempio: esecuzione di una microVM e connessione con boto3

```
import boto3, requests
client = boto3.client("lambda-microvms")
run_resp = client.run_microvm(
    imageIdentifier="arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-image",
    idlePolicy={"autoResumeEnabled": True, "maxIdleDurationSeconds": 900,
    "suspendedDurationSeconds": 300}
)
microvm_id = run_resp["microvmId"]
endpoint = run_resp["endpoint"]
print(f"MicroVM {microvm_id} running at {endpoint}")
token_resp = client.create_microvm_auth_token(
```

```

        microvmIdentifier=microvm_id, expirationInMinutes=30, allowedPorts=[{"allPorts":
        {}]}
    )
    token = token_resp["authToken"]["X-aws-proxy-auth"]
    resp = requests.get(f"https://{endpoint}/health", headers={"X-aws-proxy-auth":
        token})
    print(resp.status_code, resp.json())

```

## Node.js

Example Esempio: esecuzione di una microVM e connessione con AWS SDK per JavaScript

```

import { LambdaMicrovmsClient, RunMicrovmCommand, CreateMicrovmAuthTokenCommand }
from "@aws-sdk/client-lambda-microvms";
const client = new LambdaMicrovmsClient({});
const { microvmId, endpoint } = await client.send(new RunMicrovmCommand({
    imageIdentifier: "arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-
image",
    idlePolicy: { autoResumeEnabled: true, maxIdleDurationSeconds: 900,
    suspendedDurationSeconds: 300 }
}));
const { authToken } = await client.send(new CreateMicrovmAuthTokenCommand({
    microvmIdentifier: microvmId, expirationInMinutes: 30, allowedPorts: [{ allPorts:
    {} }]
}));
const resp = await fetch(`https://${endpoint}/health`, {
    headers: { "X-aws-proxy-auth": authToken["X-aws-proxy-auth"] }
});
console.log(await resp.json());

```

## Invio di richieste

### Bash

Example Esempio: invio di una richiesta con cURL

```

curl 'https://<microvm-endpoint>' \
-H 'X-aws-proxy-auth: <TOKEN>' \
-H 'X-aws-proxy-port: 8080'

```

## Python

Example Esempio: invio di una richiesta con la libreria delle richieste

```
import requests
response = requests.get('https://<microvm-endpoint>', headers={'X-aws-proxy-auth':
'<TOKEN>'})
print(response.text)
```

## Node.js

Example Esempio: invio di una richiesta con fetch

```
const response = await fetch('https://<microvm-endpoint>', {
  headers: { 'X-aws-proxy-auth': '<TOKEN>', 'X-aws-proxy-port': '8080' }
});
console.log(await response.text());
```

## Hook del ciclo di vita

Gli hook del ciclo di vita consentono di eseguire una logica personalizzata nei punti chiave del ciclo di vita di MicroVM, quando viene avviato, sospeso, ripreso o terminato. Usa gli hook per inizializzare lo stato per tenant, cancellare i dati prima della sospensione, aggiornare le credenziali al momento della ripresa o ripulire le risorse prima della cessazione.

Ogni hook è un endpoint HTTP esposto dall'applicazione. Lambda invia una richiesta POST all'hook in corrispondenza dell'evento del ciclo di vita appropriato. Gli hook ascoltano il percorso `/aws/lambda-microvms/runtime/v1/<hook-name>` sulla porta configurata.

La tua MicroVM inizia a ricevere traffico esterno dopo che `/runhook` ha restituito HTTP 200. Fino ad allora, l'endpoint non inoltra le richieste all'applicazione.

| Gancio                                           | Quando invocato                       | Scopo                                                                                                                                           |
|--------------------------------------------------|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>/aws/lambda-microvms/runtime/v1/run</code> | Dopo l'avvio di MicroVM da un'istanza | Inizializza lo stato per tenant, ripristina valori univoci, esegui controlli di integrità. Il traffico inizia dopo il ritorno di questo gancio. |

| Gancio                                    | Quando invocato                                   | Scopo                                                                                                                                                                                                             |
|-------------------------------------------|---------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /aws/lambda-microvms/runtime/v1/resume    | Dopo che MicroVM riprende lo stato di sospensione | Re-establish connessioni di rete, aggiornamento delle credenziali, convalida dello stato. La MicroVM rimane nello SUSPENDED stato durante l'esecuzione di questo hook; passa a dopo il ritorno dell'hook. RUNNING |
| /aws/lambda-microvms/runtime/v1/suspend   | Prima della sospensione di MicroVM                | Elimina le scritture in sospeso, chiudi le connessioni, rilascia le risorse.                                                                                                                                      |
| /aws/lambda-microvms/runtime/v1/terminate | Prima che MicroVM termini                         | Svuota i dati, invia notifiche ai sistemi esterni, pulisci.                                                                                                                                                       |

Per gli hook che vengono eseguiti durante la creazione dell'immagine (/ready/validate), consulta. [ganci per la creazione di immagini MicroVM](#)

Specifiche OpenAPI:

```
{
  "openapi": "3.0.2",
  "info": {
    "title": "Lambda MicroVMs Application Hook Interface",
    "version": "2025-12-03"
  },
  "paths": {
    "/ready": {
      "post": {
        "description": "Called by Lambda during MicroVM image creation to determine if the application has initialized.",
        "operationId": "Ready",
        "responses": {
          "200": { "description": "Successful invocation." },
          "503": { "description": "Application is not yet ready. Lambda retries until timeout." }
        }
      }
    }
  },
}
```



```
"/resume": {
  "post": {
    "description": "Called by Lambda when resuming a MicroVM that is in the
SUSPENDED state.",
    "operationId": "Resume",
    "responses": {
      "200": { "description": "Successful invocation." }
    }
  }
},
"/run": {
  "post": {
    "description": "Called by Lambda when a new MicroVM is run from a MicroVM
image.",
    "operationId": "Run",
    "requestBody": {
      "content": {
        "application/json": {
          "schema": { "$ref": "#/components/schemas/RunRequestContent" }
        }
      }
    },
    "responses": {
      "200": { "description": "Successful invocation." }
    }
  }
},
"/suspend": {
  "post": {
    "description": "Called by Lambda when suspending a MicroVM.",
    "operationId": "Suspend",
    "responses": {
      "200": { "description": "Successful invocation." }
    }
  }
},
"/terminate": {
  "post": {
    "description": "Called by Lambda when terminating a MicroVM, before resources
are released.",
    "operationId": "Terminate",
    "responses": {
      "200": { "description": "Successful invocation." }
    }
  }
}
```

```

    }
  },
  "/validate": {
    "post": {
      "description": "Called by Lambda when running a MicroVM to validate the
image build. Use this hook to perform tests that validate your application behaves
correctly when running. Lambda also samples the portions of the image that are used
when handling this request, allowing Lambda to prefetch those portions of the image to
reduce latency at run time.",
      "operationId": "Validate",
      "responses": {
        "200": { "description": "Successful invocation." },
        "503": { "description": "Validation in progress. Lambda retries until
timeout." }
      }
    }
  }
},
"components": {
  "schemas": {
    "RunRequestContent": {
      "type": "object",
      "properties": {
        "microvmId": {
          "type": "string",
          "description": "The MicroVM identifier."
        },
        "runHookPayload": {
          "type": "string",
          "description": "Run hook payload provided to RunMicrovm."
        }
      }
    }
  }
},
"servers": [
  { "url": "/aws/lambda-microvms/runtime/v1" }
]
}

```

## Sospensione e ripristino di MicroVM

Sospendi MicroVMS per ridurre i costi preservando lo stato dell'applicazione. Durante l'esecuzione, paghi i costi di elaborazione. In caso di sospensione, paghi solo i costi di archiviazione delle istantanee.

### Come sospendere

Esistono due modi per sospendere una MicroVM:

1. Politica di inattività (automatica): da configurare `maxIdleDurationSeconds` nella politica di inattività. Se non arriva traffico all'endpoint MicroVM per tale durata, Lambda sospende automaticamente la MicroVM.
2. Chiamata API (esplicita): chiamata da sospendere immediatamente: `suspend-microvm`

```
aws lambda-microvms suspend-microvm --microvm-identifier microvm-id
```

### L'hook /suspend

Prima di sospendere, Lambda chiama il tuo hook. `/suspend` Usalo per eliminare le scritture in sospeso, chiudere le connessioni di rete e rilasciare risorse che non devono persistere oltre il limite di sospensione.

### Riprendi il comportamento

Quando una MicroVM riprende (tramite una chiamata API o un ripristino automatico), Lambda ripristina la memoria e lo stato del disco dal checkpoint di sospensione. La microVM rimane nello `SUSPENDED` stato durante l'esecuzione dell'`/resumehook`. Dopo che l'hook restituisce HTTP 200, la microVM passa e inizia a ricevere traffico. `RUNNING`

Usa l'`/resumehook` per aggiornare le credenziali, ristabilire le connessioni di rete e convalidare lo stato.

```
aws lambda-microvms resume-microvm --microvm-identifier microvm-id
```

## Auto-resume

Quando il `autoResumeEnabled=true` traffico arriva all'endpoint di una microVM sospesa, Lambda riprende la microVM automaticamente. Lambda trattiene la richiesta in entrata durante il completamento del curriculum (incluso l'hook), quindi la invia all'applicazione. `/resume`

Il curriculum aggiunge latenza alla prima richiesta. La durata dipende dalla dimensione dello stato sospeso da ripristinare e dalla durata dell'`/resumehook`.

Se la ripresa non ha esito positivo, Lambda restituisce 502 Bad Gateway al chiamante.

### Note

Auto-resume aggiunge latenza solo alla prima richiesta dopo la sospensione. Le richieste successive mentre la MicroVM è in esecuzione non sono interessate.

## Scalabilità e concorrenza

È possibile creare nuove MicroVMS chiamando. `run-microvm` Ogni MicroVM ha il proprio endpoint dedicato. Non è previsto il bilanciamento del carico tra le MicroVM da un singolo endpoint.

Account-level capacità: il tuo account ha una quota per la memoria totale che può essere allocata su tutte le tue MicroVM in SUSPENDED uno stato RUNNING o in una regione e puoi scalare verticalmente fino a quattro volte questa quota. Per richiedere un aumento della quota, visita la console Service Quotas e cerca Lambda MicroVMS.

Modello di costo:

- L'esecuzione di MicroVMS comporta costi di elaborazione.
- Le microVM sospese comportano costi di archiviazione delle istantanee ma non costi di elaborazione.
- Le MicroVM terminate non comportano alcun costo.

Strategie per la gestione della capacità:

- Sospendi le microVM inattive: configura le policy inattive per sospendere automaticamente le microVM che non ricevono traffico.

- Termina le microVM che non sono più necessarie: utilizza per terminare automaticamente dopo una durata massima di sospensione o chiama in `suspendedDurationSeconds` modo esplicito. `terminate-microvm`
- Right-size criteri di inattività: impostati in base ai modelli di traffico. `maxIdleDurationSeconds`  
Tempi di inattività più brevi consentono di liberare capacità più velocemente.

## Terminazione di una MicroVM

Terminare una microVM quando non è più necessaria. La terminazione libera tutte le risorse di calcolo e interrompe tutti gli addebiti.

Prima di rilasciare risorse, Lambda richiama l'utente. `/terminate` Usalo per cancellare i dati in sospeso o inviare notifiche a sistemi esterni.

```
aws lambda-microvms terminate-microvm --microvm-identifier microvm-id
```

## Elenco delle microVMS

Elenca tutti i MicroVM presenti nel tuo account, opzionalmente filtrati per immagine:

```
aws lambda-microvms list-microvms

# Filter by image
aws lambda-microvms list-microvms --image-identifier my-image --image-version 1.0
```

## Gestione degli errori

### Errori di esecuzione

La tabella seguente elenca gli errori più comuni restituiti dall'`run-microvmAPI`:

| Errore                                     | Causa                                                                 | Soluzione                                                      |
|--------------------------------------------|-----------------------------------------------------------------------|----------------------------------------------------------------|
| <code>ServiceQuotaExceededException</code> | L'account ha raggiunto la quota di memoria per le MicroVM simultanee. | Chiudi le MicroVMS inattive o richiedi un aumento della quota. |

| Errore                    | Causa                                                             | Soluzione                                                                                                       |
|---------------------------|-------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| ResourceNotFoundException | L'immagine specifica non esiste o non è in stato. CREATED         | Verifica l'identificatore dell'immagine e conferma che la compilazione è stata completata.                      |
| ValidationException       | Uno o più parametri della richiesta non sono validi.              | Controlla i valori dei criteri inattivi, il formato dell'identificatore dell'immagine e gli ARN dei connettori. |
| ThrottlingException       | Il limite di velocità API per questa operazione è stato superato. | Implementa il backoff esponenziale con jitter.                                                                  |

## Riprova la strategia

Per gli errori transitori (`ThrottlingException`, `InternalServerErrorException`), usa il backoff esponenziale:

```
import time, random
def run_with_retry(client, params, max_retries=5):
    for attempt in range(max_retries):
        try:
            return client.run_microvm(**params)
        except client.exceptions.ThrottlingException:
            delay = (2 ** attempt) + random.uniform(0, 1)
            time.sleep(delay)
    raise Exception("Max retries exceeded")
```

Per informazioni sulle integrazioni di servizi supportate con Lambda Managed Instances, consulta [the section called “Integrazioni”](#)

## Rete

È possibile configurare l'accesso alla rete per la AWS Lambda microVM associando le risorse di Network Connector alla microVM in fase di esecuzione. I connettori di rete vengono specificati al momento della chiamata `run-microvm` e non possono essere modificati mentre una MicroVM è in esecuzione.

## Panoramica di

Ogni microVM può avere configurazioni di rete in ingresso (in entrata) e in uscita (in uscita) indipendenti:

- I connettori di rete in ingresso consentono la connettività in ingresso. I client si connettono a un endpoint HTTPS gestito dal servizio e Lambda inoltra il traffico alle porte configurate all'interno della MicroVM. I connettori di ingresso sono AWS gestiti: si fa riferimento ad essi tramite ARN quando si esegue una microVM.
- I connettori di rete in uscita consentono il traffico in uscita. Per impostazione predefinita, le microVM dispongono di un accesso pubblico a Internet. Puoi invece creare un connettore di uscita VPC gestito dal cliente per indirizzare il traffico in uscita attraverso il tuo VPC.

Un singolo connettore può essere riutilizzato su più microVM: questo è il modello di utilizzo previsto.

## Connettività in entrata

Ogni Lambda MicroVM è raggiungibile tramite un URL di endpoint HTTPS univoco, assegnato al momento della chiamata. `run-microvm` I client inviano richieste a questo endpoint tramite HTTPS. Lambda indirizza ogni richiesta a una porta all'interno della microVM, dove l'applicazione la riceve.

Per impostazione predefinita, le richieste ricevute all'endpoint vengono indirizzate alla porta 8080 all'interno della microVM. Per il routing verso una porta diversa, vedere. [the section called “Routing delle porte”](#)

I seguenti protocolli sono supportati sull'endpoint in entrata:

- HTTP/1.1
- HTTP/2
- WebSockets
- gRPC
- Server-Sent Eventi (SSE)

 Note

Il traffico tra il client e l'endpoint MicroVM è sempre crittografato con TLS. L'applicazione può gestire internamente le richieste tramite HTTP o HTTPS.

## Routing delle porte

Lambda seleziona la porta di destinazione all'interno della microVM utilizzando il seguente ordine di priorità:

1. **X-aws-proxy-portheader**: per le richieste HTTP standard, includi questa intestazione con il numero di porta di destinazione.
2. WebSocket sottoprotocollo: se il WebSocket client non è in grado di impostare intestazioni personalizzate, specifica la porta di destinazione come sottoprotocollo denominato `lambda-microvms.port.N`, dov'è *N* il numero di porta. Fornisci dei sottoprotocolli quando apri la connessione. WebSocket Per un esempio, consulta [the section called "Protocolli"](#).
3. Predefinito (8080): se non viene specificato nessuno dei due, richiede l'instradamento verso la porta 8080.

 Important

La porta di destinazione deve rientrare in quella `allowedPorts` definita nel token di autenticazione. Le richieste verso porte non autorizzate ricevono una risposta 403 Forbidden.

## Autenticazione

Tutte le richieste a un endpoint MicroVM richiedono un token di autenticazione valido nell'intestazione. `X-aws-proxy-auth` I token vengono generati utilizzando. `create-microvm-auth-token` Ogni token è una stringa JWE (JSON Web Encryption) crittografata con l'ambito di:

- Una microVM specifica (identificata da un ID).
- Un set di porte consentite (porta singola, intervallo o tutte le porte).
- Un'ora di scadenza (configurata al momento della creazione del token).



L'esempio seguente crea un token e lo utilizza per inviare una richiesta autenticata:

```
aws lambda-microvms create-microvm-auth-token \  
  --microvm-identifier microvm-id \  
  --expiration-in-minutes 30 \  
  --allowed-ports '[{"port":8080}]'
```

```
curl 'https://microvm-endpoint' \  
  -H 'X-aws-proxy-auth: TOKEN' \  
  -H 'X-aws-proxy-port: 8080'
```

Per una procedura dettagliata sulla creazione di token e sulla connessione a una MicroVM, comprese le connessioni, consulta. WebSocket [the section called “Connessione a una microVM”](#)

## Risposte agli errori

I seguenti codici di stato HTTP vengono restituiti dall'endpoint MicroVM quando non è in grado di elaborare o consegnare una richiesta all'applicazione. Queste risposte provengono dall'endpoint, non dall'applicazione.

| Codice | Stato                     | Causa e risoluzione                                                                                                                                                                    |
|--------|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 400    | Richiesta non valida      | Richiesta non valida o intestazione di porta o sottoprotocollo non validi. WebSocket Verifica il formato.                                                                              |
| 403    | Accesso negato            | Token mancante, scaduto o non valido; oppure la porta richiesta non è inclusa in quella del token. <code>allowedPorts</code> Genera un nuovo token o utilizza una porta consentita.    |
| 429    | Troppe richieste          | Limite di velocità superato (a livello di account o per microVM). Riprova con un backoff esponenziale.                                                                                 |
| 500    | Errore interno del server | Si è verificato un errore interno. Riprova la richiesta .                                                                                                                              |
| 502    | Gateway non valido        | L'applicazione non risponde o il ripristino automatico non è riuscito entro il numero massimo di tentativi di ripetizione. Consulta <a href="#">the section called “Auto-resume”</a> . |

## Intestazioni della richiesta

Lo spazio dei nomi dell'`X-aws-proxy-*` intestazione è riservato da Lambda per i metadati della richiesta, come il token di autenticazione (`X-aws-proxy-auth`) e la porta di destinazione (`X-aws-proxy-port`). Lambda rimuove le `X-aws-proxy-*` intestazioni prima di inoltrare la richiesta all'applicazione.

## Request/response larghezza di banda

Ogni Lambda MicroVM ha una request/response larghezza di banda che si ridimensiona linearmente in base alle sue dimensioni. Questa larghezza di banda si applica a tutto il traffico attraverso l'endpoint MicroVM, sia alle richieste in entrata che alle risposte in uscita.

| Dimensioni della microVM (linea di base) | Larghezza di banda massima |
|------------------------------------------|----------------------------|
| 0,5 GB, 0,25 vCPU                        | MB/s (18 Mbps)             |
| 1 GB, 0,5 vCPU                           | 2 MB/s (16 Mbps)           |
| 2 GB, 1 vCPU                             | 4 MB/s (32 Mbps)           |
| 4 GB, 2 vCPU                             | 8 MB/s (64 Mbps)           |
| 8 GB, 4 vCPU                             | 16 MB/s (128 Mbps)         |

Se riscontri una maggiore latenza delle richieste a causa della saturazione della rete, riduci la concorrenza delle richieste o le dimensioni del payload oppure seleziona una microVM di dimensioni maggiori per aumentare la larghezza di banda disponibile.

## HTTP/2 supporto

Supporto per Lambda MicroVMs HTTP/2 sull'endpoint in ingresso. Lambda negozia il protocollo tramite ALPN (Protocol Negotiation) durante l'Application-Layer handshake TLS, preferendo e ricorrendo a HTTP/2 invece di HTTP/1.1. Un client lo utilizza automaticamente. HTTP/2-capable

Da utilizzare HTTP/2 tra l'endpoint e l'applicazione all'interno della MicroVM:

- La tua applicazione funziona con TLS: Lambda negozia HTTP/2 con la tua applicazione tramite ALPN, ricorrendo a se non è supportata. HTTP/1.1 invece di HTTP/2

- L'applicazione utilizza il protocollo HTTP in chiaro: includi l'`X-aws-proxy-force-h2` header nella richiesta da utilizzare HTTP/2 per la connessione all'applicazione.

## Connettività in uscita

Per impostazione predefinita, le microVM Lambda dispongono di un accesso pubblico a Internet sul percorso di uscita. Per connettere le microVM alle risorse dei tuoi VPC privati, come RDS, ElastiCache API interne e sistemi locali tramite Direct Connect o VPN, crea un Lambda Network Connector con la tua configurazione VPC.

Quando si utilizza il VPC in uscita, il traffico in uscita è soggetto alle regole del gruppo di sicurezza e agli ACL di rete che regolano il traffico nel VPC.

## Utilizzo dei connettori di rete in uscita

I connettori di rete in uscita indirizzano il traffico in uscita dalla tua MicroVM al tuo VPC. Si crea un connettore una volta, quindi si fa riferimento ad esso tramite ARN all'avvio di MicroVMS tramite il comando `run-microvm`

## Prerequisiti

Prima di creare un connettore di rete, è necessario un ruolo IAM che consenta a Lambda di creare interfacce di rete elastiche (ENI) nel tuo VPC. Il ruolo richiede le seguenti autorizzazioni:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CreateENI",
      "Effect": "Allow",
      "Action": "ec2:CreateNetworkInterface",
      "Resource": [
        "arn:aws:ec2:*:*:network-interface/*",
        "arn:aws:ec2:*:*:subnet/*",
        "arn:aws:ec2:*:*:security-group/*"
      ]
    },
    {
      "Sid": "TagENI",
      "Effect": "Allow",
```

```

    "Action": "ec2:CreateTags",
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
      "StringEquals": {
        "ec2:ManagedResourceOperator": "network-connectors.lambda.amazonaws.com"
      }
    }
  }
}
]
}

```

## Creazione di un connettore di rete

Crea un connettore specificando le sottoreti VPC, i gruppi di sicurezza e il protocollo di rete (oppure):  
IPv4 DualStack

```

aws lambda-core create-network-connector \
  --name my-connector \
  --configuration '{
    "VpcEgressConfiguration": {
      "SubnetIds": ["subnet-xxx"],
      "SecurityGroupIds": ["sg-xxx"],
      "NetworkProtocol": "IPv4",
      "AssociatedComputeResourceTypes": ["MicroVm"]
    }
  }' \
  --operator-role arn:aws:iam::123456789012:role/NetworkConnectorOperatorRole

```

## Stati del connettore di rete

Un connettore deve essere nello ACTIVE stato prima di potervi fare riferimento a un-microvm.

| Stato    | Description                                                                                  |
|----------|----------------------------------------------------------------------------------------------|
| PENDING  | Il connettore è in fase di creazione (viene eseguito il provisioning degli ENI sottostanti). |
| ACTIVE   | Il connettore è pronto per l'uso.                                                            |
| INACTIVE | Il connettore è temporaneamente inattivo.                                                    |

| Stato         | Description                                                                  |
|---------------|------------------------------------------------------------------------------|
| FAILED        | Il provisioning o l'aggiornamento non sono riusciti. Controlla StateReason . |
| DELETING      | Il connettore è in fase di eliminazione; gli ENI sono in fase di pulizia.    |
| DELETE_FAILED | Eliminazione non riuscita.                                                   |

## Esecuzione di una microVM con un connettore di rete

Fai riferimento al connettore ARN quando esegui una microVM:

```
aws lambda-microvms run-microvm \
  --image-identifier arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-
image \
  --egress-network-connectors connector-arn \
  --idle-policy
'{"maxIdleDurationSeconds":900,"suspendedDurationSeconds":1800,"autoResumeEnabled":false}'
```

### Note

Prima di aggiornare o eliminare un connettore, assicurati che tutte le MicroVM che lo utilizzano siano terminate. La modifica di un connettore in uso attivo può causare problemi di connettività di rete per l'esecuzione di MicroVMS.

## Utilizzo di Lambda MicroVMS con endpoint VPC di interfaccia (AWS PrivateLink)

Puoi utilizzarlo AWS PrivateLink per la connettività privata sulla AWS rete tra le tue risorse VPC e Lambda MicroVMS, senza attraversare la rete Internet pubblica. MicroVMS supporta due endpoint VPC, a seconda della destinazione del traffico desiderata:

- API di gestione MicroVM (creazione di immagini, esecuzione, sospensione, interruzione): utilizza l'endpoint Lambda VPC esistente (). `com.amazonaws.region.lambda`
- Connettività a MicroVMS (traffico HTTPS verso le applicazioni in esecuzione): utilizza un endpoint separato (). `com.amazonaws.region.lambda-microvm`

## Endpoint VPC per le API di gestione delle microVM

Lambda MicroVMS condivide lo stesso servizio di endpoint VPC di Lambda ().

com.amazonaws.*region*.lambda Per istruzioni complete, consulta [Creazione di un endpoint di interfaccia per Lambda](#).

### Politica degli endpoint per le API di gestione MicroVM

Per controllare chi può utilizzare l'endpoint dell'interfaccia e quali azioni dell'API Lambda MicroVMS può eseguire, allega una policy sugli endpoint. La policy specifica il principale che può eseguire le azioni, le azioni che può eseguire e le risorse su cui può agire. Le azioni Lambda MicroVMS utilizzano il prefisso di azione IAM `lambda:`.

Per ulteriori informazioni, consulta [Controllo degli accessi ai servizi con endpoint VPC](#) nella Guida per l'utente di Amazon VPC.

La seguente policy di esempio consente all'utente di `MyUser` elencare e ottenere immagini MicroVM tramite l'endpoint:

```
{
  "Statement": [
    {
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/MyUser"
      },
      "Effect": "Allow",
      "Action": [
        "lambda:ListMicrovmImages",
        "lambda:GetMicrovmImage"
      ],
      "Resource": "*"
    }
  ]
}
```

## Endpoint VPC per la connettività MicroVM

Per mantenere privato il traffico HTTPS verso le microVMS in esecuzione, crea un endpoint di interfaccia per il servizio. `com.amazonaws.region.lambda-microvm` Questo endpoint gestisce le connessioni agli URL degli endpoint MicroVM (ad esempio). `abc123def456.lambda-microvm.us-east-1.on.aws`

Per ulteriori informazioni sulle proprietà degli endpoint di interfaccia, consulta la guida agli endpoint di [interfaccia nella documentazione di Amazon VPC](#).

## Creazione dell'endpoint

Per creare un endpoint di interfaccia per la connettività MicroVM (console)

1. Aprire la [pagina Endpoint](#) della console Amazon VPC.
2. Seleziona Crea endpoint.
3. In Categoria del servizio, assicurati che sia selezionato Servizi di AWS .
4. Per Service Name (Nome del servizio), selezionare `com.amazonaws.region.lambda-microvm`. Verifica che il tipo sia Interfaccia.
5. Creazione di un VPC e delle sottoreti
6. Per abilitare il DNS privato per l'endpoint dell'interfaccia, seleziona la casella di controllo Abilita nome DNS (scelta consigliata). Ciò garantisce che le richieste che utilizzano il nome host pubblico dell'endpoint MicroVM si risolvano automaticamente nell'endpoint dell'interfaccia, senza che siano necessarie modifiche sul lato client.
7. Per gruppo di sicurezza, scegliere uno o più gruppi di sicurezza. Il gruppo di sicurezza deve consentire il traffico TCP in uscita sulla porta 443 verso le interfacce di rete degli endpoint.
8. Seleziona Crea endpoint.

Per utilizzare l'opzione DNS privato, devi impostare gli attributi `enableDnsHostnames` tuo VPC. `enableDnsSupport` Per ulteriori informazioni, consulta [Visualizzazione e aggiornamento del supporto DNS per il VPC](#) nella Guida per l'utente di Amazon VPC.

Per creare un endpoint di interfaccia per la connettività MicroVM (CLI)AWS

```
aws ec2 create-vpc-endpoint \  
  --vpc-id vpc-ec43eb89 \  
  --vpc-endpoint-type Interface \  
  --service-name com.amazonaws.us-east-1.lambda-microvm \  
  --subnet-id subnet-abababab \  
  --security-group-id sg-1a2b3c4d \  
  --private-dns-enabled
```

Per verificare che l'endpoint sia disponibile e che il DNS privato sia attivo:

```
aws ec2 describe-vpc-endpoints \  

```

```
--vpc-endpoint-ids vpce-1a2b3c4d5e6f7g8h9 \
--query 'VpcEndpoints[0].
{State:State,PrivateDns:PrivateDnsEnabled,Dns:DnsEntries[*].DnsName}'
```

## Comportamento DNS privato

Quando il DNS privato è abilitato: l'endpoint gestisce la risoluzione DNS all'interno del tuo VPC.

\*.lambda-microvm.*region*.on.aws I nomi host degli endpoint MicroVM esistenti (ad esempio abc123def456.lambda-microvm.us-east-1.on.aws) vengono risolti negli indirizzi IP privati delle interfacce di rete degli endpoint. Non è richiesta alcuna modifica del client.

Quando il DNS privato è disabilitato, Amazon VPC genera un nome DNS specifico per l'endpoint nel modulo. *vpce-id-hash*.lambda-microvm.*region*.vpce.amazonaws.com Per indirizzare il traffico attraverso questo endpoint continuando a raggiungere la microVM corretta, devi conservare il nome host MicroVM originale in due posizioni:

- Indicazione del nome del server TLS (SNI): l'handshake TLS utilizza questo valore per identificare a quale microVM è destinata la connessione.
- Intestazione HTTP Host: il proxy utilizza questo valore per indirizzare la richiesta alla MicroVM corretta.

Se uno dei valori è impostato sul nome host dell'endpoint VPC anziché sul nome host MicroVM, la connessione non può essere indirizzata alla MicroVM corretta.

Esempio: connessione tramite un nome DNS specifico dell'endpoint

L'esempio seguente utilizza [curl](#) con il `--connect-to` flag per reindirizzare la connessione TCP all'endpoint VPC mantenendo il nome host MicroVM nell'URL, nell'SNI e nell'intestazione Host:

```
ENDPOINT_HOST=abc123def456.lambda-microvm.us-east-1.on.aws
VPCE_HOST=vpce-0a1b2c3d4e5f67890-a1b2c3d4.lambda-microvm.us-east-1.vpce.amazonaws.com

curl --connect-to "$ENDPOINT_HOST:443:$VPCE_HOST:443" \
-H "x-aws-proxy-auth: $MICROVM_AUTH_TOKEN" \
-H "x-aws-proxy-port: 8080" \
"https://$ENDPOINT_HOST/"
```

Il `--connect-to` flag indica a curl di aprire la connessione TCP all'indirizzo dell'endpoint VPC, mentre l'URL, l'SNI TLS e l'intestazione Host rimangono impostati sul nome host MicroVM.



Per ulteriori informazioni, consultare [Accesso a un servizio tramite un endpoint di interfaccia](#) in Guida per l'utente di Amazon VPC.

## Politica degli endpoint per la connettività MicroVM

È possibile allegare una policy sugli endpoint per controllare quali microVM sono raggiungibili tramite l'endpoint VPC. `lambda-microvm` Una policy sugli endpoint sul `lambda-microvm` servizio consente di definire l'ambito delle connessioni a account o organizzazioni specifici. Per impostazione predefinita, l'endpoint consente le connessioni a MicroVMS in qualsiasi account. AWS (Nota: il client che stabilisce la connessione deve ancora possedere un token di autenticazione MicroVM valido per ottenere l'accesso).

Per impostazione predefinita, un endpoint VPC ha una politica di accesso completo che consente tutto il traffico. Quando si sostituisce la policy predefinita con una policy personalizzata, Lambda MicroVMS valuta tale policy rispetto all'`lambda:ConnectMicrovm` azione su ogni connessione effettuata tramite l'endpoint. Se la policy non consente la connessione a MicroVMS, la connessione viene rifiutata con una risposta HTTP 403 Forbidden. Una policy che non concede `lambda:ConnectMicrovm` nega tutte le connessioni tramite l'endpoint.

### Note

L'`lambda:ConnectMicrovm` azione autorizza una connessione a un endpoint MicroVM tramite l'endpoint dell'interfaccia. Non è un'operazione API Lambda e non può essere utilizzata nelle policy IAM basate sull'identità o sulle risorse: è valida solo in una policy sugli endpoint VPC.

## Principal e risorsa

Le connessioni a un endpoint MicroVM sono autenticate con un token di autenticazione MicroVM anziché con Signature Version 4. AWS Per questo motivo, nessun principale IAM è associato alla connessione. Invece, Lambda MicroVMS valuta la policy degli endpoint con un principale anonimo. Ciò significa che:

- `Principal` deve essere `"*"`. Una policy che nomina un principale specifico non corrisponde a nulla e nega ogni connessione.
- Le chiavi di condizione che dipendono dall'identità del richiedente (ad esempio `aws:PrincipalArn`, `aws:PrincipalOrgID`, `aws:user-id`) non vengono compilate e non corrisponderanno.

- **Resource** deve anche esserlo. "\*" Lambda MicroVMS non prevede la valutazione delle policy degli endpoint per gli ARN delle singole risorse MicroVM. Per limitare le microVM che l'endpoint può raggiungere, utilizza la chiave di condizione anziché l'elemento `aws:ResourceAccountResource`

### Chiavi di condizione supportate

| Chiave di condizione             | Description                                                                     |
|----------------------------------|---------------------------------------------------------------------------------|
| <code>aws:ResourceAccount</code> | L' AWS account proprietario della microVM a cui si sta connettendo.             |
| <code>aws:ResourceOrgID</code>   | L'ID dell' AWS organizzazione dell'account proprietario della MicroVM.          |
| <code>aws:SourceVpce</code>      | L'ID dell'endpoint di interfaccia attraverso il quale è passata la connessione. |
| <code>aws:SourceVpc</code>       | L'ID del VPC da cui proviene la connessione.                                    |
| <code>aws:VpcSourceIp</code>     | L'indirizzo IP privato del client che ha effettuato la connessione.             |

### Esempio: consenti le connessioni solo a MicroVMS nel tuo account

La seguente politica sugli endpoint consente le connessioni tramite l'endpoint solo a MicroVMS di proprietà dell'account 111122223333. Le connessioni a MicroVMS di proprietà di qualsiasi altro account vengono negate.

```
{
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": "lambda:ConnectMicrovm",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        }
      }
    }
  ]
}
```

```
}  
  }  
  ]  
}
```

## Sicurezza e autorizzazioni

Questa sezione descrive i ruoli IAM, i token di autenticazione e i controlli di accesso per AWS Lambda MicroVMS.

### Ruoli IAM

#### Ruolo di creazione

Questo ruolo viene utilizzato durante la creazione dell'immagine. Richiede le seguenti autorizzazioni per recuperare gli artefatti di origine e generare i log: `s3:GetObject`, `logs:CreateLogGroup` e `logs:CreateLogStream` `logs:PutLogEvents`. Aggiungete anche `ecr:GetAuthorizationToken` se fate riferimento a immagini ECR private.

Il ruolo di compilazione è facoltativo. Se non fornisci un ruolo di compilazione, Lambda non può scrivere i log di compilazione in CloudWatch.

#### Ruolo di esecuzione

Questo ruolo viene utilizzato nel runtime di MicroVM per fornire le autorizzazioni di accesso per scrivere i log CloudWatch e utilizzare altri servizi. AWS

Il ruolo di esecuzione è facoltativo. Se non fornisci un ruolo di esecuzione, Lambda non emette log di runtime CloudWatch e la tua MicroVM non può accedere ad altri servizi. AWS

Gli hook del ciclo di vita vengono eseguiti nel ruolo associato alla loro fase. Build-time gli hook (`/ready/validate`) vengono eseguiti nel ruolo di compilazione. Gli hook di runtime (`/run`, `/resume/suspend`, `e/terminate`) vengono eseguiti sotto il ruolo di esecuzione.

### Policy di trust

Entrambi i ruoli richiedono una politica di fiducia che consenta al responsabile del servizio Lambda `lambda.amazonaws.com` di eseguire `sts:AssumeRole` e `sts:TagSession`

## Token di autenticazione

I token controllano l'accesso in entrata a MicroVMS in esecuzione. Sono basati sull'ambito delle porte; ogni token specifica a quali porte concede l'accesso.

```
# Token granting access to all ports
aws lambda-microvms create-microvm-auth-token \
  --microvm-identifier <id> \
  --expiration-in-minutes 30 \
  --allowed-ports '["allPorts":{}]'

# Token granting access to port 8080 only
aws lambda-microvms create-microvm-auth-token \
  --microvm-identifier <id> \
  --expiration-in-minutes 30 \
  --allowed-ports '["port":8080]'
```

## Token di accesso alla shell

L'accesso alla shell utilizza un'API token separata. La microVM di destinazione deve essere stata eseguita con il connettore di SHELL\_INGRESS rete (`arn:aws:lambda:us-east-1:aws:network-connector:aws-network-connector:SHELL_INGRESS`). Se la MicroVM non è stata avviata con questo connettore, `create-microvm-shell-auth-token` si verifica un errore con: `ValidationException`

```
aws lambda-microvms create-microvm-shell-auth-token \
  --microvm-identifier <id> \
  --expiration-in-minutes 30
```

Per una procedura dettagliata sulla connessione a una shell MicroVM, consulta [the section called “Accesso alla shell”](#)

## Applicazione di tag alle risorse

Le immagini Lambda MicroVM supportano i tag AWS delle risorse per l'organizzazione e l'allocazione dei costi. Aggiungi tag durante la creazione di un'immagine utilizzando il `--tags` parametro o in modo esplicito dopo la creazione. Puoi anche elencare ed eliminare i tag, come illustrato di seguito: puoi anche fare riferimento ai tag nelle condizioni delle policy IAM per controllare l'accesso alle risorse MicroVM.

```
# Add tags when creating an image
aws lambda-microvms create-microvm-image \
  --name my-app-image \
  --code-artifact '{"uri":"s3://my-bucket/app.zip"}' \
  --base-image-arn arn:aws:lambda:us-east-1:aws:microvm-image:al2023-1 \
  --build-role-arn arn:aws:iam::123456789012:role/MicrovmBuildRole \
  --tags '{"Environment":"production","Team":"platform"}'

# List tags on a resource
aws lambda-microvms list-tags \
  --resource arn:aws:lambda:us-east-1:123456789012:microvm-image:my-app-image

# Add tags to an existing resource
aws lambda-microvms tag-resource \
  --resource arn:aws:lambda:us-east-1:123456789012:microvm-image:my-app-image \
  --tags '{"CostCenter":"12345","Project":"sandbox-platform"}'

# Remove tags by key
aws lambda-microvms untag-resource \
  --resource arn:aws:lambda:us-east-1:123456789012:microvm-image:my-app-image \
  --tag-keys ['"CostCenter","Project"]'
```

## Riferimento alle autorizzazioni IAM

Lambda MicroVMS definisce le seguenti azioni IAM. Utilizzale nelle policy IAM per controllare l'accesso alle operazioni di MicroVM.

| Azione                                    | Description                               |
|-------------------------------------------|-------------------------------------------|
| <code>lambda:CreateMicrovmImage</code>    | Crea una nuova immagine MicroVM.          |
| <code>lambda:UpdateMicrovmImage</code>    | Aggiorna un'immagine MicroVM esistente.   |
| <code>lambda&gt;DeleteMicrovmImage</code> | Elimina un'immagine MicroVM.              |
| <code>lambda:GetMicrovmImage</code>       | Ottieni i dettagli dell'immagine MicroVM. |
| <code>lambda:ListMicrovmImages</code>     | Elenca le immagini MicroVM nell'account.  |
| <code>lambda:RunMicrovm</code>            | Esegui una nuova MicroVM.                 |

| Azione                                          | Description                                        |
|-------------------------------------------------|----------------------------------------------------|
| <code>lambda:GetMicrovm</code>                  | Ottieni lo stato e i dettagli della microVM.       |
| <code>lambda:ListMicrovms</code>                | Elenca le MicroVMS nell'account.                   |
| <code>lambda:SuspendMicrovm</code>              | Sospendi una MicroVM in esecuzione.                |
| <code>lambda:ResumeMicrovm</code>               | Riprendi una MicroVM sospesa.                      |
| <code>lambda:TerminateMicrovm</code>            | Terminare una MicroVM.                             |
| <code>lambda:CreateMicrovmAuthToken</code>      | Genera un token di autenticazione per una MicroVM. |
| <code>lambda:CreateMicrovmShellAuthToken</code> | Genera un token di accesso alla shell.             |

## Formati ARN delle risorse

```
# MicroVM image
arn:aws:lambda:<region>:<account>:microvm-image:<image-name>
# Network connector
arn:aws:lambda:<region>:<account>:network-connector:<connector-id>
```

## Esempio: Least-privilege politica per un operatore

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:ListMicrovms",
        "lambda:ListMicrovmImages"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
```

```
    "Action": [
      "lambda:RunMicrovm",
      "lambda:GetMicrovm",
      "lambda:SuspendMicrovm",
      "lambda:ResumeMicrovm",
      "lambda:TerminateMicrovm",
      "lambda:CreateMicrovmAuthToken",
      "lambda:GetMicrovmImage"
    ],
    "Resource": "arn:aws:lambda:*:123456789012:microvm-image:*"
  }
]
```

## Monitoraggio

Questa pagina tratta la CloudWatch registrazione (build e runtime) e il CloudTrail controllo per AWS Lambda MicroVMS.

### Registrazione (registri) CloudWatch

AWS Lambda MicroVMS trasmette i log in due fasi: CloudWatch

#### Crea registri

Lambda genera registri di compilazione durante la creazione dell'immagine, inclusi l'output Dockerfile dell'esecuzione, l'avvio dell'applicazione e le invocazioni degli hook. Questi log vengono trasmessi in streaming utilizzando il ruolo build e scritti nel gruppo di log predefinito. /aws/lambda-microvms/<image-name>

#### Log di runtime

Lambda trasmette in streaming le applicazioni stdout e stderr dall'esecuzione di MicroVMS a CloudWatch. I log di runtime vengono trasmessi in streaming utilizzando il ruolo di esecuzione e scritti nello stesso gruppo di log predefinito, con il flusso di log impostato per impostazione predefinita sull'ID MicroVM.

#### Important

Senza un ruolo di esecuzione, lo stdout e lo stderr dell'applicazione non vengono inoltrati in fase di esecuzione. CloudWatch Per ricevere i log di runtime, fornisci un ruolo di

```
esecuzione con, e le autorizzazioni. logs:CreateLogGroup logs:CreateLogStream  
logs:PutLogEvents
```

## Destinazioni di registro personalizzate

Puoi configurare un gruppo di log e uno stream personalizzati utilizzando il `--logging` parametro on `run-microvm` o `create-microvm-image`:

```
--logging '{"cloudWatch":{"logGroup":"/my-app/microvms","logStream":"custom-stream"}}'
```

Se si omette `logStream`, il valore predefinito è l'ID MicroVM.

È possibile disabilitare esplicitamente la registrazione passando la seguente configurazione:

```
--logging '{"disabled":{}}'
```

## CloudTrail registrazione

Lambda MicroVM è integrato con CloudTrail un servizio che fornisce una registrazione delle azioni intraprese da un utente, un ruolo o un servizio. AWS CloudTrail acquisisce tutte le chiamate API per Lambda MicroVMS come eventi. Le chiamate acquisite includono chiamate dalla console Lambda MicroVMS e chiamate in codice alle operazioni dell'API Lambda MicroVMS.

Se crei un percorso, puoi abilitare la distribuzione continua di eventi a un bucket Amazon S3, CloudTrail inclusi gli eventi per Lambda MicroVMS. Se non configuri un percorso, puoi comunque visualizzare gli eventi più recenti nella console nella cronologia degli eventi. CloudTrail

Utilizzando le informazioni raccolte da CloudTrail, è possibile determinare la richiesta effettuata a Lambda MicroVMS, l'indirizzo IP da cui è stata effettuata la richiesta, chi ha effettuato la richiesta, quando è stata effettuata e dettagli aggiuntivi.

CloudTrail registra due tipi di eventi per Lambda MicroVMS: eventi di gestione (operazioni del piano di controllo come creazione ed eliminazione di immagini) ed eventi di dati (operazioni del piano dati come l'esecuzione e la chiusura di MicroVMS). Gli eventi di gestione vengono registrati per impostazione predefinita. Gli eventi relativi ai dati richiedono un consenso esplicito.



## Eventi di gestione

Gli eventi di gestione di Lambda MicroVMS sono operazioni del piano di controllo che vengono registrate per impostazione predefinita. CloudTrail Gli eventi di gestione includono la creazione, l'aggiornamento e l'eliminazione di immagini MicroVM, nonché l'elenco e la descrizione delle risorse.

Le seguenti azioni Lambda MicroVMS vengono registrate come eventi di gestione:

- `CreateMicrovmImage`
- `DeleteMicrovmImage`
- `DeleteMicrovmImageVersion`
- `UpdateMicrovmImage`
- `UpdateMicrovmImageVersion`
- `ListMicrovmImages`
- `GetMicrovmImage`
- `ListMicrovmImageVersions`
- `GetMicrovmImageVersion`
- `ListMicrovmImageBuilds`
- `GetMicrovmImageBuild`
- `ListMicrovms`
- `GetMicrovm`
- `ListManagedMicrovmImages`

## Eventi di dati

Gli eventi di dati forniscono informazioni sulle operazioni sulle risorse eseguite su o all'interno di una risorsa (ad esempio, l'esecuzione o la chiusura di una MicroVM). Queste operazioni sono definite anche operazioni del piano dei dati. Gli eventi di dati sono spesso attività che interessano volumi elevati di dati. Per impostazione predefinita, CloudTrail non registra gli eventi relativi ai dati. È necessario abilitare esplicitamente la registrazione degli eventi di dati per Lambda MicroVMS.

Il tipo di CloudTrail risorsa per gli eventi di dati Lambda MicroVMS è.

`AWS::Lambda::MicrovmImage` È possibile utilizzare la CloudTrail console o la AWS CLI per configurare un data store di percorsi o eventi per registrare gli eventi dei dati.

Le seguenti azioni Lambda MicroVMS vengono registrate come eventi di dati:

- `RunMicrovm`
- `TerminateMicrovm`
- `SuspendMicrovm`
- `ResumeMicrovm`
- `CreateMicrovmAuthToken`
- `CreateMicrovmShellAuthToken`

Per registrare gli eventi relativi ai dati per tutte le immagini MicroVM, utilizza il seguente selettore di eventi avanzato:

```
{
  "Name": "Log Lambda MicroVM data events",
  "FieldSelectors": [
    { "Field": "eventCategory", "Equals": ["Data"] },
    { "Field": "resources.type", "Equals": ["AWS::Lambda::MicrovmImage"] }
  ]
}
```

## Abilitazione della registrazione degli eventi relativi ai dati

Per abilitare la registrazione degli eventi di dati utilizzando la AWS CLI, esegui il `put-event-selectors` comando con selettori di eventi avanzati:

```
aws cloudtrail put-event-selectors \
  --trail-name my-trail \
  --advanced-event-selectors '[{
    "Name": "Log Lambda MicroVM data events",
    "FieldSelectors": [
      { "Field": "eventCategory", "Equals": ["Data"] },
      { "Field": "resources.type", "Equals": ["AWS::Lambda::MicrovmImage"] }
    ]
  }]'
```

## Comprensione delle voci di registro CloudTrail

Un CloudTrail percorso fornisce i record delle attività delle API a un bucket Amazon S3 specificato. Ogni voce di registro rappresenta una singola richiesta API e include l'identità del chiamante, il timestamp, i parametri della richiesta e la risposta.

L'esempio seguente mostra una voce di CloudTrail registro che dimostra un evento di dati per l'azione: RunMicrovm

```
{
  "eventVersion": "1.11",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "A1B2C3D4E5F6G7EXAMPLE:mySessionName",
    "arn": "arn:aws:sts::111122223333:assumed-role/MyRole/mySessionName",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
  },
  "eventTime": "2026-04-20T18:26:44Z",
  "eventSource": "lambda.amazonaws.com",
  "eventName": "RunMicrovm",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "requestParameters": {
    "microvmImageArn": "arn:aws:lambda:us-east-1:111122223333:microvm-image:my-image"
  },
  "responseElements": {
    "microvmId": "ai-12345678-abcd-1234-ef56-123456789abc",
    "microvmState": "PENDING",
    "endpoint": "12345678-abcd-1234-ef56-123456789abc.lambda-microvm.us-east-1.on.aws",
    "microvmImageArn": "arn:aws:lambda:us-east-1:111122223333:microvm-image:my-image",
    "microvmImageVersion": "1.0"
  },
  "requestID": "7aebcd0f-cda1-11e4-aaa2-e356da31e4ff",
  "eventID": "e92a3e85-8ecd-4d23-8074-843aabfe89bf",
  "readOnly": false,
  "resources": [
    {
      "accountId": "111122223333",
      "type": "AWS::Lambda::MicrovmImage",
      "ARN": "arn:aws:lambda:us-east-1:111122223333:microvm-image:my-image"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "111122223333",
  "eventCategory": "Data"
}
```

# Risoluzione dei problemi

Questa sezione descrive come eseguire il debug e risolvere i problemi più comuni quando si lavora con MicroVMS. AWS Lambda

## Accesso alla shell

Usa l'accesso alla shell per connetterti direttamente a una MicroVM in esecuzione per il debug e la risoluzione dei problemi.

È possibile connettersi a una shell MicroVM in due modi:

- Console: seleziona la tua MicroVM nella console Lambda e scegli Connetti.
- CLI: genera un token di shell con `create-microvm-shell-auth-token`, quindi utilizza il token per stabilire una connessione.

Genera un token di shell, quindi connettiti:

```
aws lambda-microvms create-microvm-shell-auth-token \  
  --microvm-identifier <id> --expiration-in-minutes 30  
# In Console: select MicroVM -> Connect  
# In shell: ctr task ls, then ctr task exec -t --exec-id shell <id> /bin/sh
```

La MicroVM deve essere stata eseguita con il connettore di `SHELL_INGRESS` rete (`arn:aws:lambda:us-east-1:aws:network-connector:aws-network-connector:SHELL_INGRESS`). Se la MicroVM non è stata avviata con questo connettore, restituisce un `create-microvm-shell-auth-token ValidationException`

Per altri problemi:

- Controlla il `terminationMessage` campo nella `get-microvm` risposta per MicroVMS terminato.
- Controlla i log di CloudWatch compilazione per eventuali problemi di creazione delle immagini.
- Controlla lo `FAILED` stato dei connettori di rete nel `StateReason` campo.

## Risoluzione dei problemi

Questa sezione fornisce soluzioni ai problemi più comuni quando si lavora con Lambda MicroVMS.

| Caratteristiche                                          | Possibili cause e risoluzione                                                                                                                                                                                                                                                                                |
|----------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| La creazione dell'immagine non riesce (CREATION_FAILED ) | Controlla i log di compilazione in <code>/aws/lambda/microvm/&lt;image-name&gt;</code> . Verifica la <code>Dockerfile</code> sintassi, le autorizzazioni Amazon S3 e la disponibilità delle immagini di base. Esegui <code>docker build</code> localmente per la riproduzione.                               |
| MicroVM bloccato PENDING                                 | Attendi e riprova. Se persistente, controlla lo stato del servizio. Verifica che la tua quota di concorrenza non sia esaurita.                                                                                                                                                                               |
| La domanda non risponde dopo il curriculum               | Implementa il <code>/resume</code> lifecycle hook per ristabilire le connessioni e convalidare lo stato. Verifica che l'app si colleghi alla porta 8080 (o alla porta configurata) dopo la ripresa.                                                                                                          |
| 502 Bad Gateway dall'endpoint                            | L'applicazione si è bloccata o non è in ascolto. Controlla i log di runtime. Verifica <code>EXPOSE</code> e <code>CMD</code> accedi <code>Dockerfile</code> . In caso di ripristino automatico, la MicroVM potrebbe non essere stata ripristinata (controlla lo stato utilizzando). <code>get-microvm</code> |
| 429 Troppe richieste                                     | Percentuale di richieste superata. Riprova con backoff e jitter esponenziali.                                                                                                                                                                                                                                |
| Caduta delle connessioni                                 | È stato attivato il timeout di inattività. Implementa i keepalive. ping/pong Oppure estendi la <code>maxIdleDurationSeconds</code> politica di inattività.                                                                                                                                                   |
| Latenza elevata sull'endpoint                            | Saturazione della larghezza di banda. Verifica se il traffico supera la capacità di larghezza di banda per le dimensioni della tua MicroVM. Scalabilità fino a dimensioni maggiori.                                                                                                                          |
| Token di autenticazione scaduto (403)                    | I token hanno una scadenza configurabile. Genera un nuovo token prima della scadenza di quello vecchio. Implementa la logica di aggiornamento dei token nel tuo client.                                                                                                                                      |
| L'uscita VPC non funziona                                | Verifica che il connettore di rete sia in ACTIVE stato. Verifica che le regole del gruppo di sicurezza consentano il traffico in uscita. Verifica che le sottoreti abbiano percorsi verso le risorse di destinazione.                                                                                        |

## Errori comuni (creazione di immagini)

| Errore                        | Causa                                                                                                | Soluzione                                                                            |
|-------------------------------|------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| S3_ACCESS_DENIED              | Il ruolo di compilazione non dispone delle autorizzazioni per recuperare gli artefatti di Amazon S3. | Aggiungi l' <code>s3:GetObject</code> autorizzazione per il tuo bucket di artefatti. |
| S3_NO_SUCH_KEY                | La chiave dell'artefatto non esiste nel bucket.                                                      | Verifica che il percorso di Amazon S3 sia corretto.                                  |
| S3_NO_SUCH_BUCKET             | Il bucket Amazon S3 non esiste.                                                                      | Controlla il nome del bucket e conferma che è stato creato.                          |
| S3_INVALID_OBJECT             | Artefatto in una classe di archiviazione Glacier o non accessibile direttamente.                     | Sposta l'artefatto nella classe di archiviazione Standard.                           |
| S3_CROSS_REGION_ACCESS_DENIED | L'artefatto si trova in una regione diversa dall'immagine MicroVM.                                   | Assicurati che l'artefatto si trovi nella stessa regione dell'immagine MicroVM.      |
| ARCHIVE_DOCKERFILE_NOT_FOUND  | All'archivio zip manca un <code>Dockerfile</code> nella directory principale.                        | Aggiungi a <code>Dockerfile</code> alla radice del tuo archivio zip.                 |
| ARCHIVE_INVALID               | Il file di archivio non è un file ZIP valido o è danneggiato.                                        | Re-create l'archivio zip e caricalo nuovamente.                                      |
| CONTAINER_BUILD_FAILED        | <code>Dockerfile</code> Istruzioni non valide, file mancanti o errori di sintassi.                   | Esegui il debug locale utilizzando <code>Dockerfile docker build</code>              |
| DISK_STORAGE_FULL             | MicroVM ha esaurito lo storage durante la compilazione.                                              | Riduci le dimensioni degli artefatti o contatta l'assistenza.                        |

| Errore                  | Causa                              | Soluzione                                                    |
|-------------------------|------------------------------------|--------------------------------------------------------------|
| INTERNAL_PLATFORM_ERROR | Si è verificato un errore interno. | Riprova l'operazione. Se persistente, contatta l'assistenza. |

## Risoluzione dei problemi relativi ai connettori di rete

| Codice di errore                 | Causa                                                                                                                                                                                                      | Soluzione                                                                                                                                                                                                     |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DisallowedByVpcEncryptionControl | Il VPC ha una politica di controllo della crittografia che impedisce il traffico o le interfacce di rete non crittografati. Lambda non può creare ENI che soddisfino i requisiti di crittografia.          | Aggiungi Lambda all'elenco di esclusione del controllo di crittografia VPC. Se l'esclusione non è possibile, utilizza un VPC o una sottorete a cui non siano applicati controlli di crittografia restrittivi. |
| Ec2RequestLimitExceeded          | Lambda effettua chiamate API EC2 (ad esempio <code>CreateNetworkInterface</code> , <code>DescribeSubnets</code> ) per configurare la connettività. Troppe chiamate API EC2 simultanee causano limitazioni. | Riprova l'operazione dopo un breve ritardo. Se persistente, riduci le operazioni simultanee dei connettori di rete o richiedi un aumento del limite di limitazione delle API EC2 tramite l'assistenza. AWS    |
| InsufficientRolePermissions      | Il ruolo dell'operatore non dispone delle autorizzazioni EC2 richieste.                                                                                                                                    | Assicurati che il ruolo IAM disponga delle autorizzazioni di rete EC2 necessarie.                                                                                                                             |
| InternalError                    | Si è verificato un errore imprevisto all'interno del servizio Lambda durante l'elaborazione della richiesta del connettore di rete.                                                                        | Riprova l'operazione. Se il problema persiste dopo più tentativi, contatta l'AWS assistenza indicando l'ARN del connettore di rete e il timestamp approssimativo.                                             |
| InvalidSecurityGroup             | L'ID del gruppo di sicurezza non esiste, è stato eliminato o non                                                                                                                                           | Verifica che tutti gli ID dei gruppi di sicurezza esistano e appartengano allo stesso VPC delle sottoreti                                                                                                     |

| Codice di errore                  | Causa                                                                                                                                                      | Soluzione                                                                                                                                                                                                                              |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                   | appartiene allo stesso VPC delle sottoreti specificate.                                                                                                    | . Usa per convalidare <code>aws ec2 describe-security-groups --group-ids &lt;sg-id&gt; .</code>                                                                                                                                        |
| <code>InvalidSubnet</code>        | L'ID della sottorete non esiste, è stato eliminato o appartiene a un VPC diverso da quello previsto.                                                       | Verifica che tutti gli ID di sottorete esistano e appartengano al VPC corretto. Usa per <code>aws ec2 describe-subnets --subnet-ids &lt;subnet-id&gt; convalidare.</code>                                                              |
| <code>SubnetOutOfAddresses</code> | Il blocco CIDR della sottorete è esaurito: tutti gli IP sono allocati a ENI, istanze e altre risorse, quindi Lambda non può creare un'interfaccia di rete. | Libera gli indirizzi IP rimuovendo quelli non utilizzati o utilizza una sottorete diversa con ENI/istanze capacità disponibile. Prendi in considerazione sottoreti più grandi (ad esempio, /24 o più grandi) per i connettori di rete. |

## Le migliori pratiche per Lambda MicroVMs

Segui queste best practice per creare applicazioni affidabili, convenienti e sicure con MicroVMs.

### Progettazione delle applicazioni

Progetta la tua applicazione in modo che funzioni bene con le istantanee e gli hook del ciclo di vita di MicroVM:

- Usa gli hook del ciclo di vita: implementa per l'inizializzazione post-esecuzione, per una pulizia accurata, `/run` per il ripristino della connessione e `/suspend` per lo svuotamento dei dati. `/resume` `/terminate`
- Gestisci il riutilizzo delle istantanee: genera UUID, segreti e valori casuali nell'hook, non durante la creazione dell'immagine. `/run` Usa `CSPRngs` per la generazione di tutti i numeri casuali.



- **Design for resume:** convalida tutte le connessioni di rete e lo stato della cache nell'hook. `/resume`  
AWS Le connessioni SDK in genere vengono ripristinate automaticamente, ma le connessioni personalizzate potrebbero non funzionare.

## Performance

Ottimizza le prestazioni di MicroVM con le seguenti pratiche:

- **Right-size la tua microVM:** scegli la dimensione microVM più piccola che soddisfi i requisiti di CPU, memoria e larghezza di banda. La larghezza di banda si ridimensiona in base alla memoria configurata.
- **Riduci al minimo le dimensioni delle istantanee:** mantieni ridotto l'ingombro dell'applicazione per ridurre i tempi di esecuzione e ripristino. Rimuovi le dipendenze in fase di compilazione e i file temporanei prima di acquisire l'istananea dell'immagine.
- **Utilizzo suspend/resume per carichi di lavoro sensibili alla latenza:** mantieni gli ambienti utilizzati di frequente in stato di sospensione per riprendere rapidamente l'esecuzione.

## Sicurezza

Segui queste pratiche per proteggere i tuoi carichi di lavoro MicroVM:

- **Ruota i token di autenticazione:** genera token di breve durata (15-30 minuti). Implementa la logica di aggiornamento dei token nel tuo client prima della scadenza.
- **Token di ambito per porte specifiche:** utilizza il `allowedPorts` parametro per limitare l'ambito dei token solo alle porte utilizzate dall'applicazione.
- **Usa l'uscita VPC per il traffico sensibile:** indirizza il traffico verso database e API interne tramite un connettore di uscita VPC anziché la rete Internet pubblica.
- **Applica IAM con privilegi minimi:** separa i ruoli di build (creazione dell'immagine) dai ruoli di esecuzione (runtime). Concedi le autorizzazioni minime richieste.

## Ottimizzazione dei costi

Riduci i costi con le seguenti strategie:

- Configura le policy di inattività: impostate `maxIdleDurationSeconds` per sospendere automaticamente le MicroVMS inattive. Le microVM sospese non comportano costi di elaborazione.
- Imposta il valore massimo `DurationInSeconds`: evita costi eccessivi impostando un orario di terminazione fisso su ogni microVM.
- Termina le microVM inutilizzate: non lasciare le microVMS in esecuzione a tempo indeterminato. Da utilizzare per terminare automaticamente dopo periodi di inattività `suspendedDurationSeconds` prolungati.
- Risorse relative ai tag: utilizza i tag sulle immagini MicroVM per l'allocazione dei costi e la creazione di report in Cost Explorer. AWS
- Right-size la tua MicroVMS: inizia con la linea di base più piccola che soddisfi le tue esigenze di prestazioni. È possibile raggiungere una linea di base quadruplicata durante i picchi di attività senza dover effettuare nuovamente il provisioning.
- Elimina le immagini inutilizzate: elimina le versioni delle immagini MicroVM che non ti servono più per ridurre i costi di storage.

## Monitoraggio

Monitora efficacemente il tuo MicroVMS:

- Rivedi `StateReason`: quando le microVMS si interrompono inaspettatamente, la `get-microvm` risposta include un motivo dell'errore imprevisto. `stateReason`

## Integrazioni

Questa sezione descrive come integrare AWS Lambda MicroVMS con altri servizi e framework. Per informazioni dettagliate sull'utilizzo di MicroVMS con gli agenti gestiti da Claude, consulta [the section called “Agenti gestiti Claude”](#)

## Utilizzo di Lambda MicroVMS come sandbox per gli agenti gestiti Claude

AWS Lambda MicroVMS è un provider di sandbox gestito in sandbox self-hosted per Claude Managed Agent, che conserva file, pacchetti e servizi sensibili nell'infrastruttura controllata dall'utente. Anthropic ospita il loop degli agenti e il modello Claude, mentre Lambda MicroVM è il luogo in cui vengono eseguite le chiamate agli strumenti. Con questo modello, controlla l'ambiente

di esecuzione: cosa è installato, quale accesso alla rete è disponibile e quali risorse l'agente può raggiungere.

Ogni MicroVM è una macchina Firecracker-isolated virtuale con avvio in meno di un secondo basato su istantanee, funziona per un massimo di 8 ore e può essere terminata al termine della sessione. Le sessioni non condividono mai lo stato. Ottieni i limiti di sicurezza di una VM con il modello operativo serverless: nessun cluster da gestire, nessuna capacità inattiva a pagamento. Puoi orchestrare MicroVMS con AWS servizi per implementare le specifiche sandbox self-hosted come descritto di seguito. Per maggiori dettagli, consulta l'esempio di [riferimento Claude Managed Agents Self-Hosted Sandbox on Lambda MicroVMS](#).

## Come funziona

Un piano di controllo avvia una microVM per sessione di Claude:

1. Una sessione raggiunge lo stato di esecuzione e Anthropic invia un `session.status_run_started` webhook a un endpoint del tuo account.
2. Una funzione Lambda di avvio verifica la firma del webhook, quindi chiama. `RunMicroVM`
3. Il codice sulla microVM rivendica la sessione, esegue le chiamate agli strumenti (`bash`, `read`, `write`, `edit`, `glob`, `grep`) e invia i risultati ad Anthropic. `/workspace`
4. La MicroVM viene sospesa o terminata al termine della sessione.

La chiave API della tua organizzazione Anthropic non raggiunge mai l'elaborazione. AWS Il launcher passa solo un riferimento di AWS Secrets Manager alla chiave dell'ambiente Anthropic; il ruolo di esecuzione di MicroVM consente al codice di leggerlo in fase di esecuzione.

## Proprietà chiave

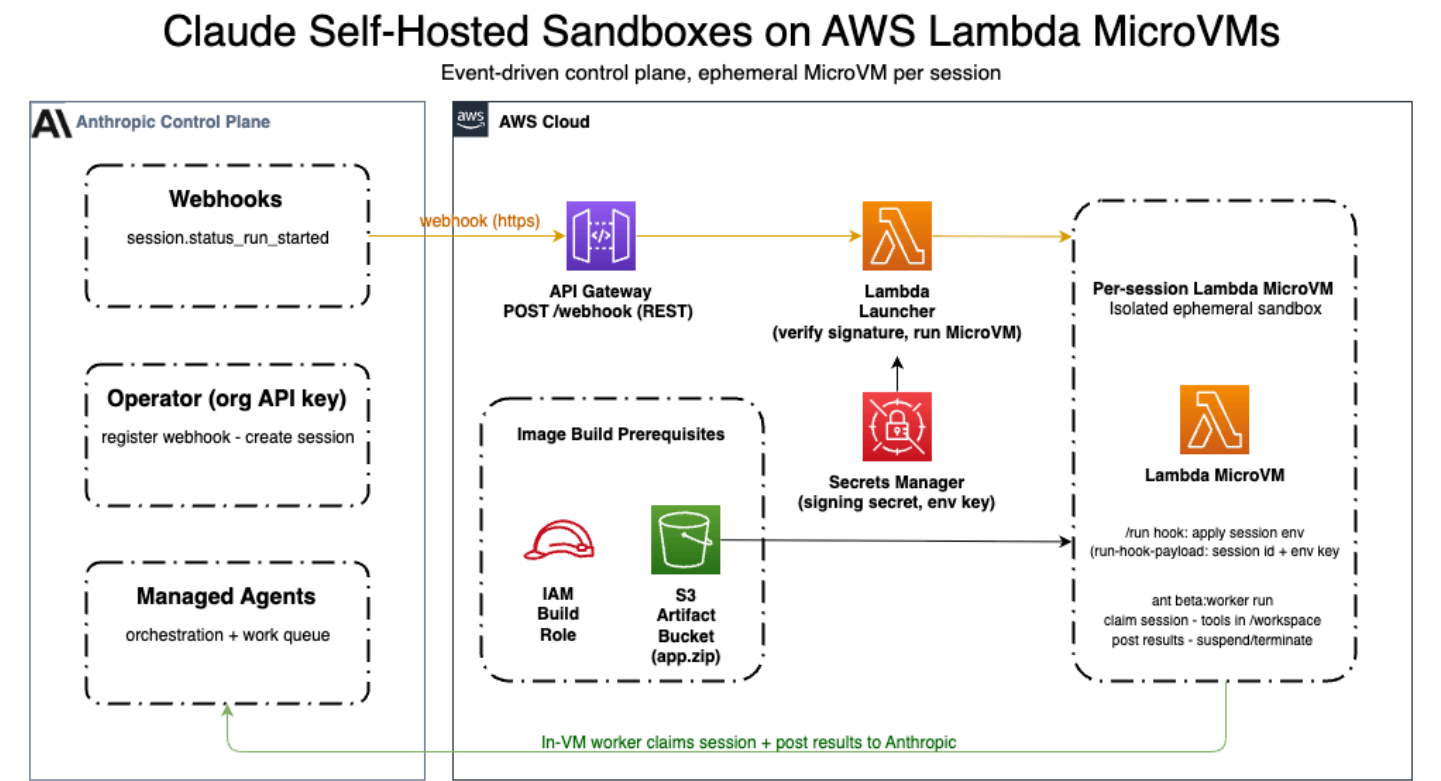
| Proprietà              | Vantaggio                                                                      |
|------------------------|--------------------------------------------------------------------------------|
| Isolamento con petardi | Hardware-virtualized limite per sessione                                       |
| Snapshot-based avvio   | Riprendi dall'istantanea di Firecracker in pochi secondi o secondi a una cifra |
| IAM tramite IMDSv2     | Utilizza credenziali a breve termine con privilegi minimi                      |

| Proprietà       | Vantaggio                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------|
| Durata statica  | Può funzionare fino a 8 ore con accesso completo al disco e alla memoria                   |
| Pay-per-session | Le microVM vengono terminate al termine delle sessioni, il che pone fine alla fatturazione |

Prerequisiti

- Un AWS account con autorizzazioni per Amazon Simple Storage Service (Amazon S3), AWS Identity and Access Management (IAM), Secrets Manager, AWS WAF, Lambda e Lambda MicroVMS
- Un agente Claude Managed Agents con un ambiente self\_hosted
- Una chiave di ambiente e una firma segreta del webhook della console Claude

Implementazione dell'implementazione di riferimento



Le Self-Hosted sandbox [Claude Managed Agents sul repository Lambda](#) MicroVMS forniscono un'implementazione minima e funzionante. e include:

- Uno CloudFormation stack (Amazon API Gateway, funzione di avvio Lambda, segreti di Secrets Manager, bucket Amazon S3, ruoli IAM)
- Un'immagine MicroVM (,, lifecycle hooks) Dockerfile Node.js EnvironmentWorker
- Uno script di distribuzione e uno script di verifica

Fasi di distribuzione:

1. Distribuisci lo stack. CloudFormation
2. Memorizza la chiave di ambiente e il segreto di firma nei segreti di Secrets Manager creati.
3. Crea l'immagine MicroVM.
4. Registra l'URL del webhook dello stack nella console Claude.
5. Verifica creando una sessione.

Per istruzioni dettagliate, consulta il repository README.

## Rete

Le microVM Lambda hanno un accesso pubblico a Internet per impostazione predefinita: non è necessaria alcuna configurazione per raggiungerle. `api.anthropic.com`

Per accedere a risorse private come un database Amazon Aurora o un ElastiCache cluster Amazon o per applicare le tue restrizioni di rete, collega un connettore di uscita VPC al momento dell'avvio. Per informazioni, consulta [Utilizzo dei connettori di rete in uscita](#).

## Policy di inattività

Imposta `suspendedDurationSeconds: 0` e `autoResumeEnabled: false` per MicroVMS per sessione. Imposta `maximumDurationInSeconds` come limite massimo per le sessioni bloccate (max 28.800 s).

```
{
  "maxIdleDurationSeconds": 120,
  "suspendedDurationSeconds": 0,
  "autoResumeEnabled": false
}
```

```
}

```

## Monitoraggio

Registri delle applicazioni: rivedi i log delle applicazioni nei CloudWatch registri:

```
aws logs tail /aws/lambda-microvms/claude-worker --follow

```

MicroVMS in esecuzione: elenca i MicroVMS in esecuzione nel tuo account:

```
aws lambda-microvms list-microvms \
  --image-identifier claude-worker \
  --query 'items[][microvmId,state,startedAt]' --output table

```

## Risoluzione dei problemi

| Caratteristiche                            | Causa                                                                                                                                                    |
|--------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Webhook restituisce 401                    | Firma segreta, mancata corrispondenza o consegna scaduta                                                                                                 |
| Nessun avvio di MicroVM                    | Webhook non registrato o autorizzazione per il ruolo di <code>session.status_run_started</code> esecuzione della funzione Lambda del launcher RunMicroVM |
| MicroVM si interrompe immediatamente       | <code>/runtimeout</code> del gancio: aumento <code>runTimeoutInSeconds</code>                                                                            |
| Il lavoratore esce immediatamente          | Da HTTPS in uscita a bloccato <code>api.anthropic.com</code>                                                                                             |
| La creazione dell'immagine non riesce S3_* | Errore di configurazione del ruolo di build o del bucket                                                                                                 |

## Utilizzo di Lambda MicroVMS come sandbox per gli agenti Cursor Cloud

Puoi utilizzare AWS Lambda MicroVMS come macchine ospitate autonomamente per i tuoi Cursor Cloud Agent, mantenendo i repository, l'esecuzione degli strumenti e l'accesso alla rete nell'infrastruttura che controlli. Cursor ospita il ciclo e il modello degli agenti; Lambda MicroVM è il

luogo in cui il lavoratore raccoglie le richieste del pool ed esegue le chiamate agli strumenti. Con questo schema, controlli l'ambiente di esecuzione: cosa è installato nell'immagine, quale accesso alla rete è disponibile e quali AWS risorse può raggiungere il ruolo IAM del lavoratore.

Ogni MicroVM è una macchina Firecracker-isolated virtuale con avvio basato su istantanee, funziona per un massimo di 8 ore e viene terminata al termine della sessione. Le sessioni non condividono mai lo stato. Ottieni i limiti di sicurezza di una VM con il modello operativo serverless: nessun cluster da gestire, nessuna capacità di pool inattiva a pagamento. Puoi orchestrare MicroVMS con una funzione Lambda con controller pianificato che risponde alle richieste di pool in sospeso. Per maggiori dettagli, consulta il [modello di riferimento Run self-hosted machines for Cursor Cloud Agents on Lambda MicroVMS. AWS](#)

### Come funziona

Un controller pianificato avvia una microVM per ogni richiesta di pool in sospeso:

1. Si avvia un agente cloud dal cursore. [com/agents](#)contro una macchina ospitata autonomamente. La richiesta rimane in sospeso fino a quando un lavoratore non la richiede.
2. La funzione Lambda del controller viene eseguita `agent worker controller --spawn ./spawn.sh` per una finestra di sondaggio Server-Sent Events (SSE) di cinque minuti. Quando vede una richiesta corrispondente, viene eseguita. `spawn.sh`
3. `spawn.sh`chiama `RunMicrovm (aws lambda-microvms run-microvm)` e ritorna senza attendere. Inoltra l'`CURSOR_*`ambiente del reclamo all'ospite tramite `--run-hook-payload`.
4. L'`/runhook microVM (hook.py)` applica quel payload e si avvia `entrypoint.sh`, il quale viene eseguito. `cursor-agent worker --pool ... start` Il lavoratore esegue le chiamate agli strumenti nel tuo account.
5. Quando la sessione è inattiva dopo il timeout di rilascio, il worker lo rilascia e la MicroVM termina.

La chiave API dell'account di servizio Cursor è archiviata in Systems Manager Parameter Store come. `AWS SecureString` Il controller e la microVM la leggono in fase di esecuzione `ssm:GetParameter`; la chiave non viene mai inserita nell'immagine.

### Proprietà chiave

| Proprietà              | Vantaggio                                |
|------------------------|------------------------------------------|
| Isolamento con petardi | Hardware-virtualized limite per sessione |

| Proprietà                          | Vantaggio                                                                                   |
|------------------------------------|---------------------------------------------------------------------------------------------|
| Avvio con istantanea               | Lambda ripristina l'ospite da un'istantanea Firecracker della tua immagine                  |
| IAM tramite il ruolo di esecuzione | L'ospite utilizza credenziali di breve durata da <code>MicroVmExecutionRoleArn</code>       |
| Durata statica                     | Ogni microVM può funzionare fino a 8 ore () --maximum-duration-in-seconds 28800             |
| Pay-per-session                    | I costi sono calcolati per il runtime di una microVM, non per la capacità inattiva del pool |

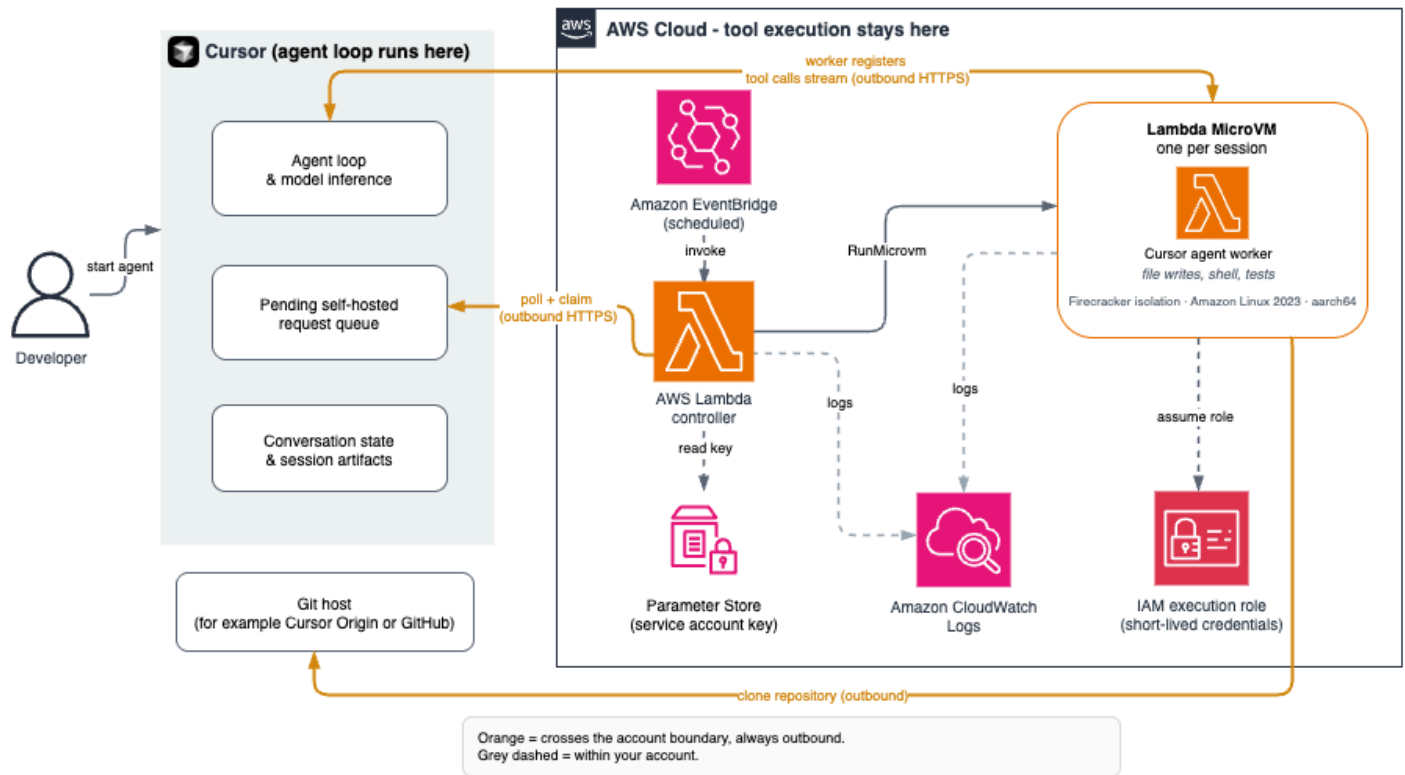
## Prerequisiti

- Un AWS account con Lambda MicroVMS abilitato, oltre all'autorizzazione per utilizzare Amazon S3, IAM e Systems Manager Parameter Store CloudFormation
- Un account Cursor Enterprise
- Un team Cursor con macchine ospitate autonomamente abilitate
- Una chiave API dell'account di [servizio](#) per i lavoratori del pool
- La [AWS CLI](#) aggiornata all'ultima versione stabile e Docker (lo script di distribuzione crea l'immagine del controller)



## Implementazione dell'implementazione di riferimento

## Cursor Self-hosted machines on AWS Lambda MicroVMs



Il repository [anysphere/aws-lambda-workers](https://github.com/anysphere/aws-lambda-workers) fornisce una distribuzione minima e funzionante. e include:

- Uno CloudFormation stack (cloudformation.yaml): una funzione Lambda del controller pianificato, una EventBridge rate(1 minute) regola, un bucket di artefatti Amazon S3, una coda di lettere morte e i ruoli IAM (,,,) MicroVmExecutionRole BuildRole SpawnRole ControllerRole
- Un'immagine del controller (,) che esegue una finestra di polling SSE per controller/ Dockerfile controller/handler.py richiamo
- Un'immagine MicroVM (:,,) microvm-image/ Dockerfile hook.py entrypoint.sh
- Uno spawn hook (spawn.sh) e uno script di distribuzione () deploy.sh

Fasi di distribuzione:

1. Archivia la chiave API dell'account di servizio in SSM.

```
aws ssm put-parameter --type SecureString \
  --name /cursor-lambda-workers/cursor-api-key \
  --value "YOUR_SERVICE_ACCOUNT_KEY"
```

2. Distribuisce lo stack. `deploy.sh` crea l'immagine del controller ed esegue `aws cloudformation deploy`

```
POOL_NAMES=default ./deploy.sh
# POOL_NAMES=gpu,default ./deploy.sh
# ALL_POOLS=true ./deploy.sh
```

3. Crea l'immagine MicroVM **microvm-image/** utilizzando gli output dello stack. Abilita gli hook `ready` and `validate image` e l'hook `run MicroVM` in modo che vengano catturati nel percorso dell'istantanea.

```
BUCKET=$(aws cloudformation describe-stacks --stack-name cursor-lambda-workers \
  --query "Stacks[0].Outputs[?OutputKey=='ArtifactBucketName'].OutputValue" --output
  text)
BUILD_ROLE=$(aws cloudformation describe-stacks --stack-name cursor-lambda-workers \
  --query "Stacks[0].Outputs[?OutputKey=='BuildRoleArn'].OutputValue" --output text)
BASE=$(aws lambda-microvms list-managed-microvm-images --query "items[0].imageArn" --
  output text)
( cd microvm-image && zip -r /tmp/app.zip . )
aws s3 cp /tmp/app.zip "s3://${BUCKET}/app.zip"
aws lambda-microvms create-microvm-image \
  --code-artifact "uri=s3://${BUCKET}/app.zip" \
  --name cursor-pool-worker \
  --base-image-arn "${BASE}" \
  --build-role-arn "${BUILD_ROLE}" \
  --environment-variables "POOL_NAME=default,CURSOR_API_KEY_PARAM_NAME=/cursor-
  lambda-workers/cursor-api-key" \
  --hooks '{"port":9000,"microvmImageHooks":
  {"ready":"ENABLED","readyTimeoutInSeconds":60,"validate":"ENABLED","validateTimeoutInSeconds":
  {"run":"ENABLED","runTimeoutInSeconds":60}}'
```

4. Avvia un agente cloud dal cursore. [com/agents](https://github.com/anysphere/aws-lambda-workers/blob/main/README.md) contro il pool (o contro un repository in modalità `repo-bound`) per la verifica.

Per istruzioni dettagliate, consulta il repository README. <https://github.com/anysphere/aws-lambda-workers/blob/main/README.md>

## Modalità pool e repo

Mantieni lo stack PoolNames e l'immagine POOL\_NAME allineati in modo che controller e guest servano lo stesso pool.

- Any-repo mode — Il routing avviene per nome del pool, non per git remote. L'ospite avvia il lavoratore da uno spazio di lavoro senza git remote, quindi il lavoratore `repo=` omette le etichette. Gli utenti scelgono il gruppo di repository Any e il nome del pool quando avviano un agente.
- Repo-bound modalità: il lavoratore serve uno o più telecomandi git specifici. O inserisce il clone nell'immagine MicroVM o impostalo in `CURSOR_REPO_URL` modo da `entrypoint.sh` clonarlo all'avvio. Il lavoratore ricava l'`repo=`etichetta dal telecomando git. I repository privati richiedono le credenziali git sul lavoratore.

## Rete

Il riferimento `spawn.sh` collega il `INTERNET_EGRESS` connettore gestito in modo che il lavoratore possa raggiungere il piano di controllo di Cursor e `ALL_INGRESS` gli hook delle immagini. Outbound-only i lavoratori non ricevono mai HTTPS in entrata dopo l'avvio.

Per accedere a risorse private (ad esempio un database Amazon Aurora o un ElastiCache cluster Amazon) o per applicare le tue restrizioni, collega un connettore di uscita VPC al momento dell'avvio al posto di. `INTERNET_EGRESS` Consulta [the section called “Utilizzo dei connettori di rete in uscita”](#).

## Monitoraggio

Registri degli ospiti: i log di MicroVM vanno a Logs: CloudWatch

```
aws logs tail /aws/lambda/microvms/cursor-pool-worker --follow
```

Log del controller: la funzione Lambda del controller accede a. `/aws/lambda/cursor-lambda-workers-controller`

MicroVMS in esecuzione: elenca i MicroVMS in esecuzione per l'immagine:

```
aws lambda-microvms list-microvms --image-identifier cursor-pool-worker
```

## Risoluzione dei problemi

| Caratteristiche                                   | Causa                                                                                                                                                                                                                      |
|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| La creazione dell'immagine non riesce (S3 o IAM)  | Verifica gli output dello stack <code>ArtifactBucketName</code> e <code>BuildRoleArn</code> ; lo zip deve atterrare in quel bucket e il ruolo di compilazione deve essere in grado di leggerlo                             |
| Nessun avvio di MicroVM                           | Il controller non è in esecuzione o non è in grado di chiamare <code>RunMicrovm</code> ; conferma che l' <code>cursor-pool-worker</code> immagine esiste e (per un controller locale) si presume <code>SpawnRoleArn</code> |
| Il lavoratore esce immediatamente                 | L'ospite è mancante <code>CURSOR_API_KEY</code> (SSM/ <code>cursor-lambda-workers/cursor-api-key</code> ) o l' <code>/runhook</code> non è stato avviato <code>cursor-agent worker ... start</code>                        |
| <code>Exec format error - node</code>             | L'immagine ha installato l'architettura CLI sbagliata; ecco le <code>microVM aarch64 () arm64</code>                                                                                                                       |
| Service-account chiave segnalata non valida       | <code>CURSOR_API_ENDPOINT</code> / <code>CURSOR_API_URL</code> sono stati inoltrati all'ospite; questi devono rimanere non impostati, quindi <code>worker start</code> utilizza l'host di autenticazione predefinito       |
| Immagine creata prima dell'attivazione degli hook | Ricostruisci l'immagine MicroVM in questo modo e gli <code>run hook ready</code> si validate trovino nel percorso dell'istantanea                                                                                          |

## Risorse correlate

- [AWS Lambda MicroVMS](#)
- [RunMicrovmRiferimento CLI](#)
- [Macchine ospitate autonomamente da Cursor Quickstart](#)
- Modello di riferimento: `-lambda-workers` [anysphere/aws](#)

# Usare Lambda con un AWS SDK

AWS I kit di sviluppo software (SDK) sono disponibili per molti linguaggi di programmazione diffusi. Ogni SDK fornisce un'API, esempi di codice e documentazione che facilitano agli sviluppatori la creazione di applicazioni nel loro linguaggio preferito.

| Documentazione sugli SDK                     | Esempi di codice                                              |
|----------------------------------------------|---------------------------------------------------------------|
| <a href="#">AWS SDK per C++</a>              | <a href="#">AWS SDK per C++ esempi di codice</a>              |
| <a href="#">AWS CLI</a>                      | <a href="#">AWS CLI esempi di codice</a>                      |
| <a href="#">AWS SDK per Go</a>               | <a href="#">AWS SDK per Go esempi di codice</a>               |
| <a href="#">AWS SDK per Java</a>             | <a href="#">AWS SDK per Java esempi di codice</a>             |
| <a href="#">AWS SDK per JavaScript</a>       | <a href="#">AWS SDK per JavaScript esempi di codice</a>       |
| <a href="#">AWS SDK per Kotlin</a>           | <a href="#">AWS SDK per Kotlin esempi di codice</a>           |
| <a href="#">AWS SDK per .NET</a>             | <a href="#">AWS SDK per .NET esempi di codice</a>             |
| <a href="#">AWS SDK per PHP</a>              | <a href="#">AWS SDK per PHP esempi di codice</a>              |
| <a href="#">Strumenti AWS per PowerShell</a> | <a href="#">Strumenti AWS per PowerShell esempi di codice</a> |
| <a href="#">AWS SDK per Python (Boto3)</a>   | <a href="#">AWS SDK per Python (Boto3) esempi di codice</a>   |
| <a href="#">AWS SDK per Ruby</a>             | <a href="#">AWS SDK per Ruby esempi di codice</a>             |
| <a href="#">AWS SDK per Rust</a>             | <a href="#">AWS SDK per Rust esempi di codice</a>             |
| <a href="#">AWS SDK per SAP ABAP</a>         | <a href="#">AWS SDK per SAP ABAP esempi di codice</a>         |
| <a href="#">AWS SDK per Swift</a>            | <a href="#">AWS SDK per Swift esempi di codice</a>            |

Per esempi specifici relativi a Lambda, consulta [Esempi di codice per l'utilizzo di Lambda AWS SDK](#).

 **Esempio di disponibilità**

Non riesci a trovare quello che ti serve? Richiedi un esempio di codice utilizzando il link  
Fornisci un feedback nella parte inferiore di questa pagina.

# Esempi di codice per l'utilizzo di Lambda AWS SDK

I seguenti esempi di codice mostrano come usare Lambda con un kit di sviluppo AWS software (SDK).

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

AWS i contributi della community sono esempi che sono stati creati e gestiti da più team. AWS Per fornire un feedback, utilizza il meccanismo disponibile nei repository collegati.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Esempi di codice

- [Esempi di base per l'utilizzo di Lambda AWS SDK](#)
  - [Hello Lambda](#)
  - [Impara le basi di Lambda con un AWS SDK](#)
  - [Azioni per l'utilizzo di Lambda AWS SDK](#)
    - [Utilizzare CreateAlias con una CLI](#)
    - [Utilizzare CreateFunction con un AWS SDK o CLI](#)
    - [Utilizzare DeleteAlias con una CLI](#)
    - [Utilizzare DeleteFunction con un AWS SDK o CLI](#)
    - [Utilizzare DeleteFunctionConcurrency con una CLI](#)
    - [Utilizzare DeleteProvisionedConcurrencyConfig con una CLI](#)
    - [Utilizzare GetAccountSettings con una CLI](#)
    - [Utilizzare GetAlias con una CLI](#)
    - [Utilizzare GetFunction con un AWS SDK o CLI](#)

- [Utilizzare GetFunctionConcurrency con una CLI](#)
- [Utilizzare GetFunctionConfiguration con una CLI](#)
- [Utilizzare GetPolicy con una CLI](#)
- [Utilizzare GetProvisionedConcurrencyConfig con una CLI](#)
- [Usa Invoke con un AWS SDK o CLI](#)
- [Utilizzare ListFunctions con un AWS SDK o CLI](#)
- [Utilizzare ListProvisionedConcurrencyConfigs con una CLI](#)
- [Utilizzare ListTags con una CLI](#)
- [Utilizzare ListVersionsByFunction con una CLI](#)
- [Utilizzare PublishVersion con una CLI](#)
- [Utilizzare PutFunctionConcurrency con una CLI](#)
- [Utilizzare PutProvisionedConcurrencyConfig con una CLI](#)
- [Utilizzare RemovePermission con una CLI](#)
- [Utilizzare TagResource con una CLI](#)
- [Utilizzare UntagResource con una CLI](#)
- [Utilizzare UpdateAlias con una CLI](#)
- [Utilizzare UpdateFunctionCode con un AWS SDK o CLI](#)
- [Utilizzare UpdateFunctionConfiguration con un AWS SDK o CLI](#)
- [Scenari per l'utilizzo di Lambda AWS SDK](#)
  - [Conferma automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda utilizzando un AWS SDK](#)
  - [Esegui automaticamente la migrazione di utenti Amazon Cognito noti con una funzione Lambda utilizzando un AWS SDK](#)
  - [Crea un'API API Gateway REST per tenere traccia COVID-19 dei dati](#)
  - [Creazione di una REST API per la libreria di prestiti](#)
  - [Creazione di un'applicazione di messaggistica con Step Functions](#)
  - [Crea un'API REST con integrazione di funzioni proxy](#)
  - [Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette](#)
  - [Creazione di un'applicazione di chat websocket con API Gateway](#)
  - [Crea un'applicazione che analizza il feedback dei clienti e sintetizza l'audio](#)



- [Creazione di una dashboard di monitoraggio con il nome della funzione come variabile](#)
- [Creazione della tua prima funzione serverless](#)
- [Richiamo a una funzione Lambda da un browser](#)
- [Trasformare i dati per l'applicazione con S3 Object Lambda](#)
- [Utilizzo di Gateway API per invocare una funzione Lambda](#)
- [Utilizzo di Step Functions per richiamare le funzioni Lambda](#)
- [Utilizzo degli eventi pianificati per invocare una funzione Lambda](#)
- [Utilizzare l'API Amazon Neptune per sviluppare una funzione Lambda che esegua query sui dati del grafico](#)
- [Utilizzo delle variabili di proprietà nei dashboard di monitoraggio per monitorare più funzioni serverless](#)
- [Scrivi dati di attività personalizzati con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito utilizzando un AWS SDK](#)
- [Esempi serverless per Lambda](#)
  - [Connessione a un database Amazon RDS in una funzione Lambda](#)
  - [Invocare una funzione Lambda da un trigger Kinesis](#)
  - [Richiamare una funzione Lambda da un trigger DynamoDB](#)
  - [Richiamare una funzione Lambda da un trigger Amazon DocumentDB](#)
  - [Invocare una funzione Lambda da un trigger Amazon MSK](#)
  - [Invocazione di una funzione Lambda da un trigger Amazon S3](#)
  - [Invocazione di una funzione Lambda da un trigger Amazon SNS](#)
  - [Richiamo di una funzione Lambda da un trigger Amazon SQS](#)
  - [Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis](#)
  - [Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB](#)
  - [Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS](#)
- [AWS contributi della community per Lambda](#)
  - [Creare e testare un'applicazione serverless](#)

# Esempi di base per l'utilizzo di Lambda AWS SDK

I seguenti esempi di codice mostrano come utilizzare le nozioni di base di with SDK. AWS Lambda AWS

## Esempi

- [Hello Lambda](#)
- [Impara le basi di Lambda con un AWS SDK](#)
- [Azioni per l'utilizzo di Lambda AWS SDK](#)
  - [Utilizzare CreateAlias con una CLI](#)
  - [Utilizzare CreateFunction con un AWS SDK o CLI](#)
  - [Utilizzare DeleteAlias con una CLI](#)
  - [Utilizzare DeleteFunction con un AWS SDK o CLI](#)
  - [Utilizzare DeleteFunctionConcurrency con una CLI](#)
  - [Utilizzare DeleteProvisionedConcurrencyConfig con una CLI](#)
  - [Utilizzare GetAccountSettings con una CLI](#)
  - [Utilizzare GetAlias con una CLI](#)
  - [Utilizzare GetFunction con un AWS SDK o CLI](#)
  - [Utilizzare GetFunctionConcurrency con una CLI](#)
  - [Utilizzare GetFunctionConfiguration con una CLI](#)
  - [Utilizzare GetPolicy con una CLI](#)
  - [Utilizzare GetProvisionedConcurrencyConfig con una CLI](#)
  - [Usa Invoke con un AWS SDK o CLI](#)
  - [Utilizzare ListFunctions con un AWS SDK o CLI](#)
  - [Utilizzare ListProvisionedConcurrencyConfigs con una CLI](#)
  - [Utilizzare ListTags con una CLI](#)
  - [Utilizzare ListVersionsByFunction con una CLI](#)
  - [Utilizzare PublishVersion con una CLI](#)
  - [Utilizzare PutFunctionConcurrency con una CLI](#)
  - [Utilizzare PutProvisionedConcurrencyConfig con una CLI](#)
  - [Utilizzare RemovePermission con una CLI](#)
  - [Utilizzare TagResource con una CLI](#)

- [Utilizzare UntagResource con una CLI](#)
- [Utilizzare UpdateAlias con una CLI](#)
- [Utilizzare UpdateFunctionCodecon un AWS SDK o CLI](#)
- [Utilizzare UpdateFunctionConfigurationcon un AWS SDK o CLI](#)

## Hello Lambda

L'esempio di codice seguente mostra come iniziare a utilizzare Lambda.

.NET

SDK per .NET

### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
namespace LambdaActions;

using Amazon.Lambda;

public class HelloLambda
{
    static async Task Main(string[] args)
    {
        var lambdaClient = new AmazonLambdaClient();

        Console.WriteLine("Hello AWS Lambda");
        Console.WriteLine("Let's get started with AWS Lambda by listing your existing Lambda functions:");

        var response = await lambdaClient.ListFunctionsAsync();
        response.Functions.ForEach(function =>
        {
            Console.WriteLine($"{function.FunctionName}\t{function.Description}");
        });
    }
}
```

```
}
```

- Per i dettagli sull'API, consulta la [ListFunctions](#) sezione AWS SDK per .NET API Reference.

## C++

### SDK per C++

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Codice per il file CMakeLists.txt CMake.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS lambda)

# Set this project's name.
project("hello_lambda")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
  libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
      "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
  endif ()

# Find the AWS SDK for C++ package.
```

```

find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
    for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
    may need to uncomment this

                                # and set the proper subdirectory to the
    executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
        ${CMAKE_CURRENT_BINARY_DIR}/${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
    hello_lambda.cpp)

target_link_libraries(${PROJECT_NAME}
    ${AWSSDK_LINK_LIBRARIES})

```

Codice per il file sorgente `hello_lambda.cpp`.

```

#include <aws/core/Aws.h>
#include <aws/lambda/LambdaClient.h>
#include <aws/lambda/model/ListFunctionsRequest.h>
#include <iostream>

/*
 * A "Hello Lambda" starter application which initializes an AWS Lambda (Lambda)
 * client and lists the Lambda functions.
 *
 * main function
 *
 * Usage: 'hello_lambda'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
    // options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;

```

```

    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::Lambda::LambdaClient lambdaClient(clientConfig);
        std::vector<Aws::String> functions;
        Aws::String marker; // Used for pagination.

        do {
            Aws::Lambda::Model::ListFunctionsRequest request;
            if (!marker.empty()) {
                request.SetMarker(marker);
            }

            Aws::Lambda::Model::ListFunctionsOutcome outcome =
lambdaClient.ListFunctions(
                request);

            if (outcome.IsSuccess()) {
                const Aws::Lambda::Model::ListFunctionsResult
&listFunctionsResult = outcome.GetResult();
                std::cout << listFunctionsResult.GetFunctions().size()
                    << " lambda functions were retrieved." << std::endl;

                for (const Aws::Lambda::Model::FunctionConfiguration
&functionConfiguration: listFunctionsResult.GetFunctions()) {
                    functions.push_back(functionConfiguration.GetFunctionName());
                    std::cout << functions.size() << " "
                        << functionConfiguration.GetDescription() <<
std::endl;

                    std::cout << "    "
                        <<
Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
                            functionConfiguration.GetRuntime()) << ": "
                        << functionConfiguration.GetHandler()
                        << std::endl;
                }
                marker = listFunctionsResult.GetNextMarker();
            } else {
                std::cerr << "Error with Lambda::ListFunctions. "
                    << outcome.GetError().GetMessage()

```

```
                << std::endl;
                result = 1;
                break;
            }
        } while (!marker.empty());
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- Per i dettagli sull'API, vedi [ListFunctions](#) in AWS SDK per C++ API Reference.

Go

SDK per Go V2

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/lambda"
)

// main uses the AWS SDK for Go (v2) to create an AWS Lambda client and list up
// to 10
// functions in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
```

```
func main() {
    ctx := context.Background()
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        fmt.Println("Couldn't load default configuration. Have you set up your AWS
account?")
        fmt.Println(err)
        return
    }
    lambdaClient := lambda.NewFromConfig(sdkConfig)

    maxItems := 10
    fmt.Printf("Let's list up to %v functions for your account.\n", maxItems)
    result, err := lambdaClient.ListFunctions(ctx, &lambda.ListFunctionsInput{
        MaxItems: aws.Int32(int32(maxItems)),
    })
    if err != nil {
        fmt.Printf("Couldn't list functions for your account. Here's why: %v\n", err)
        return
    }
    if len(result.Functions) == 0 {
        fmt.Println("You don't have any functions!")
    } else {
        for _, function := range result.Functions {
            fmt.Printf("\t\t%v\n", *function.FunctionName)
        }
    }
}
```

- Per i dettagli sull'API, consulta la [ListFunctions](#) sezione AWS SDK per Go API Reference.

## Java

### SDK per Java 2.x

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).



```
/**
 * Lists the AWS Lambda functions associated with the current AWS account.
 *
 * @param awsLambda an instance of the {@link LambdaClient} class, which is
 * used to interact with the AWS Lambda service
 *
 * @throws LambdaException if an error occurs while interacting with the AWS
 * Lambda service
 */
public static void listFunctions(LambdaClient awsLambda) {
    try {
        ListFunctionsResponse functionResult = awsLambda.listFunctions();
        List<FunctionConfiguration> list = functionResult.functions();
        for (FunctionConfiguration config : list) {
            System.out.println("The function name is " +
config.functionName());
        }

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- Per i dettagli sull'API, consulta la [ListFunctions](#) sezione AWS SDK for Java 2.x API Reference.

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});
```

```
export const helloLambda = async () => {
  const paginator = paginateListFunctions({ client }, {});
  const functions = [];

  for await (const page of paginator) {
    const funcNames = page.Functions.map((f) => f.FunctionName);
    functions.push(...funcNames);
  }

  console.log("Functions:");
  console.log(functions.join("\n"));
  return functions;
};
```

- Per i dettagli sull'API, consulta la [ListFunctions](#) sezione AWS SDK per JavaScript API Reference.

## Python

### SDK per Python (Boto3)

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
import boto3

def main():
    """
    List the Lambda functions in your AWS account.
    """
    # Create the Lambda client
    lambda_client = boto3.client("lambda")

    # Use the paginator to list the functions
```

```
paginator = lambda_client.get_paginator("list_functions")
response_iterator = paginator.paginate()

print("Here are the Lambda functions in your account:")
for page in response_iterator:
    for function in page["Functions"]:
        print(f"  {function['FunctionName']}")

if __name__ == "__main__":
    main()
```

- Per i dettagli sull'API, consulta [ListFunctions AWS SDK for Python \(Boto3\) API Reference](#).

## Ruby

### SDK per Ruby

#### Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
require 'aws-sdk-lambda'

# Creates an AWS Lambda client using the default credentials and configuration
def lambda_client
  Aws::Lambda::Client.new
end

# Lists the Lambda functions in your AWS account, paginating the results if
# necessary
def list_lambda_functions
  lambda = lambda_client

  # Use a pagination iterator to list all functions
  functions = []
  lambda.list_functions.each_page do |page|
```

```
functions.concat(page.functions)
end

# Print the name and ARN of each function
functions.each do |function|
  puts "Function name: #{function.function_name}"
  puts "Function ARN: #{function.function_arn}"
  puts
end

puts "Total functions: #{functions.count}"
end

list_lambda_functions if __FILE__ == $PROGRAM_NAME
```

- Per i dettagli sull'API, consulta la [ListFunctions](#) sezione AWS SDK per Ruby API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Impara le basi di Lambda con un AWS SDK

Gli esempi di codice seguenti mostrano come:

- Crea un ruolo IAM e una funzione Lambda, quindi carica il codice del gestore.
- Invocare la funzione con un singolo parametro e ottenere i risultati.
- Aggiorna il codice della funzione e configuralo con una variabile di ambiente.
- Invocare la funzione con nuovi parametri e ottenere i risultati. Visualizza il log di esecuzione restituito.
- Elenca le funzioni dell'account, quindi elimina le risorse.

Per ulteriori informazioni, consulta [Creare una funzione Lambda con la console](#).

## .NET

### SDK per .NET

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Creare metodi che eseguono operazioni Lambda.

```
namespace LambdaActions;

using Amazon.Lambda;
using Amazon.Lambda.Model;

/// <summary>
/// A class that implements AWS Lambda methods.
/// </summary>
public class LambdaWrapper
{
    private readonly IAmazonLambda _lambdaService;

    /// <summary>
    /// Constructor for the LambdaWrapper class.
    /// </summary>
    /// <param name="lambdaService">An initialized Lambda service client.</param>
    public LambdaWrapper(IAmazonLambda lambdaService)
    {
        _lambdaService = lambdaService;
    }

    /// <summary>
    /// Creates a new Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function.</param>
    /// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
    /// bucket where the zip file containing the code is located.</param>
    /// <param name="s3Key">The Amazon S3 key of the zip file.</param>
    /// <param name="role">The Amazon Resource Name (ARN) of a role with the
    /// appropriate Lambda permissions.</param>
    /// <param name="handler">The name of the handler function.</param>
```

```

    /// <returns>The Amazon Resource Name (ARN) of the newly created
    /// Lambda function.</returns>
    public async Task<string> CreateLambdaFunctionAsync(
        string functionName,
        string s3Bucket,
        string s3Key,
        string role,
        string handler)
    {
        // Defines the location for the function code.
        // S3Bucket - The S3 bucket where the file containing
        //           the source code is stored.
        // S3Key     - The name of the file containing the code.
        var functionCode = new FunctionCode
        {
            S3Bucket = s3Bucket,
            S3Key = s3Key,
        };

        var createFunctionRequest = new CreateFunctionRequest
        {
            FunctionName = functionName,
            Description = "Created by the Lambda .NET API",
            Code = functionCode,
            Handler = handler,
            Runtime = Runtime.Dotnet6,
            Role = role,
        };

        var reponse = await
        _lambdaService.CreateFunctionAsync(createFunctionRequest);
        return reponse.FunctionArn;
    }

    /// <summary>
    /// Delete an AWS Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// delete.</param>
    /// <returns>A Boolean value that indicates the success of the action.</
returns>
    public async Task<bool> DeleteFunctionAsync(string functionName)
    {

```

```
var request = new DeleteFunctionRequest
{
    FunctionName = functionName,
};

var response = await _lambdaService.DeleteFunctionAsync(request);

// A return value of NoContent means that the request was processed.
// In this case, the function was deleted, and the return value
// is intentionally blank.
return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
}

/// <summary>
/// Gets information about a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function for
/// which to retrieve information.</param>
/// <returns>Async Task.</returns>
public async Task<FunctionConfiguration> GetFunctionAsync(string
functionName)
{
    var functionRequest = new GetFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.GetFunctionAsync(functionRequest);
    return response.Configuration;
}

/// <summary>
/// Invoke a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// invoke.</param>
/// <param name="parameters">The parameter values that will be passed to the
function.</param>
/// <returns>A System Threading Task.</returns>
public async Task<string> InvokeFunctionAsync(
    string functionName,
    string parameters)
```

```

    {
        var payload = parameters;
        var request = new InvokeRequest
        {
            FunctionName = functionName,
            Payload = payload,
        };

        var response = await _lambdaService.InvokeAsync(request);
        MemoryStream stream = response.Payload;
        string returnValue =
System.Text.Encoding.UTF8.GetString(stream.ToArray());
        return returnValue;
    }

    /// <summary>
    /// Get a list of Lambda functions.
    /// </summary>
    /// <returns>A list of FunctionConfiguration objects.</returns>
    public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
    {
        var functionList = new List<FunctionConfiguration>();

        var functionPaginator =
            _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
        await foreach (var function in functionPaginator.Functions)
        {
            functionList.Add(function);
        }

        return functionList;
    }

    /// <summary>
    /// Update an existing Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to update.</
param>
    /// <param name="bucketName">The bucket where the zip file containing
    /// the Lambda function code is stored.</param>
    /// <param name="key">The key name of the source code file.</param>
    /// <returns>Async Task.</returns>

```



```

public async Task UpdateFunctionCodeAsync(
    string functionName,
    string bucketName,
    string key)
{
    var functionCodeRequest = new UpdateFunctionCodeRequest
    {
        FunctionName = functionName,
        Publish = true,
        S3Bucket = bucketName,
        S3Key = key,
    };

    var response = await
_lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
    Console.WriteLine($"The Function was last modified at
{response.LastModified}.");
}

/// <summary>
/// Update the code of a Lambda function.
/// </summary>
/// <param name="functionName">The name of the function to update.</param>
/// <param name="functionHandler">The code that performs the function's
actions.</param>
/// <param name="environmentVariables">A dictionary of environment
variables.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> UpdateFunctionConfigurationAsync(
    string functionName,
    string functionHandler,
    Dictionary<string, string> environmentVariables)
{
    var request = new UpdateFunctionConfigurationRequest
    {
        Handler = functionHandler,
        FunctionName = functionName,
        Environment = new Amazon.Lambda.Model.Environment { Variables =
environmentVariables },
    };

    var response = await
_lambdaService.UpdateFunctionConfigurationAsync(request);

```

```
        Console.WriteLine(response.LastModified);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

}
```

Creare una funzione che esegue lo scenario.

```
global using System.Threading.Tasks;
global using Amazon.IdentityManagement;
global using Amazon.Lambda;
global using LambdaActions;
global using LambdaScenarioCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;

using Amazon.Lambda.Model;
using Microsoft.Extensions.Configuration;

namespace LambdaBasics;

public class LambdaBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
                        LogLevel.Information)
            )
            .Build();
    }
}
```

```

        .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
    .ConfigureServices((_, services) =>
services.AddAWSService<IAmazonLambda>()
.AddAWSService<IAmazonIdentityManagementService>()
.AddTransient<LambdaWrapper>()
.AddTransient<LambdaRoleWrapper>()
.AddTransient<UIWrapper>()
)
    .Build();

var configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("settings.json") // Load test settings from .json file.
    .AddJsonFile("settings.local.json",
true) // Optionally load local settings.
    .Build();

logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
    .CreateLogger<LambdaBasics>();

var lambdaWrapper = host.Services.GetRequiredService<LambdaWrapper>();
var lambdaRoleWrapper =
host.Services.GetRequiredService<LambdaRoleWrapper>();
var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

string functionName = configuration["FunctionName"]!;
string roleName = configuration["RoleName"]!;
string policyDocument = "{" +
    "  \"Version\": \"2012-10-17\", " +
    "  \"Statement\": [ " +
    "    { " +
    "      \"Effect\": \"Allow\", " +
    "      \"Principal\": { " +
    "        \"Service\": \"lambda.amazonaws.com\" " +
    "      }, " +
    "      \"Action\": \"sts:AssumeRole\" " +
    "    } " +
    "  ] " +
    "}";

var incrementHandler = configuration["IncrementHandler"];
var calculatorHandler = configuration["CalculatorHandler"];

```

```
var bucketName = configuration["BucketName"];
var incrementKey = configuration["IncrementKey"];
var calculatorKey = configuration["CalculatorKey"];
var policyArn = configuration["PolicyArn"];

uiWrapper.DisplayLambdaBasicsOverview();

// Create the policy to use with the AWS Lambda functions and then attach
the
// policy to a new role.
var roleArn = await lambdaRoleWrapper.CreateLambdaRoleAsync(roleName,
policyDocument);

Console.WriteLine("Waiting for role to become active.");
uiWrapper.WaitABit(15, "Wait until the role is active before trying to
use it.");

// Attach the appropriate AWS Identity and Access Management (IAM) role
policy to the new role.
var success = await
lambdaRoleWrapper.AttachLambdaRolePolicyAsync(policyArn, roleName);
uiWrapper.WaitABit(10, "Allow time for the IAM policy to be attached to
the role.");

// Create the Lambda function using a zip file stored in an Amazon Simple
Storage Service
// (Amazon S3) bucket.
uiWrapper.DisplayTitle("Create Lambda Function");
Console.WriteLine($"Creating the AWS Lambda function: {functionName}.");
var lambdaArn = await lambdaWrapper.CreateLambdaFunctionAsync(
    functionName,
    bucketName,
    incrementKey,
    roleArn,
    incrementHandler);

Console.WriteLine("Waiting for the new function to be available.");
Console.WriteLine($"The AWS Lambda ARN is {lambdaArn}");

// Get the Lambda function.
Console.WriteLine($"Getting the {functionName} AWS Lambda function.");
FunctionConfiguration config;
do
{
```

```

        config = await lambdaWrapper.GetFunctionAsync(functionName);
        Console.WriteLine(".");
    }
    while (config.State != State.Active);

    Console.WriteLine($"\\n\\nThe function, {functionName} has been created.");
    Console.WriteLine($"\\nThe runtime of this Lambda function is
{config.Runtime}.");

    uiWrapper.PressEnter();

    // List the Lambda functions.
    uiWrapper.DisplayTitle("Listing all Lambda functions.");
    var functions = await lambdaWrapper.ListFunctionsAsync();
    DisplayFunctionList(functions);

    uiWrapper.DisplayTitle("Invoke increment function");
    Console.WriteLine("Now that it has been created, invoke the Lambda
increment function.");
    string? value;
    do
    {
        Console.WriteLine("Enter a value to increment: ");
        value = Console.ReadLine();
    }
    while (string.IsNullOrEmpty(value));

    string functionParameters = "{" +
        "\\\"action\\\": \\\"increment\\\", " +
        "\\\"x\\\": \\\"" + value + "\\\" " +
        "}";
    var answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
    Console.WriteLine($"\\\"{value}\\\" + 1 = {answer}.");

    uiWrapper.DisplayTitle("Update function");
    Console.WriteLine("Now update the Lambda function code.");
    await lambdaWrapper.UpdateFunctionCodeAsync(functionName, bucketName,
calculatorKey);

    do
    {
        config = await lambdaWrapper.GetFunctionAsync(functionName);
        Console.WriteLine(".");
    }
    while (config.State != State.Active);

```

```
    }
    while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

    await lambdaWrapper.UpdateFunctionConfigurationAsync(
        functionName,
        calculatorHandler,
        new Dictionary<string, string> { { "LOG_LEVEL", "DEBUG" } });

    do
    {
        config = await lambdaWrapper.GetFunctionAsync(functionName);
        Console.WriteLine(".");
    }
    while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

    uiWrapper.DisplayTitle("Call updated function");
    Console.WriteLine("Now call the updated function...");

    bool done = false;

    do
    {
        string? opSelected;

        Console.WriteLine("Select the operation to perform:");
        Console.WriteLine("\t1. add");
        Console.WriteLine("\t2. subtract");
        Console.WriteLine("\t3. multiply");
        Console.WriteLine("\t4. divide");
        Console.WriteLine("\t0r enter \"q\" to quit.");
        Console.WriteLine("Enter the number (1, 2, 3, 4, or q) of the
operation you want to perform: ");
        do
        {
            Console.Write("Your choice? ");
            opSelected = Console.ReadLine();
        }
        while (opSelected == string.Empty);

        var operation = (opSelected) switch
        {
            "1" => "add",
            "2" => "subtract",
            "3" => "multiply",
```

```
        "4" => "divide",
        "q" => "quit",
        _ => "add",
    };

    if (operation == "quit")
    {
        done = true;
    }
    else
    {
        // Get two numbers and an action from the user.
        value = string.Empty;
        do
        {
            Console.Write("Enter the first value: ");
            value = Console.ReadLine();
        }
        while (value == string.Empty);

        string? value2;
        do
        {
            Console.Write("Enter a second value: ");
            value2 = Console.ReadLine();
        }
        while (value2 == string.Empty);

        functionParameters = "{" +
            "\"action\": \"" + operation + "\", " +
            "\"x\": \"" + value + "\", " +
            "\"y\": \"" + value2 + "\"" +
            "}";

        answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
            functionParameters);
        Console.WriteLine($"The answer when we {operation} the two
            numbers is: {answer}.");
    }

    uiWrapper.PressEnter();
} while (!done);

// Delete the function created earlier.
```

```
uiWrapper.DisplayTitle("Clean up resources");
// Detach the IAM policy from the IAM role.
Console.WriteLine("First detach the IAM policy from the role.");
success = await lambdaRoleWrapper.DetachLambdaRolePolicyAsync(policyArn,
roleName);
uiWrapper.WaitABit(15, "Let's wait for the policy to be fully detached
from the role.");

Console.WriteLine("Delete the AWS Lambda function.");
success = await lambdaWrapper.DeleteFunctionAsync(functionName);
if (success)
{
    Console.WriteLine($"The {functionName} function was deleted.");
}
else
{
    Console.WriteLine($"Could not remove the function {functionName}");
}

// Now delete the IAM role created for use with the functions
// created by the application.
Console.WriteLine("Now we can delete the role that we created.");
success = await lambdaRoleWrapper.DeleteLambdaRoleAsync(roleName);
if (success)
{
    Console.WriteLine("The role has been successfully removed.");
}
else
{
    Console.WriteLine("Couldn't delete the role.");
}

Console.WriteLine("The Lambda Scenario is now complete.");
uiWrapper.PressEnter();

// Displays a formatted list of existing functions returned by the
// LambdaMethods.ListFunctions.
void DisplayFunctionList(List<FunctionConfiguration> functions)
{
    functions.ForEach(functionConfig =>
    {
        Console.WriteLine($"{functionConfig.FunctionName}\t{functionConfig.Description}");
    });
}
```



```

        });
    }
}

namespace LambdaActions;

using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

public class LambdaRoleWrapper
{
    private readonly IAmazonIdentityManagementService _lambdaRoleService;

    public LambdaRoleWrapper(IAmazonIdentityManagementService lambdaRoleService)
    {
        _lambdaRoleService = lambdaRoleService;
    }

    /// <summary>
    /// Attach an AWS Identity and Access Management (IAM) role policy to the
    /// IAM role to be assumed by the AWS Lambda functions created for the
    scenario.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM
    policy.</param>
    /// <param name="roleName">The name of the IAM role to attach the IAM policy
    to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachLambdaRolePolicyAsync(string policyArn, string
    roleName)
    {
        var response = await _lambdaRoleService.AttachRolePolicyAsync(new
    AttachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to create.</param>
    /// <param name="policyDocument">The policy document for the new IAM role.</
    param>

```

```

    /// <returns>A string representing the ARN for newly created role.</returns>
    public async Task<string> CreateLambdaRoleAsync(string roleName, string
policyDocument)
    {
        var request = new CreateRoleRequest
        {
            AssumeRolePolicyDocument = policyDocument,
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.CreateRoleAsync(request);
        return response.Role.Arn;
    }

    /// <summary>
    /// Deletes an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteLambdaRoleAsync(string roleName)
    {
        var request = new DeleteRoleRequest
        {
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.DeleteRoleAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    public async Task<bool> DetachLambdaRolePolicyAsync(string policyArn, string
roleName)
    {
        var response = await _lambdaRoleService.DetachRolePolicyAsync(new
DetachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}

namespace LambdaScenarioCommon;

public class UIWrapper

```

```
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the AWS Lambda Basics scenario.
    /// </summary>
    public void DisplayLambdaBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to AWS Lambda Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an AWS Identity and Access Management (IAM) role that will be assumed by the functions we create.");
        Console.WriteLine("\t2. Attaches an IAM role policy that has Lambda permissions.");
        Console.WriteLine("\t3. Creates a Lambda function that increments the value passed to it.");
        Console.WriteLine("\t4. Calls the increment function and passes a value.");
        Console.WriteLine("\t5. Updates the code so that the function is a simple calculator.");
        Console.WriteLine("\t6. Calls the calculator function with the values entered.");
        Console.WriteLine("\t7. Deletes the Lambda function.");
        Console.WriteLine("\t7. Detaches the IAM role policy.");
        Console.WriteLine("\t8. Deletes the IAM role.");
        PressEnter();
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
```

```
/// <returns>The padded string.</returns>
public string CenterString(string strToCenter)
{
    var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
    var leftPad = new string(' ', padAmount);
    return $"{leftPad}{strToCenter}";
}

/// <summary>
/// Display a line of hyphens, the centered text of the title and another
/// line of hyphens.
/// </summary>
/// <param name="strTitle">The string to be displayed.</param>
public void DisplayTitle(string strTitle)
{
    Console.WriteLine(SepBar);
    Console.WriteLine(CenterString(strTitle));
    Console.WriteLine(SepBar);
}

/// <summary>
/// Display a countdown and wait for a number of seconds.
/// </summary>
/// <param name="numSeconds">The number of seconds to wait.</param>
public void WaitABit(int numSeconds, string msg)
{
    Console.WriteLine(msg);

    // Wait for the requested number of seconds.
    for (int i = numSeconds; i > 0; i--)
    {
        System.Threading.Thread.Sleep(1000);
        Console.Write($"{i}...");
    }

    PressEnter();
}
}
```

Definire un gestore Lambda che incrementa un numero.

```

using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace LambdaIncrement;

public class Function
{
    /// <summary>
    /// A simple function increments the integer parameter.
    /// </summary>
    /// <param name="input">A JSON string containing an action, which must be
    /// "increment" and a string representing the value to increment.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</
    param>
    /// <returns>A string representing the incremented value of the parameter.</
    returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
    context)
    {
        if (input["action"] == "increment")
        {
            int inputValue = Convert.ToInt32(input["x"]);
            return inputValue + 1;
        }
        else
        {
            return 0;
        }
    }
}

```

Definire un secondo gestore Lambda che esegue operazioni aritmetiche.

```

using Amazon.Lambda.Core;

```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace LambdaCalculator;

public class Function
{
    /// <summary>
    /// A simple function that takes two number in string format and performs
    /// the requested arithmetic function.
    /// </summary>
    /// <param name="input">JSON data containing an action, and x and y values.
    /// Valid actions include: add, subtract, multiply, and divide.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</
param>
    /// <returns>A string representing the results of the calculation.</returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
context)
    {
        var action = input["action"];
        int x = Convert.ToInt32(input["x"]);
        int y = Convert.ToInt32(input["y"]);
        int result;
        switch (action)
        {
            case "add":
                result = x + y;
                break;
            case "subtract":
                result = x - y;
                break;
            case "multiply":
                result = x * y;
                break;
            case "divide":
                if (y == 0)
                {
                    Console.Error.WriteLine("Divide by zero error.");
                    result = 0;
                }
        }
    }
}
```

```
        else
            result = x / y;
            break;
        default:
            Console.Error.WriteLine($"{action} is not a valid operation.");
            result = 0;
            break;
    }
    return result;
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per .NET .
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## C++

### SDK per C++

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
//! Get started with functions scenario.
/*!
\param clientConfig: AWS client configuration.
\return bool: Successful completion.
```

```

*/
bool AwsDoc::Lambda::getStartedWithFunctionsScenario(
    const Aws::Client::ClientConfiguration &clientConfig) {

    Aws::Lambda::LambdaClient client(clientConfig);

    // 1. Create an AWS Identity and Access Management (IAM) role for Lambda
    function.
    Aws::String roleArn;
    if (!getIamRoleArn(roleArn, clientConfig)) {
        return false;
    }

    // 2. Create a Lambda function.
    int seconds = 0;
    do {
        Aws::Lambda::Model::CreateFunctionRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        request.SetDescription(LAMBDA_DESCRIPTION); // Optional.
#ifdef USE_CPP_LAMBDA_FUNCTION
        request.SetRuntime(Aws::Lambda::Model::Runtime::provided_al2);
        request.SetTimeout(15);
        request.SetMemorySize(128);

        // Assume the AWS Lambda function was built in Docker with same
        architecture
        // as this code.
#ifdef __x86_64__
        request.SetArchitectures({Aws::Lambda::Model::Architecture::x86_64});
#elif defined(__aarch64__)
        request.SetArchitectures({Aws::Lambda::Model::Architecture::arm64});
#else
#error "Unimplemented architecture"
#endif // defined(architecture)
#else
        request.SetRuntime(Aws::Lambda::Model::Runtime::python3_9);
#endif

        request.SetRole(roleArn);
        request.SetHandler(LAMBDA_HANDLER_NAME);
        request.SetPublish(true);
        Aws::Lambda::Model::FunctionCode code;
        std::ifstream ifstream(INCREMENT_LAMBDA_CODE.c_str(),
                               std::ios_base::in | std::ios_base::binary);
        if (!ifstream.is_open()) {

```



```

        std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
std::endl;

#ifdef USE_CPP_LAMBDA_FUNCTION
        std::cerr
            << "The cpp Lambda function must be built following the
instructions in the cpp_lambda/README.md file. "
            << std::endl;
#endif

        deleteIamRole(clientConfig);
        return false;
    }

    Aws::StringStream buffer;
    buffer << ifstream.rdbuf();

    code.SetZipFile(Aws::Utils::ByteBuffer((unsigned char *)
buffer.str().c_str(),
                                           buffer.str().length()));

    request.SetCode(code);

    Aws::Lambda::Model::CreateFunctionOutcome outcome =
client.CreateFunction(
    request);

    if (outcome.IsSuccess()) {
        std::cout << "The lambda function was successfully created. " <<
seconds
            << " seconds elapsed." << std::endl;
        break;
    }
    else if (outcome.GetError().GetErrorType() ==
        Aws::Lambda::LambdaErrors::INVALID_PARAMETER_VALUE &&
        outcome.GetError().GetMessage().find("role") >= 0) {
        if ((seconds % 5) == 0) { // Log status every 10 seconds.
            std::cout
                << "Waiting for the IAM role to become available as a
CreateFunction parameter. "
                << seconds
                << " seconds elapsed." << std::endl;

            std::cout << outcome.GetError().GetMessage() << std::endl;
        }
    }
}

```

```

        else {
            std::cerr << "Error with CreateFunction. "
                << outcome.GetError().GetMessage()
                << std::endl;
            deleteIamRole(clientConfig);
            return false;
        }
        ++seconds;
        std::this_thread::sleep_for(std::chrono::seconds(1));
    } while (60 > seconds);

    std::cout << "The current Lambda function increments 1 by an input." <<
std::endl;

// 3. Invoke the Lambda function.
{
    int increment = askQuestionForInt("Enter an increment integer: ");

    Aws::Lambda::Model::InvokeResult invokeResult;
    Aws::Utils::Json::JsonValue jsonPayload;
    jsonPayload.WithString("action", "increment");
    jsonPayload.WithInteger("number", increment);
    if (invokeLambdaFunction(jsonPayload, Aws::Lambda::Model::LogType::Tail,
        invokeResult, client)) {
        Aws::Utils::Json::JsonValue jsonValue(invokeResult.GetPayload());
        Aws::Map<Aws::String, Aws::Utils::Json::JsonValue> values =
            jsonValue.View().GetAllObjects();
        auto iter = values.find("result");
        if (iter != values.end() && iter->second.IsIntegerType()) {
            {
                std::cout << INCREMENT_RESULT_PREFIX
                    << iter->second.AsInteger() << std::endl;
            }
        }
        else {
            std::cout << "There was an error in execution. Here is the log."
                << std::endl;
            Aws::Utils::ByteBuffer buffer =
                Aws::Utils::HashingUtils::Base64Decode(
                    invokeResult.GetLogResult());
            std::cout << "With log " << buffer.GetUnderlyingData() <<
std::endl;
        }
    }
}

```

```

    }

    std::cout
        << "The Lambda function will now be updated with new code. Press
return to continue, ";
    Aws::String answer;
    std::getline(std::cin, answer);

    // 4. Update the Lambda function code.
    {
        Aws::Lambda::Model::UpdateFunctionCodeRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        std::ifstream ifstream(CALCULATOR_LAMBDA_CODE.c_str(),
                                std::ios_base::in | std::ios_base::binary);
        if (!ifstream.is_open()) {
            std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
std::endl;
}

#ifdef USE_CPP_LAMBDA_FUNCTION
    std::cerr
        << "The cpp Lambda function must be built following the
instructions in the cpp_lambda/README.md file. "
        << std::endl;
#endif

    deleteLambdaFunction(client);
    deleteIamRole(clientConfig);
    return false;
}

    Aws::StringStream buffer;
    buffer << ifstream.rdbuf();
    request.SetZipFile(
        Aws::Utils::ByteBuffer((unsigned char *) buffer.str().c_str(),
                                buffer.str().length()));

    request.SetPublish(true);

    Aws::Lambda::Model::UpdateFunctionCodeOutcome outcome =
client.UpdateFunctionCode(
    request);

    if (outcome.IsSuccess()) {
        std::cout << "The lambda code was successfully updated." <<
std::endl;
    }
}

```

```

        else {
            std::cerr << "Error with Lambda::UpdateFunctionCode. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    }

    std::cout
        << "This function uses an environment variable to control the logging
level."
        << std::endl;
    std::cout
        << "UpdateFunctionConfiguration will be used to set the LOG_LEVEL to
DEBUG."
        << std::endl;
    seconds = 0;

    // 5. Update the Lambda function configuration.
    do {
        ++seconds;
        std::this_thread::sleep_for(std::chrono::seconds(1));
        Aws::Lambda::Model::UpdateFunctionConfigurationRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        Aws::Lambda::Model::Environment environment;
        environment.AddVariables("LOG_LEVEL", "DEBUG");
        request.SetEnvironment(environment);

        Aws::Lambda::Model::UpdateFunctionConfigurationOutcome outcome =
client.UpdateFunctionConfiguration(
            request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda configuration was successfully updated."
                      << std::endl;
            break;
        }

        // RESOURCE_IN_USE: function code update not completed.
        else if (outcome.GetError().GetErrorType() !=
            Aws::Lambda::LambdaErrors::RESOURCE_IN_USE) {
            if ((seconds % 10) == 0) { // Log status every 10 seconds.
                std::cout << "Lambda function update in progress . After " <<
seconds
                            << " seconds elapsed." << std::endl;
            }
        }
    } while (true);

```

```

        }
    }
    else {
        std::cerr << "Error with Lambda::UpdateFunctionConfiguration. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

} while (0 < seconds);

if (0 > seconds) {
    std::cerr << "Function failed to become active." << std::endl;
}
else {
    std::cout << "Updated function active after " << seconds << " seconds."
              << std::endl;
}

std::cout
    << "\nThe new code applies an arithmetic operator to two variables, x
an y."
    << std::endl;
std::vector<Aws::String> operators = {"plus", "minus", "times", "divided-
by"};
for (size_t i = 0; i < operators.size(); ++i) {
    std::cout << "    " << i + 1 << " " << operators[i] << std::endl;
}

// 6. Invoke the updated Lambda function.
do {
    int operatorIndex = askQuestionForIntRange("Select an operator index 1 -
4 ", 1,
                                              4);

    int x = askQuestionForInt("Enter an integer for the x value ");
    int y = askQuestionForInt("Enter an integer for the y value ");

    Aws::Utils::Json::JsonValue calculateJsonPayload;
    calculateJsonPayload.WithString("action", operators[operatorIndex - 1]);
    calculateJsonPayload.WithInteger("x", x);
    calculateJsonPayload.WithInteger("y", y);
    Aws::Lambda::Model::InvokeResult calculatedResult;
    if (invokeLambdaFunction(calculateJsonPayload,
                            Aws::Lambda::Model::LogType::Tail,
                            calculatedResult, client)) {

```

```

        Aws::Utils::Json::JsonValue jsonValue(calculatedResult.GetPayload());
        Aws::Map<Aws::String, Aws::Utils::Json::JsonView> values =
            jsonValue.View().GetAllObjects();
        auto iter = values.find("result");
        if (iter != values.end() && iter->second.IsIntegerType()) {
            std::cout << ARITHMETIC_RESULT_PREFIX << x << " "
                << operators[operatorIndex - 1] << " "
                << y << " is " << iter->second.AsInteger() <<
std::endl;
        }
        else if (iter != values.end() && iter->second.IsFloatingPointType())
{
            std::cout << ARITHMETIC_RESULT_PREFIX << x << " "
                << operators[operatorIndex - 1] << " "
                << y << " is " << iter->second.AsDouble() << std::endl;
        }
        else {
            std::cout << "There was an error in execution. Here is the log."
                << std::endl;
            Aws::Utils::ByteBuffer buffer =
        Aws::Utils::HashingUtils::Base64Decode(
                calculatedResult.GetLogResult());
            std::cout << "With log " << buffer.GetUnderlyingData() <<
std::endl;
        }
    }

    answer = askQuestion("Would you like to try another operation? (y/n) ");
    } while (answer == "y");

    std::cout
        << "A list of the lambda functions will be retrieved. Press return to
continue, ";
    std::getline(std::cin, answer);

    // 7. List the Lambda functions.

    std::vector<Aws::String> functions;
    Aws::String marker;

    do {
        Aws::Lambda::Model::ListFunctionsRequest request;
        if (!marker.empty()) {
            request.SetMarker(marker);

```

```

    }

    Aws::Lambda::Model::ListFunctionsOutcome outcome = client.ListFunctions(
        request);

    if (outcome.IsSuccess()) {
        const Aws::Lambda::Model::ListFunctionsResult &result =
outcome.GetResult();
        std::cout << result.GetFunctions().size()
            << " lambda functions were retrieved." << std::endl;

        for (const Aws::Lambda::Model::FunctionConfiguration
&functionConfiguration: result.GetFunctions()) {
            functions.push_back(functionConfiguration.GetFunctionName());
            std::cout << functions.size() << " "
                << functionConfiguration.GetDescription() << std::endl;
            std::cout << "    "
                <<
Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
                functionConfiguration.GetRuntime()) << ": "
                << functionConfiguration.GetHandler()
                << std::endl;
        }
        marker = result.GetNextMarker();
    }
    else {
        std::cerr << "Error with Lambda::ListFunctions. "
            << outcome.GetError().GetMessage()
            << std::endl;
    }
} while (!marker.empty());

// 8. Get a Lambda function.
if (!functions.empty()) {
    std::stringstream question;
    question << "Choose a function to retrieve between 1 and " <<
functions.size()
        << " ";
    int functionIndex = askQuestionForIntRange(question.str(), 1,
static_cast<int>(functions.size()));

    Aws::String functionName = functions[functionIndex - 1];

```

```

        Aws::Lambda::Model::GetFunctionRequest request;
        request.SetFunctionName(functionName);

        Aws::Lambda::Model::GetFunctionOutcome outcome =
client.GetFunction(request);

        if (outcome.IsSuccess()) {
            std::cout << "Function retrieve.\n" <<

outcome.GetResult().GetConfiguration().Jsonize().View().WriteReadable()
            << std::endl;
        }
        else {
            std::cerr << "Error with Lambda::GetFunction. "
            << outcome.GetError().GetMessage()
            << std::endl;
        }
    }
}

std::cout << "The resources will be deleted. Press return to continue, ";
std::getline(std::cin, answer);

// 9. Delete the Lambda function.
bool result = deleteLambdaFunction(client);

// 10. Delete the IAM role.
return result && deleteIamRole(clientConfig);
}

//! Routine which invokes a Lambda function and returns the result.
/*!
 \param jsonPayload: Payload for invoke function.
 \param logType: Log type setting for invoke function.
 \param invokeResult: InvokeResult object to receive the result.
 \param client: Lambda client.
 \return bool: Successful completion.
 */
bool
AwsDoc::Lambda::invokeLambdaFunction(const Aws::Utils::Json::JsonValue
&jsonPayload,
                                     Aws::Lambda::Model::LogType logType,
                                     Aws::Lambda::Model::InvokeResult
&invokeResult,
                                     const Aws::Lambda::LambdaClient &client) {

```



```

    int seconds = 0;
    bool result = false;
    /*
     * In this example, the Invoke function can be called before recently created
resources are
     * available. The Invoke function is called repeatedly until the resources
are
     * available.
     */
    do {
        Aws::Lambda::Model::InvokeRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        request.SetLogType(logType);
        std::shared_ptr<Aws::IOStream> payload =
        Aws::MakeShared<Aws::StringStream>(
            "FunctionTest");
        *payload << jsonPayload.View().WriteReadable();
        request.SetBody(payload);
        request.SetContentType("application/json");
        Aws::Lambda::Model::InvokeOutcome outcome = client.Invoke(request);

        if (outcome.IsSuccess()) {
            invokeResult = std::move(outcome.GetResult());
            result = true;
            break;
        }

        // ACCESS_DENIED: because the role is not available yet.
        // RESOURCE_CONFLICT: because the Lambda function is being created or
updated.
        else if ((outcome.GetError().GetErrorType() ==
            Aws::Lambda::LambdaErrors::ACCESS_DENIED) ||
            (outcome.GetError().GetErrorType() ==
            Aws::Lambda::LambdaErrors::RESOURCE_CONFLICT)) {
            if ((seconds % 5) == 0) { // Log status every 10 seconds.
                std::cout << "Waiting for the invoke api to be available, status
" <<

                ((outcome.GetError().GetErrorType() ==
                    Aws::Lambda::LambdaErrors::ACCESS_DENIED ?
                    "ACCESS_DENIED" : "RESOURCE_CONFLICT")) << ". " <<

seconds

                << " seconds elapsed." << std::endl;
            }
        }
    }
}

```

```
        else {
            std::cerr << "Error with Lambda::InvokeRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;

            break;
        }
        ++seconds;
        std::this_thread::sleep_for(std::chrono::seconds(1));
    } while (seconds < 60);

    return result;
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per C++ .
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

Go

SDK per Go V2

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Creare uno scenario interattivo che ti mostri come iniziare a usare le funzioni Lambda.

```
import (
    "archive/zip"
```

```
"bytes"
"context"
"encoding/base64"
"encoding/json"
"errors"
"fmt"
"log"
"os"
"strings"
"time"

"github.com/aws/aws-sdk-go-v2/aws"
"github.com/aws/aws-sdk-go-v2/service/iam"
iamtypes "github.com/aws/aws-sdk-go-v2/service/iam/types"
"github.com/aws/aws-sdk-go-v2/service/lambda"
"github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
"github.com/awsdocs/aws-doc-sdk-examples/gov2/lambda/actions"
)

// GetStartedFunctionsScenario shows you how to use AWS Lambda to perform the
// following
// actions:
//
// 1. Create an AWS Identity and Access Management (IAM) role and Lambda
//    function, then upload handler code.
// 2. Invoke the function with a single parameter and get results.
// 3. Update the function code and configure with an environment variable.
// 4. Invoke the function with new parameters and get results. Display the
//    returned execution log.
// 5. List the functions for your account, then clean up resources.
type GetStartedFunctionsScenario struct {
    sdkConfig      aws.Config
    functionWrapper actions.FunctionWrapper
    questioner     demotools.IQuestioner
    helper         IScenarioHelper
    isTestRun      bool
}

// NewGetStartedFunctionsScenario constructs a GetStartedFunctionsScenario
// instance from a configuration.
// It uses the specified config to get a Lambda client and create wrappers for
// the actions
// used in the scenario.
```

```

func NewGetStartedFunctionsScenario(sdkConfig aws.Config, questioner
    demotools.IQuestioner,
    helper IScenarioHelper) GetStartedFunctionsScenario {
    lambdaClient := lambda.NewFromConfig(sdkConfig)
    return GetStartedFunctionsScenario{
        sdkConfig:      sdkConfig,
        functionWrapper: actions.FunctionWrapper{LambdaClient: lambdaClient},
        questioner:      questioner,
        helper:          helper,
    }
}

// Run runs the interactive scenario.
func (scenario GetStartedFunctionsScenario) Run(ctx context.Context) {
    defer func() {
        if r := recover(); r != nil {
            log.Printf("Something went wrong with the demo.\n")
        }
    }()

    log.Println(strings.Repeat("-", 88))
    log.Println("Welcome to the AWS Lambda get started with functions demo.")
    log.Println(strings.Repeat("-", 88))

    role := scenario.GetOrCreateRole(ctx)
    funcName := scenario.CreateFunction(ctx, role)
    scenario.InvokeIncrement(ctx, funcName)
    scenario.UpdateFunction(ctx, funcName)
    scenario.InvokeCalculator(ctx, funcName)
    scenario.ListFunctions(ctx)
    scenario.Cleanup(ctx, role, funcName)

    log.Println(strings.Repeat("-", 88))
    log.Println("Thanks for watching!")
    log.Println(strings.Repeat("-", 88))
}

// GetOrCreateRole checks whether the specified role exists and returns it if it
// does.
// Otherwise, a role is created that specifies Lambda as a trusted principal.
// The AWSLambdaBasicExecutionRole managed policy is attached to the role and the
// role
// is returned.

```

```

func (scenario GetStartedFunctionsScenario) GetOrCreateRole(ctx context.Context)
    *iamtypes.Role {
    var role *iamtypes.Role
    iamClient := iam.NewFromConfig(scenario.sdkConfig)
    log.Println("First, we need an IAM role that Lambda can assume.")
    roleName := scenario.questioner.Ask("Enter a name for the role:",
    demotools.NotEmpty{})
    getOutput, err := iamClient.GetRole(ctx, &iam.GetRoleInput{
        RoleName: aws.String(roleName)})
    if err != nil {
        var noSuch *iamtypes.NoSuchEntityException
        if errors.As(err, &noSuch) {
            log.Printf("Role %v doesn't exist. Creating it....\n", roleName)
        } else {
            log.Panicf("Couldn't check whether role %v exists. Here's why: %v\n",
                roleName, err)
        }
    } else {
        role = getOutput.Role
        log.Printf("Found role %v.\n", *role.RoleName)
    }
    if role == nil {
        trustPolicy := PolicyDocument{
            Version: "2012-10-17",
            Statement: []PolicyStatement{{
                Effect:      "Allow",
                Principal: map[string]string{"Service": "lambda.amazonaws.com"},
                Action:      []string{"sts:AssumeRole"},
            }},
        }
        policyArn := "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
        createOutput, err := iamClient.CreateRole(ctx, &iam.CreateRoleInput{
            AssumeRolePolicyDocument: aws.String(trustPolicy.String()),
            RoleName:                  aws.String(roleName),
        })
        if err != nil {
            log.Panicf("Couldn't create role %v. Here's why: %v\n", roleName, err)
        }
        role = createOutput.Role
        _, err = iamClient.AttachRolePolicy(ctx, &iam.AttachRolePolicyInput{
            PolicyArn: aws.String(policyArn),
            RoleName:  aws.String(roleName),
        })
        if err != nil {

```

```

    log.Panicf("Couldn't attach a policy to role %v. Here's why: %v\n", roleName,
err)
}
log.Printf("Created role %v.\n", *role.RoleName)
log.Println("Let's give AWS a few seconds to propagate resources...")
scenario.helper.Pause(10)
}
log.Println(strings.Repeat("-", 88))
return role
}

// CreateFunction creates a Lambda function and uploads a handler written in
Python.
// The code for the Python handler is packaged as a []byte in .zip format.
func (scenario GetStartedFunctionsScenario) CreateFunction(ctx context.Context,
role *iamtypes.Role) string {
log.Println("Let's create a function that increments a number.\n" +
"The function uses the 'lambda_handler_basic.py' script found in the\n" +
"'handlers' directory of this project.")
funcName := scenario.questioner.Ask("Enter a name for the Lambda function:",
demotools.NotEmpty{})
zipPackage := scenario.helper.CreateDeploymentPackage("lambda_handler_basic.py",
fmt.Sprintf("%v.py", funcName))
log.Printf("Creating function %v and waiting for it to be ready.", funcName)
funcState := scenario.functionWrapper.CreateFunction(ctx, funcName,
fmt.Sprintf("%v.lambda_handler", funcName),
role.Arn, zipPackage)
log.Printf("Your function is %v.", funcState)
log.Println(strings.Repeat("-", 88))
return funcName
}

// InvokeIncrement invokes a Lambda function that increments a number. The
function
// parameters are contained in a Go struct that is used to serialize the
parameters to
// a JSON payload that is passed to the function.
// The result payload is deserialized into a Go struct that contains an int
value.
func (scenario GetStartedFunctionsScenario) InvokeIncrement(ctx context.Context,
funcName string) {
parameters := actions.IncrementParameters{Action: "increment"}
log.Println("Let's invoke our function. This function increments a number.")

```

```

parameters.Number = scenario.questioner.AskInt("Enter a number to increment:",
demotools.NotEmpty{})
log.Printf("Invoking %v with %v...\n", funcName, parameters.Number)
invokeOutput := scenario.functionWrapper.Invoke(ctx, funcName, parameters,
false)
var payload actions.LambdaResultInt
err := json.Unmarshal(invokeOutput.Payload, &payload)
if err != nil {
    log.Panicf("Couldn't unmarshal payload from invoking %v. Here's why: %v\n",
        funcName, err)
}
log.Printf("Invoking %v with %v returned %v.\n", funcName, parameters.Number,
payload)
log.Println(strings.Repeat("-", 88))
}

// UpdateFunction updates the code for a Lambda function by uploading a simple
arithmetic
// calculator written in Python. The code for the Python handler is packaged as a
// []byte in .zip format.
// After the code is updated, the configuration is also updated with a new log
// level that instructs the handler to log additional information.
func (scenario GetStartedFunctionsScenario) UpdateFunction(ctx context.Context,
funcName string) {
    log.Println("Let's update the function to an arithmetic calculator.\n" +
        "The function uses the 'lambda_handler_calculator.py' script found in the \n" +
        "'handlers' directory of this project.")
    scenario.questioner.Ask("Press Enter when you're ready.")
    log.Println("Creating deployment package...")
    zipPackage :=
scenario.helper.CreateDeploymentPackage("lambda_handler_calculator.py",
    fmt.Sprintf("%v.py", funcName))
    log.Println("...and updating the Lambda function and waiting for it to be
ready.")
    funcState := scenario.functionWrapper.UpdateFunctionCode(ctx, funcName,
zipPackage)
    log.Printf("Updated function %v. Its current state is %v.", funcName, funcState)
    log.Println("This function uses an environment variable to control logging
level.")
    log.Println("Let's set it to DEBUG to get the most logging.")
    scenario.functionWrapper.UpdateFunctionConfiguration(ctx, funcName,
        map[string]string{"LOG_LEVEL": "DEBUG"})
    log.Println(strings.Repeat("-", 88))
}

```

```

// InvokeCalculator invokes the Lambda calculator function. The parameters are
// stored in a
// Go struct that is used to serialize the parameters to a JSON payload. That
// payload is then passed
// to the function.
// The result payload is deserialized to a Go struct that stores the result as
// either an
// int or float32, depending on the kind of operation that was specified.
func (scenario GetStartedFunctionsScenario) InvokeCalculator(ctx context.Context,
funcName string) {
    wantInvoke := true
    choices := []string{"plus", "minus", "times", "divided-by"}
    for wantInvoke {
        choice := scenario.questioner.AskChoice("Select an arithmetic operation:\n",
choices)
        x := scenario.questioner.AskInt("Enter a value for x:", demotools.NotEmpty{})
        y := scenario.questioner.AskInt("Enter a value for y:", demotools.NotEmpty{})
        log.Printf("Invoking %v %v %v...", x, choices[choice], y)
        calcParameters := actions.CalculatorParameters{
            Action: choices[choice],
            X:      x,
            Y:      y,
        }
        invokeOutput := scenario.functionWrapper.Invoke(ctx, funcName, calcParameters,
true)
        var payload any
        if choice == 3 { // divide-by results in a float.
            payload = actions.LambdaResultFloat{}
        } else {
            payload = actions.LambdaResultInt{}
        }
        err := json.Unmarshal(invokeOutput.Payload, &payload)
        if err != nil {
            log.Panicf("Couldn't unmarshal payload from invoking %v. Here's why: %v\n",
funcName, err)
        }
        log.Printf("Invoking %v with %v %v %v returned %v.\n", funcName,
calcParameters.X, calcParameters.Action, calcParameters.Y, payload)
        scenario.questioner.Ask("Press Enter to see the logs from the call.")
        logRes, err := base64.StdEncoding.DecodeString(*invokeOutput.LogResult)
        if err != nil {
            log.Panicf("Couldn't decode log result. Here's why: %v\n", err)
        }
    }
}

```



```

    log.Println(string(logRes))
    wantInvoke = scenario.questioner.AskBool("Do you want to calculate again? (y/n)", "y")
}
log.Println(strings.Repeat("-", 88))
}

// ListFunctions lists up to the specified number of functions for your account.
func (scenario GetStartedFunctionsScenario) ListFunctions(ctx context.Context) {
    count := scenario.questioner.AskInt(
        "Let's list functions for your account. How many do you want to see?",
        demotools.NotEmpty{})
    functions := scenario.functionWrapper.ListFunctions(ctx, count)
    log.Printf("Found %v functions:", len(functions))
    for _, function := range functions {
        log.Printf("\t%v", *function.FunctionName)
    }
    log.Println(strings.Repeat("-", 88))
}

// Cleanup removes the IAM and Lambda resources created by the example.
func (scenario GetStartedFunctionsScenario) Cleanup(ctx context.Context, role
    *iamtypes.Role, funcName string) {
    if scenario.questioner.AskBool("Do you want to clean up resources created for
    this example? (y/n)",
        "y") {
        iamClient := iam.NewFromConfig(scenario.sdkConfig)
        policiesOutput, err := iamClient.ListAttachedRolePolicies(ctx,
            &iam.ListAttachedRolePoliciesInput{RoleName: role.RoleName})
        if err != nil {
            log.Panicf("Couldn't get policies attached to role %v. Here's why: %v\n",
                *role.RoleName, err)
        }
        for _, policy := range policiesOutput.AttachedPolicies {
            _, err = iamClient.DetachRolePolicy(ctx, &iam.DetachRolePolicyInput{
                PolicyArn: policy.PolicyArn, RoleName: role.RoleName,
            })
            if err != nil {
                log.Panicf("Couldn't detach policy %v from role %v. Here's why: %v\n",
                    *policy.PolicyArn, *role.RoleName, err)
            }
        }
        _, err = iamClient.DeleteRole(ctx, &iam.DeleteRoleInput{RoleName:
            role.RoleName})
    }
}

```

```

    if err != nil {
        log.Panicf("Couldn't delete role %v. Here's why: %v\n", *role.RoleName, err)
    }
    log.Printf("Deleted role %v.\n", *role.RoleName)

    scenario.functionWrapper.DeleteFunction(ctx, funcName)
    log.Printf("Deleted function %v.\n", funcName)
} else {
    log.Println("Okay. Don't forget to delete the resources when you're done with them.")
}
}

// IScenarioHelper abstracts I/O and wait functions from a scenario so that they
// can be mocked for unit testing.
type IScenarioHelper interface {
    Pause(secs int)
    CreateDeploymentPackage(sourceFile string, destinationFile string) *bytes.Buffer
}

// ScenarioHelper lets the caller specify the path to Lambda handler functions.
type ScenarioHelper struct {
    HandlerPath string
}

// Pause waits for the specified number of seconds.
func (helper *ScenarioHelper) Pause(secs int) {
    time.Sleep(time.Duration(secs) * time.Second)
}

// CreateDeploymentPackage creates an AWS Lambda deployment package from a source
// file. The
// deployment package is stored in .zip format in a bytes.Buffer. The buffer can
// be
// used to pass a []byte to Lambda when creating the function.
// The specified destinationFile is the name to give the file when it's deployed
// to Lambda.
func (helper *ScenarioHelper) CreateDeploymentPackage(sourceFile string,
    destinationFile string) *bytes.Buffer {
    var err error
    buffer := &bytes.Buffer{}
    writer := zip.NewWriter(buffer)
    zFile, err := writer.Create(destinationFile)
    if err != nil {

```

```

    log.Panicf("Couldn't create destination archive %v. Here's why: %v\n",
destinationFile, err)
}
sourceBody, err := os.ReadFile(fmt.Sprintf("%v/%v", helper.HandlerPath,
sourceFile))
if err != nil {
    log.Panicf("Couldn't read handler source file %v. Here's why: %v\n",
        sourceFile, err)
} else {
    _, err = zFile.Write(sourceBody)
    if err != nil {
        log.Panicf("Couldn't write handler %v to zip archive. Here's why: %v\n",
            sourceFile, err)
    }
}
err = writer.Close()
if err != nil {
    log.Panicf("Couldn't close zip writer. Here's why: %v\n", err)
}
return buffer
}

```

Creare una struttura che racchiude le singole operazioni Lambda.

```

import (
    "bytes"
    "context"
    "encoding/json"
    "errors"
    "log"
    "time"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/lambda"
    "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {

```

```
LambdaClient *lambda.Client
}

// GetFunction gets data about the Lambda function specified by functionName.
func (wrapper FunctionWrapper) GetFunction(ctx context.Context, functionName
string) types.State {
    var state types.State
    funcOutput, err := wrapper.LambdaClient.GetFunction(ctx,
    &lambda.GetFunctionInput{
        FunctionName: aws.String(functionName),
    })
    if err != nil {
        log.Panicf("Couldn't get function %v. Here's why: %v\n", functionName, err)
    } else {
        state = funcOutput.Configuration.State
    }
    return state
}

// CreateFunction creates a new Lambda function from code contained in the
zipPackage
// buffer. The specified handlerName must match the name of the file and function
// contained in the uploaded code. The role specified by iamRoleArn is assumed by
// Lambda and grants specific permissions.
// When the function already exists, types.StateActive is returned.
// When the function is created, a lambda.FunctionActiveV2Waiter is used to wait
until the
// function is active.
func (wrapper FunctionWrapper) CreateFunction(ctx context.Context, functionName
string, handlerName string,
iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
    var state types.State
    _, err := wrapper.LambdaClient.CreateFunction(ctx, &lambda.CreateFunctionInput{
        Code:          &types.FunctionCode{ZipFile: zipPackage.Bytes()},
        FunctionName:   aws.String(functionName),
        Role:          iamRoleArn,
        Handler:       aws.String(handlerName),
        Publish:       true,
        Runtime:       types.RuntimePython39,
    })
    if err != nil {
```

```

var resConflict *types.ResourceConflictException
if errors.As(err, &resConflict) {
    log.Printf("Function %v already exists.\n", functionName)
    state = types.StateActive
} else {
    log.Panicf("Couldn't create function %v. Here's why: %v\n", functionName, err)
}
} else {
    waiter := lambda.NewFunctionActiveV2Waiter(wrapper.LambdaClient)
    funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
        FunctionName: aws.String(functionName)}, 1*time.Minute)
    if err != nil {
        log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
functionName, err)
    } else {
        state = funcOutput.Configuration.State
    }
}
return state
}

// UpdateFunctionCode updates the code for the Lambda function specified by
// functionName.
// The existing code for the Lambda function is entirely replaced by the code in
// the
// zipPackage buffer. After the update action is called, a
// lambda.FunctionUpdatedV2Waiter
// is used to wait until the update is successful.
func (wrapper FunctionWrapper) UpdateFunctionCode(ctx context.Context,
functionName string, zipPackage *bytes.Buffer) types.State {
    var state types.State
    _, err := wrapper.LambdaClient.UpdateFunctionCode(ctx,
&lambda.UpdateFunctionCodeInput{
    FunctionName: aws.String(functionName), ZipFile: zipPackage.Bytes(),
})
    if err != nil {
        log.Panicf("Couldn't update code for function %v. Here's why: %v\n",
functionName, err)
    } else {
        waiter := lambda.NewFunctionUpdatedV2Waiter(wrapper.LambdaClient)
        funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
            FunctionName: aws.String(functionName)}, 1*time.Minute)

```

```
    if err != nil {
        log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
            functionName, err)
    } else {
        state = funcOutput.Configuration.State
    }
}
return state
}

// UpdateFunctionConfiguration updates a map of environment variables configured
// for
// the Lambda function specified by functionName.
func (wrapper FunctionWrapper) UpdateFunctionConfiguration(ctx context.Context,
    functionName string, envVars map[string]string) {
    _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(ctx,
        &lambda.UpdateFunctionConfigurationInput{
            FunctionName: aws.String(functionName),
            Environment: &types.Environment{Variables: envVars},
        })
    if err != nil {
        log.Panicf("Couldn't update configuration for %v. Here's why: %v",
            functionName, err)
    }
}

// ListFunctions lists up to maxItems functions for the account. This function
// uses a
// lambda.ListFunctionsPaginator to paginate the results.
func (wrapper FunctionWrapper) ListFunctions(ctx context.Context, maxItems int)
    []types.FunctionConfiguration {
    var functions []types.FunctionConfiguration
    paginator := lambda.NewListFunctionsPaginator(wrapper.LambdaClient,
        &lambda.ListFunctionsInput{
            MaxItems: aws.Int32(int32(maxItems)),
        })
    for paginator.HasMorePages() && len(functions) < maxItems {
        pageOutput, err := paginator.NextPage(ctx)
        if err != nil {
            log.Panicf("Couldn't list functions for your account. Here's why: %v\n", err)
        }
    }
}
```

```
    }
    functions = append(functions, pageOutput.Functions...)
  }
  return functions
}

// DeleteFunction deletes the Lambda function specified by functionName.
func (wrapper FunctionWrapper) DeleteFunction(ctx context.Context, functionName
  string) {
  _, err := wrapper.LambdaClient.DeleteFunction(ctx, &lambda.DeleteFunctionInput{
    FunctionName: aws.String(functionName),
  })
  if err != nil {
    log.Panicf("Couldn't delete function %v. Here's why: %v\n", functionName, err)
  }
}

// Invoke invokes the Lambda function specified by functionName, passing the
  parameters
// as a JSON payload. When getLog is true, types.LogTypeTail is specified, which
  tells
// Lambda to include the last few log lines in the returned result.
func (wrapper FunctionWrapper) Invoke(ctx context.Context, functionName string,
  parameters any, getLog bool) *lambda.InvokeOutput {
  logType := types.LogTypeNone
  if getLog {
    logType = types.LogTypeTail
  }
  payload, err := json.Marshal(parameters)
  if err != nil {
    log.Panicf("Couldn't marshal parameters to JSON. Here's why %v\n", err)
  }
  invokeOutput, err := wrapper.LambdaClient.Invoke(ctx, &lambda.InvokeInput{
    FunctionName: aws.String(functionName),
    LogType:      logType,
    Payload:      payload,
  })
  if err != nil {
    log.Panicf("Couldn't invoke function %v. Here's why: %v\n", functionName, err)
  }
}
```

```
    return invokeOutput
}

// IncrementParameters is used to serialize parameters to the increment Lambda
// handler.
type IncrementParameters struct {
    Action string `json:"action"`
    Number int    `json:"number"`
}

// CalculatorParameters is used to serialize parameters to the calculator Lambda
// handler.
type CalculatorParameters struct {
    Action string `json:"action"`
    X      int    `json:"x"`
    Y      int    `json:"y"`
}

// LambdaResultInt is used to deserialize an int result from a Lambda handler.
type LambdaResultInt struct {
    Result int `json:"result"`
}

// LambdaResultFloat is used to deserialize a float32 result from a Lambda
// handler.
type LambdaResultFloat struct {
    Result float32 `json:"result"`
}
```

Definire un gestore Lambda che incrementa un numero.

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    """
```



```
    Accepts an action and a single number, performs the specified action on the
    number,
    and returns the result. The only allowable action is 'increment'.

    :param event: The event dict that contains the parameters sent when the
    function
                   is invoked.
    :param context: The context in which the function is called.
    :return: The result of the action.
    """
    result = None
    action = event.get("action")
    if action == "increment":
        result = event.get("number", 0) + 1
        logger.info("Calculated result of %s", result)
    else:
        logger.error("%s is not a valid action.", action)

    response = {"result": result}
    return response
```

Definire un secondo gestore Lambda che esegue operazioni aritmetiche.

```
import logging
import os

logger = logging.getLogger()

# Define a list of Python lambda functions that are called by this AWS Lambda
function.
ACTIONS = {
    "plus": lambda x, y: x + y,
    "minus": lambda x, y: x - y,
    "times": lambda x, y: x * y,
    "divided-by": lambda x, y: x / y,
}

def lambda_handler(event, context):
```

```

"""
    Accepts an action and two numbers, performs the specified action on the
    numbers,
    and returns the result.

    :param event: The event dict that contains the parameters sent when the
    function
                   is invoked.
    :param context: The context in which the function is called.
    :return: The result of the specified action.
"""

# Set the log level based on a variable configured in the Lambda environment.
logger.setLevel(os.environ.get("LOG_LEVEL", logging.INFO))
logger.debug("Event: %s", event)

action = event.get("action")
func = ACTIONS.get(action)
x = event.get("x")
y = event.get("y")
result = None
try:
    if func is not None and x is not None and y is not None:
        result = func(x, y)
        logger.info("%s %s %s is %s", x, action, y, result)
    else:
        logger.error("I can't calculate %s %s %s.", x, action, y)
except ZeroDivisionError:
    logger.warning("I can't divide %s by 0!", x)

response = {"result": result}
return response

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)

- [ListFunctions](#)
- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/*
 * Lambda function names appear as:
 *
 * arn:aws:lambda:us-west-2:335556666777:function:HelloFunction
 *
 * To find this value, look at the function in the AWS Management Console.
 *
 * Before running this Java code example, set up your development environment,
 * including your credentials.
 *
 * For more information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 *
 * This example performs the following tasks:
 *
 * 1. Creates an AWS Lambda function.
 * 2. Gets a specific AWS Lambda function.
 * 3. Lists all Lambda functions.
 * 4. Invokes a Lambda function.
 * 5. Updates the Lambda function code and invokes it again.
 * 6. Updates a Lambda function's configuration value.
 * 7. Deletes a Lambda function.
 */

public class LambdaScenario {
```

```

    public static final String DASHES = new String(new char[80]).replace("\0",
    "-");

    public static void main(String[] args) throws InterruptedException {
        final String usage = ""

        Usage:
            <functionName> <role> <handler> <bucketName> <key>\s

        Where:
            functionName - The name of the Lambda function.\s
            role - The AWS Identity and Access Management (IAM) service role
that has Lambda permissions.\s
            handler - The fully qualified method name (for example,
example.Handler::handleRequest).\s
            bucketName - The Amazon Simple Storage Service (Amazon S3) bucket
name that contains the .zip or .jar used to update the Lambda function's code.\s
            key - The Amazon S3 key name that represents the .zip or .jar
(for example, LambdaHello-1.0-SNAPSHOT.jar).
            """;

        if (args.length != 5) {
            System.out.println(usage);
            return;
        }

        String functionName = args[0];
        String role = args[1];
        String handler = args[2];
        String bucketName = args[3];
        String key = args[4];
        LambdaClient awsLambda = LambdaClient.builder()
            .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the AWS Lambda Basics scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Create an AWS Lambda function.");
        String funArn = createLambdaFunction(awsLambda, functionName, key,
bucketName, role, handler);
        System.out.println("The AWS Lambda ARN is " + funArn);
        System.out.println(DASHES);

```

```
        System.out.println(DASHES);
        System.out.println("2. Get the " + functionName + " AWS Lambda
function.");
        getFunction(awsLambda, functionName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. List all AWS Lambda functions.");
        listFunctions(awsLambda);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Invoke the Lambda function.");
        System.out.println("*** Sleep for 1 min to get Lambda function ready.");
        Thread.sleep(60000);
        invokeFunction(awsLambda, functionName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Update the Lambda function code and invoke it
again.");
        updateFunctionCode(awsLambda, functionName, bucketName, key);
        System.out.println("*** Sleep for 1 min to get Lambda function ready.");
        Thread.sleep(60000);
        invokeFunction(awsLambda, functionName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Update a Lambda function's configuration value.");
        updateFunctionConfiguration(awsLambda, functionName, handler);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Delete the AWS Lambda function.");
        LambdaScenario.deleteLambdaFunction(awsLambda, functionName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The AWS Lambda scenario completed successfully");
        System.out.println(DASHES);
        awsLambda.close();
    }
```

```

/**
 * Creates a new Lambda function in AWS using the AWS Lambda Java API.
 *
 * @param awsLambda    the AWS Lambda client used to interact with the AWS
Lambda service
 * @param functionName the name of the Lambda function to create
 * @param key          the S3 key of the function code
 * @param bucketName   the name of the S3 bucket containing the function code
 * @param role         the IAM role to assign to the Lambda function
 * @param handler      the fully qualified class name of the function handler
 * @return the Amazon Resource Name (ARN) of the created Lambda function
 */
public static String createLambdaFunction(LambdaClient awsLambda,
                                         String functionName,
                                         String key,
                                         String bucketName,
                                         String role,
                                         String handler) {

    try {
        LambdaWaiter waiter = awsLambda.waiter();
        FunctionCode code = FunctionCode.builder()
            .s3Key(key)
            .s3Bucket(bucketName)
            .build();

        CreateFunctionRequest functionRequest =
CreateFunctionRequest.builder()
            .functionName(functionName)
            .description("Created by the Lambda Java API")
            .code(code)
            .handler(handler)
            .runtime(Runtime.JAVA17)
            .role(role)
            .build();

        // Create a Lambda function using a waiter
        CreateFunctionResponse functionResponse =
awsLambda.createFunction(functionRequest);
        GetFunctionRequest getFunctionRequest = GetFunctionRequest.builder()
            .functionName(functionName)
            .build();

        WaiterResponse<GetFunctionResponse> waiterResponse =
waiter.waitUntilFunctionExists(getFunctionRequest);
    }
}

```

```

        waiterResponse.matched().response().ifPresent(System.out::println);
        return functionResponse.functionArn();

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
    return "";
}

/**
 * Retrieves information about an AWS Lambda function.
 *
 * @param awsLambda an instance of the {@link LambdaClient} class, which
is used to interact with the AWS Lambda service
 * @param functionName the name of the AWS Lambda function to retrieve
information about
 */
public static void getFunction(LambdaClient awsLambda, String functionName) {
    try {
        GetFunctionRequest functionRequest = GetFunctionRequest.builder()
            .functionName(functionName)
            .build();

        GetFunctionResponse response =
awsLambda.getFunction(functionRequest);
        System.out.println("The runtime of this Lambda function is " +
response.configuration().runtime());

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

/**
 * Lists the AWS Lambda functions associated with the current AWS account.
 *
 * @param awsLambda an instance of the {@link LambdaClient} class, which is
used to interact with the AWS Lambda service
 *
 * @throws LambdaException if an error occurs while interacting with the AWS
Lambda service
 */

```

```
public static void listFunctions(LambdaClient awsLambda) {
    try {
        ListFunctionsResponse functionResult = awsLambda.listFunctions();
        List<FunctionConfiguration> list = functionResult.functions();
        for (FunctionConfiguration config : list) {
            System.out.println("The function name is " +
config.functionName());
        }

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

/**
 * Invokes a specific AWS Lambda function.
 *
 * @param awsLambda an instance of {@link LambdaClient} to interact with
the AWS Lambda service
 * @param functionName the name of the AWS Lambda function to be invoked
 */
public static void invokeFunction(LambdaClient awsLambda, String
functionName) {
    InvokeResponse res;
    try {
        // Need a SdkBytes instance for the payload.
        JSONObject jsonObj = new JSONObject();
        jsonObj.put("inputValue", "2000");
        String json = jsonObj.toString();
        SdkBytes payload = SdkBytes.fromUtf8String(json);

        InvokeRequest request = InvokeRequest.builder()
            .functionName(functionName)
            .payload(payload)
            .build();

        res = awsLambda.invoke(request);
        String value = res.payload().asUtf8String();
        System.out.println(value);

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```



```

    }
}

/**
 * Updates the code for an AWS Lambda function.
 *
 * @param awsLambda the AWS Lambda client
 * @param functionName the name of the Lambda function to update
 * @param bucketName the name of the S3 bucket where the function code is
located
 * @param key the key (file name) of the function code in the S3 bucket
 * @throws LambdaException if there is an error updating the function code
 */
public static void updateFunctionCode(LambdaClient awsLambda, String
functionName, String bucketName, String key) {
    try {
        LambdaWaiter waiter = awsLambda.waiter();
        UpdateFunctionCodeRequest functionCodeRequest =
UpdateFunctionCodeRequest.builder()
            .functionName(functionName)
            .publish(true)
            .s3Bucket(bucketName)
            .s3Key(key)
            .build();

        UpdateFunctionCodeResponse response =
awsLambda.updateFunctionCode(functionCodeRequest);
        GetFunctionConfigurationRequest getFunctionConfigRequest =
GetFunctionConfigurationRequest.builder()
            .functionName(functionName)
            .build();

        WaiterResponse<GetFunctionConfigurationResponse> waiterResponse =
waiter
            .waitUntilFunctionUpdated(getFunctionConfigRequest);
        waiterResponse.matched().response().ifPresent(System.out::println);
        System.out.println("The last modified value is " +
response.lastModified());

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

```

```
/**
 * Updates the configuration of an AWS Lambda function.
 *
 * @param awsLambda      the {@link LambdaClient} instance to use for the AWS
Lambda operation
 * @param functionName  the name of the AWS Lambda function to update
 * @param handler        the new handler for the AWS Lambda function
 *
 * @throws LambdaException if there is an error while updating the function
configuration
 */
public static void updateFunctionConfiguration(LambdaClient awsLambda, String
functionName, String handler) {
    try {
        UpdateFunctionConfigurationRequest configurationRequest =
UpdateFunctionConfigurationRequest.builder()
            .functionName(functionName)
            .handler(handler)
            .runtime(Runtime.JAVA17)
            .build();

        awsLambda.updateFunctionConfiguration(configurationRequest);

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

/**
 * Deletes an AWS Lambda function.
 *
 * @param awsLambda      an instance of the {@link LambdaClient} class, which
is used to interact with the AWS Lambda service
 * @param functionName  the name of the Lambda function to be deleted
 *
 * @throws LambdaException if an error occurs while deleting the Lambda
function
 */
public static void deleteLambdaFunction(LambdaClient awsLambda, String
functionName) {
    try {
        DeleteFunctionRequest request = DeleteFunctionRequest.builder()
```

```
        .functionName(functionName)
        .build();

    awsLambda.deleteFunction(request);
    System.out.println("The " + functionName + " function was deleted");

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for Java 2.x .
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Crea un ruolo AWS Identity and Access Management (IAM) che conceda a Lambda l'autorizzazione di scrittura nei log.

```
logger.log(`Creating role (${NAME_ROLE_LAMBDA})...`);
```

```
const response = await createRole(NAME_ROLE_LAMBDA);

import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

Creare una funzione Lambda e caricare il codice del gestore.

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

Invocare la funzione con un singolo parametro e ottenere i risultati.

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

Aggiornare il codice della funzione e configurare il suo ambiente Lambda con una variabile di ambiente.

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};

const updateFunctionConfiguration = async (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });

  const result = await client.send(command);
  await waitFunctionUpdated({ FunctionName: funcName });
  return result;
};
```

```
};
```

Elencare le funzioni per l'account.

```
const listFunctions = () => {  
  const client = new LambdaClient({});  
  const command = new ListFunctionsCommand({});  
  
  return client.send(command);  
};
```

Eliminare il ruolo IAM e la funzione Lambda.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";  
  
const client = new IAMClient({});  
  
/**  
 *  
 * @param {string} roleName  
 */  
export const deleteRole = (roleName) => {  
  const command = new DeleteRoleCommand({ RoleName: roleName });  
  return client.send(command);  
};  
  
/**  
 * @param {string} funcName  
 */  
const deleteFunction = (funcName) => {  
  const client = new LambdaClient({});  
  const command = new DeleteFunctionCommand({ FunctionName: funcName });  
  return client.send(command);  
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per JavaScript .
  - [CreateFunction](#)
  - [DeleteFunction](#)

- [GetFunction](#)
- [Invoke](#)
- [ListFunctions](#)
- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)

## Kotlin

### SDK per Kotlin

#### Note

C'è altro da fare. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <functionName> <role> <handler> <bucketName> <updatedBucketName>
    <key>

    Where:
        functionName - The name of the AWS Lambda function.
        role - The AWS Identity and Access Management (IAM) service role that
        has AWS Lambda permissions.
        handler - The fully qualified method name (for example,
        example.Handler::handleRequest).
        bucketName - The Amazon Simple Storage Service (Amazon S3) bucket
        name that contains the ZIP or JAR used for the Lambda function's code.
        updatedBucketName - The Amazon S3 bucket name that contains the .zip
        or .jar used to update the Lambda function's code.
        key - The Amazon S3 key name that represents the .zip or .jar file
        (for example, LambdaHello-1.0-SNAPSHOT.jar).
    """

    if (args.size != 6) {
        println(usage)
        exitProcess(1)
    }
}
```

```
val functionName = args[0]
val role = args[1]
val handler = args[2]
val bucketName = args[3]
val updatedBucketName = args[4]
val key = args[5]

println("Creating a Lambda function named $functionName.")
val funArn = createScFunction(functionName, bucketName, key, handler, role)
println("The AWS Lambda ARN is $funArn")

// Get a specific Lambda function.
println("Getting the $functionName AWS Lambda function.")
getFunction(functionName)

// List the Lambda functions.
println("Listing all AWS Lambda functions.")
listFunctionsSc()

// Invoke the Lambda function.
println("*** Invoke the Lambda function.")
invokeFunctionSc(functionName)

// Update the AWS Lambda function code.
println("*** Update the Lambda function code.")
updateFunctionCode(functionName, updatedBucketName, key)

// println("*** Invoke the function again after updating the code.")
invokeFunctionSc(functionName)

// Update the AWS Lambda function configuration.
println("Update the run time of the function.")
updateFunctionConfiguration(functionName, handler)

// Delete the AWS Lambda function.
println("Delete the AWS Lambda function.")
delFunction(functionName)
}

suspend fun createScFunction(
    myFunctionName: String,
    s3BucketName: String,
    myS3Key: String,
```



```

        myHandler: String,
        myRole: String,
    ): String {
        val functionCode =
            FunctionCode {
                s3Bucket = s3BucketName
                s3Key = myS3Key
            }

        val request =
            CreateFunctionRequest {
                functionName = myFunctionName
                code = functionCode
                description = "Created by the Lambda Kotlin API"
                handler = myHandler
                role = myRole
                runtime = Runtime.Java17
            }

        // Create a Lambda function using a waiter
        LambdaClient { region = "us-east-1" }.use { awsLambda ->
            val functionResponse = awsLambda.createFunction(request)
            awsLambda.waitForFunctionActive {
                functionName = myFunctionName
            }
            return functionResponse.functionArn.toString()
        }
    }

suspend fun getFunction(functionNameVal: String) {
    val functionRequest =
        GetFunctionRequest {
            functionName = functionNameVal
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        val response = awsLambda.getFunction(functionRequest)
        println("The runtime of this Lambda function is
        ${response.configuration?.runtime}")
    }
}

suspend fun listFunctionsSc() {
    val request =

```

```

        ListFunctionsRequest {
            maxItems = 10
        }

        LambdaClient { region = "us-east-1" }.use { awsLambda ->
            val response = awsLambda.listFunctions(request)
            response.functions?.forEach { function ->
                println("The function name is ${function.functionName}")
            }
        }
    }

suspend fun invokeFunctionSc(functionNameVal: String) {
    val json = """"{"inputValue":"1000"}""""
    val byteArray = json.trimIndent().encodeToByteArray()
    val request =
        InvokeRequest {
            functionName = functionNameVal
            payload = byteArray
            logType = LogType.Tail
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        val res = awsLambda.invoke(request)
        println("The function payload is
        ${res.payload?.toString(Charsets.UTF_8)}")
    }
}

suspend fun updateFunctionCode(
    functionNameVal: String?,
    bucketName: String?,
    key: String?,
) {
    val functionCodeRequest =
        UpdateFunctionCodeRequest {
            functionName = functionNameVal
            publish = true
            s3Bucket = bucketName
            s3Key = key
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        val response = awsLambda.updateFunctionCode(functionCodeRequest)
    }
}

```

```
        awsLambda.waitForFunctionUpdated {
            functionName = functionNameVal
        }
        println("The last modified value is " + response.lastModified)
    }
}

suspend fun updateFunctionConfiguration(
    functionNameVal: String?,
    handlerVal: String?,
) {
    val configurationRequest =
        UpdateFunctionConfigurationRequest {
            functionName = functionNameVal
            handler = handlerVal
            runtime = Runtime.Java17
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        awsLambda.updateFunctionConfiguration(configurationRequest)
    }
}

suspend fun delFunction(myFunctionName: String) {
    val request =
        DeleteFunctionRequest {
            functionName = myFunctionName
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        awsLambda.deleteFunction(request)
        println("$myFunctionName was deleted")
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Kotlin.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)

- [ListFunctions](#)
- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)

## PHP

### SDK per PHP

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
namespace Lambda;

use Aws\S3\S3Client;
use GuzzleHttp\Psr7\Stream;
use Iam\IAMService;

class GettingStartedWithLambda
{
    public function run()
    {
        echo("\n");
        echo("-----\n");
        print("Welcome to the AWS Lambda getting started demo using PHP!\n");
        echo("-----\n");

        $clientArgs = [
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
        ];
        $uniqid = uniqid();

        $iamService = new IAMService();
        $s3client = new S3Client($clientArgs);
        $lambdaService = new LambdaService();

        echo "First, let's create a role to run our Lambda code.\n";
```

```

$roleName = "test-lambda-role-$uniqid";
$rolePolicyDocument = "{
  \"Version\": \"2012-10-17\",
  \"Statement\": [
    {
      \"Effect\": \"Allow\",
      \"Principal\": {
        \"Service\": \"lambda.amazonaws.com\"
      },
      \"Action\": \"sts:AssumeRole\"
    }
  ]
}";
$role = $iamService->createRole($roleName, $rolePolicyDocument);
echo "Created role {$role['RoleName']}.\\n";

$iamService->attachRolePolicy(
  $role['RoleName'],
  "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
);
echo "Attached the AWSLambdaBasicExecutionRole to {$role['RoleName']}.\\n";

\\n";

echo "\\nNow let's create an S3 bucket and upload our Lambda code there.\\n";

$bucketName = "amzn-s3-demo-bucket-$uniqid";
$s3client->createBucket([
  'Bucket' => $bucketName,
]);
echo "Created bucket $bucketName.\\n";

$functionName = "doc_example_lambda_$uniqid";
$codeBasic = __DIR__ . "/lambda_handler_basic.zip";
$handler = "lambda_handler_basic";
$file = file_get_contents($codeBasic);
$s3client->putObject([
  'Bucket' => $bucketName,
  'Key' => $functionName,
  'Body' => $file,
]);
echo "Uploaded the Lambda code.\\n";

$createLambdaFunction = $lambdaService->createFunction($functionName,
  $role, $bucketName, $handler);

```

```

    // Wait until the function has finished being created.
    do {
        $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
        } while ($getLambdaFunction['Configuration']['State'] == "Pending");
        echo "Created Lambda function {$getLambdaFunction['Configuration']
['FunctionName']}.\\n";

        sleep(1);

        echo "\\nOk, let's invoke that Lambda code.\\n";
        $basicParams = [
            'action' => 'increment',
            'number' => 3,
        ];
        /** @var Stream $invokeFunction */
        $invokeFunction = $lambdaService->invoke($functionName, $basicParams)
['Payload'];
        $result = json_decode($invokeFunction->getContents())->result;
        echo "After invoking the Lambda code with the input of
{$basicParams['number']} we received $result.\\n";

        echo "\\nSince that's working, let's update the Lambda code.\\n";
        $codeCalculator = "lambda_handler_calculator.zip";
        $handlerCalculator = "lambda_handler_calculator";
        echo "First, put the new code into the S3 bucket.\\n";
        $file = file_get_contents($codeCalculator);
        $s3client->putObject([
            'Bucket' => $bucketName,
            'Key' => $functionName,
            'Body' => $file,
        ]);
        echo "New code uploaded.\\n";

        $lambdaService->updateFunctionCode($functionName, $bucketName,
$functionName);
        // Wait for the Lambda code to finish updating.
        do {
            $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
            } while ($getLambdaFunction['Configuration']['LastUpdateStatus'] !==
"Successful");
            echo "New Lambda code uploaded.\\n";

```

```

    $environment = [
        'Variable' => ['Variables' => ['LOG_LEVEL' => 'DEBUG']],
    ];
    $lambdaService->updateFunctionConfiguration($functionName,
$handlerCalculator, $environment);
    do {
        $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
    } while ($getLambdaFunction['Configuration']['LastUpdateStatus'] !==
"Successful");
    echo "Lambda code updated with new handler and a LOG_LEVEL of DEBUG for
more information.\n";

    echo "Invoke the new code with some new data.\n";
    $calculatorParams = [
        'action' => 'plus',
        'x' => 5,
        'y' => 4,
    ];
    $invokeFunction = $lambdaService->invoke($functionName,
$calculatorParams, "Tail");
    $result = json_decode($invokeFunction['Payload']->getContents())->result;
    echo "Indeed, {$calculatorParams['x']} + {$calculatorParams['y']} does
equal $result.\n";
    echo "Here's the extra debug info: ";
    echo base64_decode($invokeFunction['LogResult']) . "\n";

    echo "\nBut what happens if you try to divide by zero?\n";
    $divZeroParams = [
        'action' => 'divide',
        'x' => 5,
        'y' => 0,
    ];
    $invokeFunction = $lambdaService->invoke($functionName, $divZeroParams,
"Tail");
    $result = json_decode($invokeFunction['Payload']->getContents())->result;
    echo "You get a |$result| result.\n";
    echo "And an error message: ";
    echo base64_decode($invokeFunction['LogResult']) . "\n";

    echo "\nHere's all the Lambda functions you have in this Region:\n";
    $listLambdaFunctions = $lambdaService->listFunctions(5);
    $allLambdaFunctions = $listLambdaFunctions['Functions'];
    $next = $listLambdaFunctions->get('NextMarker');

```

```

        while ($next != false) {
            $listLambdaFunctions = $lambdaService->listFunctions(5, $next);
            $next = $listLambdaFunctions->get('NextMarker');
            $allLambdaFunctions = array_merge($allLambdaFunctions,
$listLambdaFunctions['Functions']);
        }
        foreach ($allLambdaFunctions as $function) {
            echo "{$function['FunctionName']}\n";
        }

        echo "\n\nAnd don't forget to clean up your data!\n";

        $lambdaService->deleteFunction($functionName);
        echo "Deleted Lambda function.\n";
        $iamService->deleteRole($role['RoleName']);
        echo "Deleted Role.\n";
        $deleteObjects = $s3client->listObjectsV2([
            'Bucket' => $bucketName,
        ]);
        $deleteObjects = $s3client->deleteObjects([
            'Bucket' => $bucketName,
            'Delete' => [
                'Objects' => $deleteObjects['Contents'],
            ]
        ]);
        echo "Deleted all objects from the S3 bucket.\n";
        $s3client->deleteBucket(['Bucket' => $bucketName]);
        echo "Deleted the bucket.\n";
    }
}

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per PHP .
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)



- [UpdateFunctionConfiguration](#)

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Definire un gestore Lambda che incrementa un numero.

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    """
    Accepts an action and a single number, performs the specified action on the
    number,
    and returns the result. The only allowable action is 'increment'.

    :param event: The event dict that contains the parameters sent when the
    function
                   is invoked.
    :param context: The context in which the function is called.
    :return: The result of the action.
    """
    result = None
    action = event.get("action")
    if action == "increment":
        result = event.get("number", 0) + 1
        logger.info("Calculated result of %s", result)
    else:
        logger.error("%s is not a valid action.", action)

    response = {"result": result}
    return response
```

Definire un secondo gestore Lambda che esegue operazioni aritmetiche.

```
import logging
import os

logger = logging.getLogger()

# Define a list of Python lambda functions that are called by this AWS Lambda
function.
ACTIONS = {
    "plus": lambda x, y: x + y,
    "minus": lambda x, y: x - y,
    "times": lambda x, y: x * y,
    "divided-by": lambda x, y: x / y,
}

def lambda_handler(event, context):
    """
    Accepts an action and two numbers, performs the specified action on the
    numbers,
    and returns the result.

    :param event: The event dict that contains the parameters sent when the
    function
        is invoked.
    :param context: The context in which the function is called.
    :return: The result of the specified action.
    """
    # Set the log level based on a variable configured in the Lambda environment.
    logger.setLevel(os.environ.get("LOG_LEVEL", logging.INFO))
    logger.debug("Event: %s", event)

    action = event.get("action")
    func = ACTIONS.get(action)
    x = event.get("x")
    y = event.get("y")
    result = None
```

```

try:
    if func is not None and x is not None and y is not None:
        result = func(x, y)
        logger.info("%s %s %s is %s", x, action, y, result)
    else:
        logger.error("I can't calculate %s %s %s.", x, action, y)
except ZeroDivisionError:
    logger.warning("I can't divide %s by 0!", x)

response = {"result": result}
return response

```

Creare funzioni che eseguono il wrap delle operazioni Lambda.

```

class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    @staticmethod
    def create_deployment_package(source_file, destination_file):
        """
        Creates a Lambda deployment package in .zip format in an in-memory
        buffer. This
        buffer can be passed directly to Lambda when creating the function.

        :param source_file: The name of the file that contains the Lambda handler
            function.
        :param destination_file: The name to give the file when it's deployed to
            Lambda.
        :return: The deployment package.
        """
        buffer = io.BytesIO()
        with zipfile.ZipFile(buffer, "w") as zipped:
            zipped.write(source_file, destination_file)
        buffer.seek(0)
        return buffer.read()

    def get_iam_role(self, iam_role_name):

```

```

"""
Get an AWS Identity and Access Management (IAM) role.

:param iam_role_name: The name of the role to retrieve.
:return: The IAM role.
"""
role = None
try:
    temp_role = self.iam_resource.Role(iam_role_name)
    temp_role.load()
    role = temp_role
    logger.info("Got IAM role %s", role.name)
except ClientError as err:
    if err.response["Error"]["Code"] == "NoSuchEntity":
        logger.info("IAM role %s does not exist.", iam_role_name)
    else:
        logger.error(
            "Couldn't get IAM role %s. Here's why: %s: %s",
            iam_role_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
return role

def create_iam_role_for_lambda(self, iam_role_name):
    """
    Creates an IAM role that grants the Lambda function basic permissions. If
a
    role with the specified name already exists, it is used for the demo.

    :param iam_role_name: The name of the role to create.
    :return: The role and a value that indicates whether the role is newly
created.
    """
    role = self.get_iam_role(iam_role_name)
    if role is not None:
        return role, False

    lambda_assume_role_policy = {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",

```

```

        "Principal": {"Service": "lambda.amazonaws.com"},
        "Action": "sts:AssumeRole",
    }
],
}
policy_arn = "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"

try:
    role = self.iam_resource.create_role(
        RoleName=iam_role_name,
        AssumeRolePolicyDocument=json.dumps(lambda_assume_role_policy),
    )
    logger.info("Created role %s.", role.name)
    role.attach_policy(PolicyArn=policy_arn)
    logger.info("Attached basic execution policy to role %s.", role.name)
except ClientError as error:
    if error.response["Error"]["Code"] == "EntityAlreadyExists":
        role = self.iam_resource.Role(iam_role_name)
        logger.warning("The role %s already exists. Using it.",
iam_role_name)
    else:
        logger.exception(
            "Couldn't create role %s or attach policy %s.",
            iam_role_name,
            policy_arn,
        )
        raise

    return role, True

def get_function(self, function_name):
    """
    Gets data about a Lambda function.

    :param function_name: The name of the function.
    :return: The function data.
    """
    response = None
    try:
        response =
self.lambda_client.get_function(FunctionName=function_name)
    except ClientError as err:
        if err.response["Error"]["Code"] == "ResourceNotFoundException":

```

```

        logger.info("Function %s does not exist.", function_name)
    else:
        logger.error(
            "Couldn't get function %s. Here's why: %s: %s",
            function_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    return response

def create_function(
    self, function_name, handler_name, iam_role, deployment_package
):
    """
    Deploys a Lambda function.

    :param function_name: The name of the Lambda function.
    :param handler_name: The fully qualified name of the handler function.
    This
                        must include the file name and the function name.
    :param iam_role: The IAM role to use for the function.
    :param deployment_package: The deployment package that contains the
    function
                        code in .zip format.
    :return: The Amazon Resource Name (ARN) of the newly created function.
    """
    try:
        response = self.lambda_client.create_function(
            FunctionName=function_name,
            Description="AWS Lambda doc example",
            Runtime="python3.9",
            Role=iam_role.arn,
            Handler=handler_name,
            Code={"ZipFile": deployment_package},
            Publish=True,
        )
        function_arn = response["FunctionArn"]
        waiter = self.lambda_client.get_waiter("function_active_v2")
        waiter.wait(FunctionName=function_name)
        logger.info(
            "Created function '%s' with ARN: '%s'.",
            function_name,

```

```

        response["FunctionArn"],
    )
except ClientError:
    logger.error("Couldn't create function %s.", function_name)
    raise
else:
    return function_arn

def delete_function(self, function_name):
    """
    Deletes a Lambda function.

    :param function_name: The name of the function to delete.
    """
    try:
        self.lambda_client.delete_function(FunctionName=function_name)
    except ClientError:
        logger.exception("Couldn't delete function %s.", function_name)
        raise

def invoke_function(self, function_name, function_params, get_log=False):
    """
    Invokes a Lambda function.

    :param function_name: The name of the function to invoke.
    :param function_params: The parameters of the function as a dict. This
dict
                           is serialized to JSON before it is sent to
Lambda.
    :param get_log: When true, the last 4 KB of the execution log are
included in
                   the response.
    :return: The response from the function invocation.
    """
    try:
        response = self.lambda_client.invoke(
            FunctionName=function_name,
            Payload=json.dumps(function_params),
            LogType="Tail" if get_log else "None",
        )
        logger.info("Invoked function %s.", function_name)
    except ClientError:

```

```

        logger.exception("Couldn't invoke function %s.", function_name)
        raise
    return response

def update_function_code(self, function_name, deployment_package):
    """
    Updates the code for a Lambda function by submitting a .zip archive that
    contains
    the code for the function.

    :param function_name: The name of the function to update.
    :param deployment_package: The function code to update, packaged as bytes
in
                                .zip format.
    :return: Data about the update, including the status.
    """
    try:
        response = self.lambda_client.update_function_code(
            FunctionName=function_name, ZipFile=deployment_package
        )
    except ClientError as err:
        logger.error(
            "Couldn't update function %s. Here's why: %s: %s",
            function_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return response

def update_function_configuration(self, function_name, env_vars):
    """
    Updates the environment variables for a Lambda function.

    :param function_name: The name of the function to update.
    :param env_vars: A dict of environment variables to update.
    :return: Data about the update, including the status.
    """
    try:
        response = self.lambda_client.update_function_configuration(
            FunctionName=function_name, Environment={"Variables": env_vars}

```



```

    )
except ClientError as err:
    logger.error(
        "Couldn't update function configuration %s. Here's why: %s: %s",
        function_name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return response

def list_functions(self):
    """
    Lists the Lambda functions for the current account.
    """
    try:
        func_paginator = self.lambda_client.get_paginator("list_functions")
        for func_page in func_paginator.paginate():
            for func in func_page["Functions"]:
                print(func["FunctionName"])
                desc = func.get("Description")
                if desc:
                    print(f"\t{desc}")
                    print(f"\t{func['Runtime']}: {func['Handler']}")
    except ClientError as err:
        logger.error(
            "Couldn't list functions. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise

```

Creare una funzione che esegue lo scenario.

```

class UpdateFunctionWaiter(CustomWaiter):
    """A custom waiter that waits until a function is successfully updated."""

```

```

def __init__(self, client):
    super().__init__(
        "UpdateSuccess",
        "GetFunction",
        "Configuration.LastUpdateStatus",
        {"Successful": WaitState.SUCCESS, "Failed": WaitState.FAILURE},
        client,
    )

def wait(self, function_name):
    self._wait(FunctionName=function_name)

def run_scenario(lambda_client, iam_resource, basic_file, calculator_file,
lambda_name):
    """
    Runs the scenario.

    :param lambda_client: A Boto3 Lambda client.
    :param iam_resource: A Boto3 IAM resource.
    :param basic_file: The name of the file that contains the basic Lambda
handler.
    :param calculator_file: The name of the file that contains the calculator
Lambda handler.
    :param lambda_name: The name to give resources created for the scenario, such
as the
                        IAM role and the Lambda function.
    """
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the AWS Lambda getting started with functions demo.")
    print("-" * 88)

    wrapper = LambdaWrapper(lambda_client, iam_resource)

    print("Checking for IAM role for Lambda...")
    iam_role, should_wait = wrapper.create_iam_role_for_lambda(lambda_name)
    if should_wait:
        logger.info("Giving AWS time to create resources...")
        wait(10)

    print(f"Looking for function {lambda_name}...")
    function = wrapper.get_function(lambda_name)

```

```

if function is None:
    print("Zipping the Python script into a deployment package...")
    deployment_package = wrapper.create_deployment_package(
        basic_file, f"{lambda_name}.py"
    )
    print(f"...and creating the {lambda_name} Lambda function.")
    wrapper.create_function(
        lambda_name, f"{lambda_name}.lambda_handler", iam_role,
        deployment_package
    )
else:
    print(f"Function {lambda_name} already exists.")
print("-" * 88)

print(f"Let's invoke {lambda_name}. This function increments a number.")
action_params = {
    "action": "increment",
    "number": q.ask("Give me a number to increment: ", q.is_int),
}
print(f"Invoking {lambda_name}...")
response = wrapper.invoke_function(lambda_name, action_params)
print(
    f"Incrementing {action_params['number']} resulted in "
    f"{json.load(response['Payload'])}"
)
print("-" * 88)

print(f"Let's update the function to an arithmetic calculator.")
q.ask("Press Enter when you're ready.")
print("Creating a new deployment package...")
deployment_package = wrapper.create_deployment_package(
    calculator_file, f"{lambda_name}.py"
)
print(f"...and updating the {lambda_name} Lambda function.")
update_waiter = UpdateFunctionWaiter(lambda_client)
wrapper.update_function_code(lambda_name, deployment_package)
update_waiter.wait(lambda_name)
print(f"This function uses an environment variable to control logging
level.")
print(f"Let's set it to DEBUG to get the most logging.")
wrapper.update_function_configuration(
    lambda_name, {"LOG_LEVEL": logging.getLevelName(logging.DEBUG)}
)

```

```

actions = ["plus", "minus", "times", "divided-by"]
want_invoke = True
while want_invoke:
    print(f"Let's invoke {lambda_name}. You can invoke these actions:")
    for index, action in enumerate(actions):
        print(f"{index + 1}: {action}")
    action_params = {}
    action_index = q.ask(
        "Enter the number of the action you want to take: ",
        q.is_int,
        q.in_range(1, len(actions)),
    )
    action_params["action"] = actions[action_index - 1]
    print(f"You've chosen to invoke 'x {action_params['action']} y'.")
    action_params["x"] = q.ask("Enter a value for x: ", q.is_int)
    action_params["y"] = q.ask("Enter a value for y: ", q.is_int)
    print(f"Invoking {lambda_name}...")
    response = wrapper.invoke_function(lambda_name, action_params, True)
    print(
        f"Calculating {action_params['x']} {action_params['action']} "
        f"{action_params['y']} "
        f"resulted in {json.load(response['Payload'])}"
    )
    q.ask("Press Enter to see the logs from the call.")
    print(base64.b64decode(response["LogResult"]).decode())
    want_invoke = q.ask("That was fun. Shall we do it again? (y/n) ",
q.is_yesno)
    print("-" * 88)

    if q.ask(
        "Do you want to list all of the functions in your account? (y/n) ",
q.is_yesno
    ):
        wrapper.list_functions()
        print("-" * 88)

    if q.ask("Ready to delete the function and role? (y/n) ", q.is_yesno):
        for policy in iam_role.attached_policies.all():
            policy.detach_role(RoleName=iam_role.name)
        iam_role.delete()
        print(f"Deleted role {lambda_name}.")
        wrapper.delete_function(lambda_name)
        print(f"Deleted function {lambda_name}.")

```

```
print("\nThanks for watching!")
print("-" * 88)

if __name__ == "__main__":
    try:
        run_scenario(
            boto3.client("lambda"),
            boto3.resource("iam"),
            "lambda_handler_basic.py",
            "lambda_handler_calculator.py",
            "doc_example_lambda_calculator",
        )
    except Exception:
        logging.exception("Something went wrong with the demo!")
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Python (Boto3).
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Configura le autorizzazioni IAM prerequisite per una funzione Lambda in grado di scrivere log.

```

# Get an AWS Identity and Access Management (IAM) role.
#
# @param iam_role_name: The name of the role to retrieve.
# @param action: Whether to create or destroy the IAM apparatus.
# @return: The IAM role.
def manage_iam(iam_role_name, action)
  case action
  when 'create'
    create_iam_role(iam_role_name)
  when 'destroy'
    destroy_iam_role(iam_role_name)
  else
    raise "Incorrect action provided. Must provide 'create' or 'destroy'"
  end
end

private

def create_iam_role(iam_role_name)
  role_policy = {
    'Version': '2012-10-17',
    'Statement': [
      {
        'Effect': 'Allow',
        'Principal': { 'Service': 'lambda.amazonaws.com' },
        'Action': 'sts:AssumeRole'
      }
    ]
  }
  role = @iam_client.create_role(
    role_name: iam_role_name,
    assume_role_policy_document: role_policy.to_json
  )
  @iam_client.attach_role_policy(
    {
      policy_arn: 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole',
      role_name: iam_role_name
    }
  )
  wait_for_role_to_exist(iam_role_name)
  @logger.debug("Successfully created IAM role: #{role['role']['arn']}")
  sleep(10)
end

```

```

    [role, role_policy.to_json]
  end

  def destroy_iam_role(iam_role_name)
    @iam_client.detach_role_policy(
      {
        policy_arn: 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole',
        role_name: iam_role_name
      }
    )
    @iam_client.delete_role(role_name: iam_role_name)
    @logger.debug("Detached policy & deleted IAM role: #{iam_role_name}")
  end

  def wait_for_role_to_exist(iam_role_name)
    @iam_client.wait_until(:role_exists, { role_name: iam_role_name }) do |w|
      w.max_attempts = 5
      w.delay = 5
    end
  end
end

```

Definisci un gestore Lambda che incrementa un numero fornito come parametro di invocazione.

```

require 'logger'

# A function that increments a whole number by one (1) and logs the result.
# Requires a manually-provided runtime parameter, 'number', which must be Int
#
# @param event [Hash] Parameters sent when the function is invoked
# @param context [Hash] Methods and properties that provide information
# about the invocation, function, and execution environment.
# @return incremented_number [String] The incremented number.
def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  log_level = ENV['LOG_LEVEL']
  logger.level = case log_level
                 when 'debug'
                   Logger::DEBUG
                 when 'info'
                   Logger::INFO

```

```

        else
          Logger::ERROR
        end
      logger.debug('This is a debug log message.')
      logger.info('This is an info log message. Code executed successfully!')
      number = event['number'].to_i
      incremented_number = number + 1
      logger.info("You provided #{number.round} and it was incremented to
#{incremented_number.round}")
      incremented_number.round.to_s
    end
  end
end

```

Comprimi la funzione Lambda in un pacchetto di implementazione:

```

# Creates a Lambda deployment package in .zip format.
#
# @param source_file: The name of the object, without suffix, for the Lambda
file and zip.
# @return: The deployment package.
def create_deployment_package(source_file)
  Dir.chdir(File.dirname(__FILE__))
  if File.exist?('lambda_function.zip')
    File.delete('lambda_function.zip')
    @logger.debug('Deleting old zip: lambda_function.zip')
  end
  Zip::File.open('lambda_function.zip', create: true) do |zipfile|
    zipfile.add('lambda_function.rb', "#{source_file}.rb")
  end
  @logger.debug("Zipping #{source_file}.rb into: lambda_function.zip.")
  File.read('lambda_function.zip').to_s
rescue StandardError => e
  @logger.error("There was an error creating deployment package:\n
#{e.message}")
end

```

Crea una nuova funzione Lambda.

```

# Deploys a Lambda function.
#
# @param function_name: The name of the Lambda function.
# @param handler_name: The fully qualified name of the handler function.

```



```

# @param role_arn: The IAM role to use for the function.
# @param deployment_package: The deployment package that contains the function
code in .zip format.
# @return: The Amazon Resource Name (ARN) of the newly created function.
def create_function(function_name, handler_name, role_arn, deployment_package)
  response = @lambda_client.create_function({
    role: role_arn.to_s,
    function_name: function_name,
    handler: handler_name,
    runtime: 'ruby2.7',
    code: {
      zip_file: deployment_package
    },
    environment: {
      variables: {
        'LOG_LEVEL' => 'info'
      }
    }
  })

  @lambda_client.wait_until(:function_active_v2, { function_name:
function_name }) do |w|
    w.max_attempts = 5
    w.delay = 5
  end
  response
rescue Aws::Lambda::Errors::ServiceException => e
  @logger.error("There was an error creating #{function_name}:\n #{e.message}")
rescue Aws::Waiters::Errors::WaiterFailed => e
  @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
end

```

Invocare la tua funzione Lambda con parametri di runtime facoltativi.

```

# Invokes a Lambda function.
# @param function_name [String] The name of the function to invoke.
# @param payload [nil] Payload containing runtime parameters.
# @return [Object] The response from the function invocation.
def invoke_function(function_name, payload = nil)
  params = { function_name: function_name }
  params[:payload] = payload unless payload.nil?
  @lambda_client.invoke(params)

```

```
rescue Aws::Lambda::Errors::ServiceException => e
  @logger.error("There was an error executing #{function_name}:\n
#{e.message}")
end
```

Aggiorna la configurazione della funzione Lambda per inserire una nuova variabile di ambiente.

```
# Updates the environment variables for a Lambda function.
# @param function_name: The name of the function to update.
# @param log_level: The log level of the function.
# @return: Data about the update, including the status.
def update_function_configuration(function_name, log_level)
  @lambda_client.update_function_configuration({
    function_name: function_name,
    environment: {
      variables: {
        'LOG_LEVEL' => log_level
      }
    }
  })

  @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name }) do |w|
    w.max_attempts = 5
    w.delay = 5
  end

  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error updating configurations for
#{function_name}:\n #{e.message}")
  rescue Aws::Waiters::Errors::WaiterFailed => e
    @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
  end
end
```

Aggiorna il codice della funzione Lambda con un pacchetto di implementazione diverso contenente codice diverso.

```
# Updates the code for a Lambda function by submitting a .zip archive that
contains
# the code for the function.
#
```

```

# @param function_name: The name of the function to update.
# @param deployment_package: The function code to update, packaged as bytes in
#                               .zip format.
# @return: Data about the update, including the status.
def update_function_code(function_name, deployment_package)
  @lambda_client.update_function_code(
    function_name: function_name,
    zip_file: deployment_package
  )
  @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name }) do |w|
    w.max_attempts = 5
    w.delay = 5
  end
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error updating function code for:
#{function_name}:\n #{e.message}")
    nil
  rescue Aws::Waiters::Errors::WaiterFailed => e
    @logger.error("Failed waiting for #{function_name} to update:\n
#{e.message}")
  end
end

```

Elenca tutte le funzioni Lambda esistenti utilizzando l'impaginatore integrato.

```

# Lists the Lambda functions for the current account.
def list_functions
  functions = []
  @lambda_client.list_functions.each do |response|
    response['functions'].each do |function|
      functions.append(function['function_name'])
    end
  end
  functions
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error listing functions:\n #{e.message}")
  end
end

```

Elimina una funzione Lambda specifica.

```

# Deletes a Lambda function.

```

```
# @param function_name: The name of the function to delete.
def delete_function(function_name)
  print "Deleting function: #{function_name}..."
  @lambda_client.delete_function(
    function_name: function_name
  )
  print 'Done!'.green
rescue Aws::Lambda::Errors::ServiceException => e
  @logger.error("There was an error deleting #{function_name}:\n #{e.message}")
end
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Ruby .
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Le Cargo.toml con dipendenze utilizzate in questo scenario.

```
[package]
name = "lambda-code-examples"
version = "0.1.0"
edition = "2021"
```

```
# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-sdk-ec2 = { version = "1.3.0" }
aws-sdk-iam = { version = "1.3.0" }
aws-sdk-lambda = { version = "1.3.0" }
aws-sdk-s3 = { version = "1.4.0" }
aws-smithy-types = { version = "1.0.1" }
aws-types = { version = "1.0.1" }
clap = { version = "4.4", features = ["derive"] }
tokio = { version = "1.20.1", features = ["full"] }
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
tracing = "0.1.37"
serde_json = "1.0.94"
anyhow = "1.0.71"
uuid = { version = "1.3.3", features = ["v4"] }
lambda_runtime = "0.8.0"
serde = "1.0.164"
```

Una raccolta di utilità che semplificano le invocazioni a Lambda per questo scenario. Questo file è in `src/ations` formato.rs nella cassa.

```
use anyhow::anyhow;
use aws_sdk_iam::operation::{create_role::CreateRoleError,
    delete_role::DeleteRoleOutput};
use aws_sdk_lambda::{
    operation::{
        delete_function::DeleteFunctionOutput, get_function::GetFunctionOutput,
        invoke::InvokeOutput, list_functions::ListFunctionsOutput,
        update_function_code::UpdateFunctionCodeOutput,
        update_function_configuration::UpdateFunctionConfigurationOutput,
    },
    primitives::ByteStream,
    types::{Environment, FunctionCode, LastUpdateStatus, State},
};
use aws_sdk_s3::{
    error::ErrorMetadata,
    operation::{delete_bucket::DeleteBucketOutput,
        delete_object::DeleteObjectOutput},
```

```

    types::CreateBucketConfiguration,
};
use aws_smithy_types::Blob;
use serde::{ser::SerializeMap, Serialize};
use std::{fmt::Display, path::PathBuf, str::FromStr, time::Duration};
use tracing::{debug, info, warn};

/* Operation describes */
#[derive(Clone, Copy, Debug, Serialize)]
pub enum Operation {
    #[serde(rename = "plus")]
    Plus,
    #[serde(rename = "minus")]
    Minus,
    #[serde(rename = "times")]
    Times,
    #[serde(rename = "divided-by")]
    DividedBy,
}

impl FromStr for Operation {
    type Err = anyhow::Error;

    fn from_str(s: &str) -> Result<Self, Self::Err> {
        match s {
            "plus" => Ok(Operation::Plus),
            "minus" => Ok(Operation::Minus),
            "times" => Ok(Operation::Times),
            "divided-by" => Ok(Operation::DividedBy),
            _ => Err(anyhow!("Unknown operation {s}")),
        }
    }
}

impl Display for Operation {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match self {
            Operation::Plus => write!(f, "plus"),
            Operation::Minus => write!(f, "minus"),
            Operation::Times => write!(f, "times"),
            Operation::DividedBy => write!(f, "divided-by"),
        }
    }
}

```

```

/**
 * InvokeArgs will be serialized as JSON and sent to the AWS Lambda handler.
 */
#[derive(Debug)]
pub enum InvokeArgs {
    Increment(i32),
    Arithmetic(Operation, i32, i32),
}

impl Serialize for InvokeArgs {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: serde::Serializer,
    {
        match self {
            InvokeArgs::Increment(i) => serializer.serialize_i32(*i),
            InvokeArgs::Arithmetic(o, i, j) => {
                let mut map: S::SerializeMap =
serializer.serialize_map(Some(3))?;
                map.serialize_key(&"op".to_string())?;
                map.serialize_value(&o.to_string())?;
                map.serialize_key(&"i".to_string())?;
                map.serialize_value(&i)?;
                map.serialize_key(&"j".to_string())?;
                map.serialize_value(&j)?;
                map.end()
            }
        }
    }
}

/** A policy document allowing Lambda to execute this function on the account's
    behalf. */
const ROLE_POLICY_DOCUMENT: &str = r#"{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "Service": "lambda.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
    ]
}"#;

```

```
/**
 * A LambdaManager gathers all the resources necessary to run the Lambda example
 * scenario.
 * This includes instantiated aws_sdk clients and details of resource names.
 */
pub struct LambdaManager {
    iam_client: aws_sdk_iam::Client,
    lambda_client: aws_sdk_lambda::Client,
    s3_client: aws_sdk_s3::Client,
    lambda_name: String,
    role_name: String,
    bucket: String,
    own_bucket: bool,
}

// These unit type structs provide nominal typing on top of String parameters for
// LambdaManager::new
pub struct LambdaName(pub String);
pub struct RoleName(pub String);
pub struct Bucket(pub String);
pub struct OwnBucket(pub bool);

impl LambdaManager {
    pub fn new(
        iam_client: aws_sdk_iam::Client,
        lambda_client: aws_sdk_lambda::Client,
        s3_client: aws_sdk_s3::Client,
        lambda_name: LambdaName,
        role_name: RoleName,
        bucket: Bucket,
        own_bucket: OwnBucket,
    ) -> Self {
        Self {
            iam_client,
            lambda_client,
            s3_client,
            lambda_name: lambda_name.0,
            role_name: role_name.0,
            bucket: bucket.0,
            own_bucket: own_bucket.0,
        }
    }
}
```



```

/**
 * Load the AWS configuration from the environment.
 * Look up lambda_name and bucket if none are given, or generate a random
name if not present in the environment.
 * If the bucket name is provided, the caller needs to have created the
bucket.
 * If the bucket name is generated, it will be created.
 */
pub async fn load_from_env(lambda_name: Option<String>, bucket:
Option<String>) -> Self {
    let sdk_config = aws_config::load_from_env().await;
    let lambda_name = LambdaName(lambda_name.unwrap_or_else(|| {
        std::env::var("LAMBDA_NAME").unwrap_or_else(|_|
"rust_lambda_example".to_string())
    }));
    let role_name = RoleName(format!("{}",_role", lambda_name.0));
    let (bucket, own_bucket) =
        match bucket {
            Some(bucket) => (Bucket(bucket), false),
            None => (
                Bucket(std::env::var("LAMBDA_BUCKET").unwrap_or_else(|_| {
                    format!("rust-lambda-example-{}", uuid::Uuid::new_v4())
                })),
                true,
            ),
        };

    let s3_client = aws_sdk_s3::Client::new(&sdk_config);

    if own_bucket {
        info!("Creating bucket for demo: {}", bucket.0);
        s3_client
            .create_bucket()
            .bucket(bucket.0.clone())
            .create_bucket_configuration(
                CreateBucketConfiguration::builder()
                    .location_constraint(aws_sdk_s3::types::BucketLocationConstraint::from(
                        sdk_config.region().unwrap().as_ref(),
                    ))
                    .build(),
            )
            .send()
            .await
    }

```

```

        .unwrap();
    }

    Self::new(
        aws_sdk_iam::Client::new(&sdk_config),
        aws_sdk_lambda::Client::new(&sdk_config),
        s3_client,
        lambda_name,
        role_name,
        bucket,
        OwnBucket(own_bucket),
    )
}

/**
 * Upload function code from a path to a zip file.
 * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
 * The easiest way to create such a zip is to use `cargo lambda build --
output-format Zip`.
 */
async fn prepare_function(
    &self,
    zip_file: PathBuf,
    key: Option<String>,
) -> Result<FunctionCode, anyhow::Error> {
    let body = ByteStream::from_path(zip_file).await?;

    let key = key.unwrap_or_else(|| format!("{}", self.lambda_name));

    info!("Uploading function code to s3://{}/{}", self.bucket, key);
    let _ = self
        .s3_client
        .put_object()
        .bucket(self.bucket.clone())
        .key(key.clone())
        .body(body)
        .send()
        .await?;

    Ok(FunctionCode::builder()
        .s3_bucket(self.bucket.clone())
        .s3_key(key)
        .build())
}

```

```

/**
 * Create a function, uploading from a zip file.
 */
pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
anyhow::Error> {
    let code = self.prepare_function(zip_file, None).await?;

    let key = code.s3_key().unwrap().to_string();

    let role = self.create_role().await.map_err(|e| anyhow!(e))?;

    info!("Created iam role, waiting 15s for it to become active");
    tokio::time::sleep(Duration::from_secs(15)).await;

    info!("Creating lambda function {}", self.lambda_name);
    let _ = self
        .lambda_client
        .create_function()
        .function_name(self.lambda_name.clone())
        .code(code)
        .role(role.arn())
        .runtime(aws_sdk_lambda::types::Runtime::ProvidedAl2)
        .handler("_unused")
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;

    self.lambda_client
        .publish_version()
        .function_name(self.lambda_name.clone())
        .send()
        .await?;

    Ok(key)
}

/**
 * Create an IAM execution role for the managed Lambda function.
 * If the role already exists, use that instead.
 */

```

```

    async fn create_role(&self) -> Result<aws_sdk_iam::types::Role,
CreateRoleError> {
        info!("Creating execution role for function");
        let get_role = self
            .iam_client
            .get_role()
            .role_name(self.role_name.clone())
            .send()
            .await;
        if let Ok(get_role) = get_role {
            if let Some(role) = get_role.role {
                return Ok(role);
            }
        }

        let create_role = self
            .iam_client
            .create_role()
            .role_name(self.role_name.clone())
            .assume_role_policy_document(ROLE_POLICY_DOCUMENT)
            .send()
            .await;

        match create_role {
            Ok(create_role) => match create_role.role {
                Some(role) => Ok(role),
                None => Err(CreateRoleError::generic(
                    ErrorMetadata::builder()
                        .message("CreateRole returned empty success")
                        .build(),
                )),
            },
            Err(err) => Err(err.into_service_error()),
        }
    }

    /**
     * Poll `is_function_ready` with a 1-second delay. It returns when the
     function is ready or when there's an error checking the function's state.
     */
    pub async fn wait_for_function_ready(&self) -> Result<(), anyhow::Error> {
        info!("Waiting for function");
        while !self.is_function_ready(None).await? {
            info!("Function is not ready, sleeping 1s");

```

```

        tokio::time::sleep(Duration::from_secs(1)).await;
    }
    Ok(())
}

/**
 * Check if a Lambda function is ready to be invoked.
 * A Lambda function is ready for this scenario when its state is active and
its LastUpdateStatus is Successful.
 * Additionally, if a sha256 is provided, the function must have that as its
current code hash.
 * Any missing properties or failed requests will be reported as an Err.
 */
async fn is_function_ready(
    &self,
    expected_code_sha256: Option<&str>,
) -> Result<bool, anyhow::Error> {
    match self.get_function().await {
        Ok(func) => {
            if let Some(config) = func.configuration() {
                if let Some(state) = config.state() {
                    info!(?state, "Checking if function is active");
                    if !matches!(state, State::Active) {
                        return Ok(false);
                    }
                }
            }
            match config.last_update_status() {
                Some(last_update_status) => {
                    info!(?last_update_status, "Checking if function is
ready");

                    match last_update_status {
                        LastUpdateStatus::Successful => {
                            // continue
                        }
                        LastUpdateStatus::Failed |
LastUpdateStatus::InProgress => {
                            return Ok(false);
                        }
                        unknown => {
                            warn!(
                                status_variant = unknown.as_str(),
                                "LastUpdateStatus unknown"
                            );
                            return Err(anyhow!(

```

```

                                "Unknown LastUpdateStatus, fn config is
{config:?}"

                                ));
                                }
                                }
                                }
                                None => {
                                    warn!("Missing last update status");
                                    return Ok(false);
                                }
                                };
                                if expected_code_sha256.is_none() {
                                    return Ok(true);
                                }
                                if let Some(code_sha256) = config.code_sha256() {
                                    return Ok(code_sha256 ==
expected_code_sha256.unwrap_or_default());
                                }
                                }
                                }
                                Err(e) => {
                                    warn!(?e, "Could not get function while waiting");
                                }
                                }
                                Ok(false)
                                }

/** Get the Lambda function with this Manager's name. */
pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error>
{
    info!("Getting lambda function");
    self.lambda_client
        .get_function()
        .function_name(self.lambda_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from)
}

/** List all Lambda functions in the current Region. */
pub async fn list_functions(&self) -> Result<ListFunctionsOutput,
anyhow::Error> {
    info!("Listing lambda functions");
    self.lambda_client

```

```

        .list_functions()
        .send()
        .await
        .map_err(anyhow::Error::from)
    }

    /** Invoke the lambda function using calculator InvokeArgs. */
    pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
        info!(?args, "Invoking {}", self.lambda_name);
        let payload = serde_json::to_string(&args)?;
        debug!(?payload, "Sending payload");
        self.lambda_client
            .invoke()
            .function_name(self.lambda_name.clone())
            .payload(Blob::new(payload))
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

    /** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
    pub async fn update_function_code(
        &self,
        zip_file: PathBuf,
        key: String,
    ) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
        let function_code = self.prepare_function(zip_file, Some(key)).await?;

        info!("Updating code for {}", self.lambda_name);
        let update = self
            .lambda_client
            .update_function_code()
            .function_name(self.lambda_name.clone())
            .s3_bucket(self.bucket.clone())
            .s3_key(function_code.s3_key().unwrap().to_string())
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        Ok(update)
    }

```

```

}

/** Update the environment for a function. */
pub async fn update_function_configuration(
    &self,
    environment: Environment,
) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
    info!(
        ?environment,
        "Updating environment for {}", self.lambda_name
    );
    let updated = self
        .lambda_client
        .update_function_configuration()
        .function_name(self.lambda_name.clone())
        .environment(environment)
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;

    Ok(updated)
}

/** Delete a function and its role, and if possible or necessary, its
associated code object and bucket. */
pub async fn delete_function(
    &self,
    location: Option<String>,
) -> (
    Result<DeleteFunctionOutput, anyhow::Error>,
    Result<DeleteRoleOutput, anyhow::Error>,
    Option<Result<DeleteObjectOutput, anyhow::Error>>,
) {
    info!("Deleting lambda function {}", self.lambda_name);
    let delete_function = self
        .lambda_client
        .delete_function()
        .function_name(self.lambda_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from);

```



```

        info!("Deleting iam role {}", self.role_name);
        let delete_role = self
            .iam_client
            .delete_role()
            .role_name(self.role_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from);

        let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
            if let Some(location) = location {
                info!("Deleting object {location}");
                Some(
                    self.s3_client
                        .delete_object()
                        .bucket(self.bucket.clone())
                        .key(location)
                        .send()
                        .await
                        .map_err(anyhow::Error::from),
                )
            } else {
                info!(?location, "Skipping delete object");
                None
            };

        (delete_function, delete_role, delete_object)
    }

    pub async fn cleanup(
        &self,
        location: Option<String>,
    ) -> (
        (
            Result<DeleteFunctionOutput, anyhow::Error>,
            Result<DeleteRoleOutput, anyhow::Error>,
            Option<Result<DeleteObjectOutput, anyhow::Error>>,
        ),
        Option<Result<DeleteBucketOutput, anyhow::Error>>,
    ) {
        let delete_function = self.delete_function(location).await;

        let delete_bucket = if self.own_bucket {
            info!("Deleting bucket {}", self.bucket);

```

```

        if delete_function.2.is_none() ||
delete_function.2.as_ref().unwrap().is_ok() {
            Some(
                self.s3_client
                    .delete_bucket()
                    .bucket(self.bucket.clone())
                    .send()
                    .await
                    .map_err(anyhow::Error::from),
            )
        } else {
            None
        }
    } else {
        info!("No bucket to clean up");
        None
    };
}

(delete_function, delete_bucket)
}
}

/**
 * Testing occurs primarily as an integration test running the `scenario` bin
 * successfully.
 * Each action relies deeply on the internal workings and state of Amazon Simple
 * Storage Service (Amazon S3), Lambda, and IAM working together.
 * It is therefore infeasible to mock the clients to test the individual actions.
 */
#[cfg(test)]
mod test {
    use super::{InvokeArgs, Operation};
    use serde_json::json;

    /** Make sure that the JSON output of serializing InvokeArgs is what's
    expected by the calculator. */
    #[test]
    fn test_serialize() {
        assert_eq!(json!(InvokeArgs::Increment(5)), 5);
        assert_eq!(
            json!(InvokeArgs::Arithmetic(Operation::Plus, 5, 7)).to_string(),
            r#"{"op":"plus","i":5,"j":7}"#.to_string(),
        );
    }
}

```

```
}
```

Un file binario per eseguire lo scenario dall'inizio alla fine, utilizzando i flag della linea di comando per controllare alcuni comportamenti. Questo file è `src/bin /scenario.rs` nella cassa.

```
/*  
## Service actions  
  
Service actions wrap the SDK call, taking a client and any specific parameters  
necessary for the call.  
  
* CreateFunction  
* GetFunction  
* ListFunctions  
* Invoke  
* UpdateFunctionCode  
* UpdateFunctionConfiguration  
* DeleteFunction  
  
## Scenario  
A scenario runs at a command prompt and prints output to the user on the result  
of each service action. A scenario can run in one of two ways: straight through,  
printing out progress as it goes, or as an interactive question/answer script.  
  
## Getting started with functions  
  
Use an SDK to manage AWS Lambda functions: create a function, invoke it, update  
its code, invoke it again, view its output and logs, and delete it.  
  
This scenario uses two Lambda handlers:  
_Note: Handlers don't use AWS SDK API calls._  
  
The increment handler is straightforward:  
  
1. It accepts a number, increments it, and returns the new value.  
2. It performs simple logging of the result.  
  
The arithmetic handler is more complex:  
1. It accepts a set of actions ['plus', 'minus', 'times', 'divided-by'] and two  
numbers, and returns the result of the calculation.
```

2. It uses an environment variable to control log level (such as DEBUG, INFO, WARNING, ERROR).

It logs a few things at different levels, such as:

- \* DEBUG: Full event data.
- \* INFO: The calculation result.
- \* WARN~ING~: When a divide by zero error occurs.
- \* This will be the typical `RUST\_LOG` variable.

The steps of the scenario are:

1. Create an AWS Identity and Access Management (IAM) role that meets the following requirements:

- \* Has an assume\_role policy that grants 'lambda.amazonaws.com' the 'sts:AssumeRole' action.

- \* Attaches the 'arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole' managed role.

- \* \_You must wait for ~10 seconds after the role is created before you can use it!\_

2. Create a function (CreateFunction) for the increment handler by packaging it as a zip and doing one of the following:

- \* Adding it with CreateFunction Code.ZipFile.

- \* --or--

- \* Uploading it to Amazon Simple Storage Service (Amazon S3) and adding it with CreateFunction Code.S3Bucket/S3Key.

- \* \_Note: Zipping the file does not have to be done in code.\_

- \* If you have a waiter, use it to wait until the function is active.

Otherwise, call GetFunction until State is Active.

3. Invoke the function with a number and print the result.

4. Update the function (UpdateFunctionCode) to the arithmetic handler by packaging it as a zip and doing one of the following:

- \* Adding it with UpdateFunctionCode ZipFile.

- \* --or--

- \* Uploading it to Amazon S3 and adding it with UpdateFunctionCode S3Bucket/S3Key.

5. Call GetFunction until Configuration.LastUpdateStatus is 'Successful' (or 'Failed').

6. Update the environment variable by calling UpdateFunctionConfiguration and pass it a log level, such as:

- \* Environment={'Variables': {'RUST\_LOG': 'TRACE'}}

7. Invoke the function with an action from the list and a couple of values. Include LogType='Tail' to get logs in the result. Print the result of the calculation and the log.

8. [Optional] Invoke the function to provoke a divide-by-zero error and show the log result.
9. List all functions for the account, using pagination (`ListFunctions`).
10. Delete the function (`DeleteFunction`).
11. Delete the role.

Each step should use the function created in Service Actions to abstract calling the SDK.

```
*/

use aws_sdk_lambda::{operation::invoke::InvokeOutput, types::Environment};
use clap::Parser;
use std::{collections::HashMap, path::PathBuf};
use tracing::{debug, info, warn};
use tracing_subscriber::EnvFilter;

use lambda_code_examples::actions::{
    InvokeArgs::{Arithmetic, Increment},
    LambdaManager, Operation,
};

#[derive(Debug, Parser)]
pub struct Opt {
    /// The AWS Region.
    #[structopt(short, long)]
    pub region: Option<String>,

    // The bucket to use for the FunctionCode.
    #[structopt(short, long)]
    pub bucket: Option<String>,

    // The name of the Lambda function.
    #[structopt(short, long)]
    pub lambda_name: Option<String>,

    // The number to increment.
    #[structopt(short, long, default_value = "12")]
    pub inc: i32,

    // The left operand.
    #[structopt(long, default_value = "19")]
    pub num_a: i32,

    // The right operand.
```

```

#[structopt(long, default_value = "23")]
pub num_b: i32,

// The arithmetic operation.
#[structopt(short, long, default_value = "plus")]
pub operation: Operation,

#[structopt(long)]
pub cleanup: Option<bool>,

#[structopt(long)]
pub no_cleanup: Option<bool>,
}

fn code_path(lambda: &str) -> PathBuf {
    PathBuf::from(format!("../target/lambda/{lambda}/bootstrap.zip"))
}

fn log_invoke_output(invoker: &InvokeOutput, message: &str) {
    if let Some(payload) = invoker.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
        info!("Could not extract payload")
    }
    if let Some(logs) = invoker.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}

async fn main_block(
    opt: &Opt,
    manager: &LambdaManager,
    code_location: String,
) -> Result<(), anyhow::Error> {
    let invoker = manager.invoke(Increment(opt.inc)).await?;
    log_invoke_output(&invoker, "Invoked function configured as increment");

    let update_code = manager
        .update_function_code(code_path("arithmetic"), code_location.clone())
        .await?;

```

```

let code_sha256 = update_code.code_sha256().unwrap_or("Unknown SHA");
info!(?code_sha256, "Updated function code with arithmetic.zip");

let arithmetic_args = Arithmetic(opt.operation, opt.num_a, opt.num_b);
let invoke = manager.invoke(arithmetic_args).await?;
log_invoke_output(&invoke, "Invoked function configured as arithmetic");

let update = manager
    .update_function_configuration(
        Environment::builder()
            .set_variables(Some(HashMap::from([
                "RUST_LOG".to_string(),
                "trace".to_string(),
            ])))
            .build(),
    )
    .await?;
let updated_environment = update.environment();
info!(?updated_environment, "Updated function configuration");

let invoke = manager
    .invoke(Arithmetic(opt.operation, opt.num_a, opt.num_b))
    .await?;
log_invoke_output(
    &invoke,
    "Invoked function configured as arithmetic with increased logging",
);

let invoke = manager
    .invoke(Arithmetic(Operation::DividedBy, opt.num_a, 0))
    .await?;
log_invoke_output(
    &invoke,
    "Invoked function configured as arithmetic with divide by zero",
);

Ok:::<(), anyhow::Error>(( ))
}

#[tokio::main]
async fn main() {
    tracing_subscriber::fmt()
        .without_time()
        .with_file(true)

```

```
.with_line_number(true)
.with_env_filter(EnvFilter::from_default_env())
.init();

let opt = Opt::parse();
let manager = LambdaManager::load_from_env(opt.lambda_name.clone(),
opt.bucket.clone()).await;

let key = match manager.create_function(code_path("increment")).await {
    Ok(init) => {
        info!(?init, "Created function, initially with increment.zip");
        let run_block = main_block(&opt, &manager, init.clone()).await;
        info!(?run_block, "Finished running example, cleaning up");
        Some(init)
    }
    Err(err) => {
        warn!(?err, "Error happened when initializing function");
        None
    }
};

if Some(false) == opt.cleanup || Some(true) == opt.no_cleanup {
    info!("Skipping cleanup")
} else {
    let delete = manager.cleanup(key).await;
    info!(?delete, "Deleted function & cleaned up resources");
}
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API SDK AWS per Rust.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)



## SAP ABAP

## SDK per SAP ABAP

 Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

TRY.
    "Create an AWS Identity and Access Management (IAM) role that grants AWS
    Lambda permission to write to logs."
    DATA(lv_policy_document) = `{` &&
        `"Version": "2012-10-17",` &&
        `"Statement": [` &&
            `{` &&
                `"Effect": "Allow",` &&
                `"Action": [` &&
                    `"sts:AssumeRole"` &&
                `],` &&
                `"Principal": {` &&
                    `"Service": [` &&
                        `"lambda.amazonaws.com"` &&
                    `]` &&
                `}` &&
            `}` &&
        `]` &&
    `}`.

TRY.
    DATA(lo_create_role_output) = lo_iam->createrole(
        iv_role_name = iv_role_name
        iv_assumerolepolicydocument = lv_policy_document
        iv_description = 'Grant lambda permission to write to
logs' ).

    DATA(lv_role_arn) = lo_create_role_output->get_role( )->get_arn( ).
    MESSAGE 'IAM role created.' TYPE 'I'.
    WAIT UP TO 10 SECONDS.          " Make sure that the IAM role is
ready for use. "
    CATCH /aws1/cx_iamentityalrddyexx.
    DATA(lo_role) = lo_iam->getrole( iv_role_name = iv_role_name ).

```

```

        lv_role_arn = lo_role->get_role( )->get_arn( ).
    CATCH /aws1/cx_iaminvalidinputex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
    CATCH /aws1/cx_iammalformedplydocex.
        MESSAGE 'Policy document in the request is malformed.' TYPE 'E'.
    ENDTRY.

TRY.
    lo_iam->attachrolepolicy(
        iv_role_name = iv_role_name
        iv_policyarn = 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole' ).
        MESSAGE 'Attached policy to the IAM role.' TYPE 'I'.
    CATCH /aws1/cx_iaminvalidinputex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
    CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'The requested resource entity does not exist.' TYPE 'E'.
    CATCH /aws1/cx_iamplynotattachableex.
        MESSAGE 'Service role policies can only be attached to the service-
linked role for their service.' TYPE 'E'.
    CATCH /aws1/cx_iamunmodableentityex.
        MESSAGE 'Service that depends on the service-linked role is not
modifiable.' TYPE 'E'.
    ENDTRY.

" Create a Lambda function and upload handler code. "
" Lambda function performs 'increment' action on a number. "
TRY.
    lo_lmd->createfunction(
        iv_functionname = iv_function_name
        iv_runtime = `python3.9`
        iv_role = lv_role_arn
        iv_handler = iv_handler
        io_code = io_initial_zip_file
        iv_description = 'AWS Lambda code example' ).
        MESSAGE 'Lambda function created.' TYPE 'I'.
    CATCH /aws1/cx_lmdcodestorageexcdex.
        MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
    CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
    CATCH /aws1/cx_lmdresourcenotfoundex.
        MESSAGE 'The requested resource does not exist.' TYPE 'E'.
    ENDTRY.

```

```

    " Verify the function is in Active state "
    WHILE lo_lmd->getfunction( iv_functionname = iv_function_name )-
>get_configuration( )->ask_state( ) <> 'Active'.
        IF sy-index = 10.
            EXIT.                " Maximum 10 seconds. "
        ENDIF.
        WAIT UP TO 1 SECONDS.
    ENDWHILE.

    "Invoke the function with a single parameter and get results."
    TRY.
        DATA(lv_json) = /aws1/cl_rt_util=>string_to_xstring(
            `{` &&
            `"action": "increment",` &&
            `"number": 10` &&
            `}` ).
        DATA(lo_initial_invoke_output) = lo_lmd->invoke(
            iv_functionname = iv_function_name
            iv_payload = lv_json ).
        ov_initial_invoke_payload = lo_initial_invoke_output->get_payload( ).
        " ov_initial_invoke_payload is returned for testing purposes. "
        DATA(lo_writer_json) = cl_sxml_string_writer=>create( type =
if_sxml=>co_xt_json ).
        CALL TRANSFORMATION id SOURCE XML ov_initial_invoke_payload RESULT
XML lo_writer_json.
        DATA(lv_result) = cl_abap_codepage=>convert_from( lo_writer_json-
>get_output( ) ).
        MESSAGE 'Lambda function invoked.' TYPE 'I'.
        CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
        CATCH /aws1/cx_lmdinvrequestctx.
        MESSAGE 'Unable to parse request body as JSON.' TYPE 'E'.
        CATCH /aws1/cx_lmdresourcenotfoundex.
        MESSAGE 'The requested resource does not exist.' TYPE 'E'.
        CATCH /aws1/cx_lmdunsuppedmediatyp00.
        MESSAGE 'Invoke request body does not have JSON as its content type.'
TYPE 'E'.
    ENDTRY.

    " Update the function code and configure its Lambda environment with an
environment variable. "
    " Lambda function is updated to perform 'decrement' action also. "
    TRY.
        lo_lmd->updatefunctioncode(

```

```

        iv_functionname = iv_function_name
        iv_zipfile = io_updated_zip_file ).
    WAIT UP TO 10 SECONDS.          " Make sure that the update is
completed. "
    MESSAGE 'Lambda function code updated.' TYPE 'I'.
    CATCH /aws1/cx_lmdcodestorageexcdex.
    MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
    CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
    CATCH /aws1/cx_lmdresourcenotfoundex.
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.
ENDTRY.

TRY.
    DATA lt_variables TYPE /aws1/
cl_lmdenvironmentvaria00=>tt_environmentvariables.
    DATA ls_variable LIKE LINE OF lt_variables.
    ls_variable-key = 'LOG_LEVEL'.
    ls_variable-value = NEW /aws1/cl_lmdenvironmentvaria00( iv_value =
'info' ).
    INSERT ls_variable INTO TABLE lt_variables.

    lo_lmd->updatefunctionconfiguration(
        iv_functionname = iv_function_name
        io_environment = NEW /aws1/cl_lmdenvironment( it_variables =
lt_variables ) ).
    WAIT UP TO 10 SECONDS.          " Make sure that the update is
completed. "
    MESSAGE 'Lambda function configuration/settings updated.' TYPE 'I'.
    CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
    CATCH /aws1/cx_lmdresourceconflictex.
    MESSAGE 'Resource already exists or another operation is in
progress.' TYPE 'E'.
    CATCH /aws1/cx_lmdresourcenotfoundex.
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.
ENDTRY.

"Invoke the function with new parameters and get results. Display the
execution log that's returned from the invocation."
TRY.
    lv_json = /aws1/cl_rt_util=>string_to_xstring(
        `{`  &&
        `"action": "decrement",`  &&

```

```

        ` "number": 10 ` &&
        ` } ` ).
    DATA(lo_updated_invoke_output) = lo_lmd->invoke(
        iv_functionname = iv_function_name
        iv_payload = lv_json ).
    ov_updated_invoke_payload = lo_updated_invoke_output->get_payload( ).
    " ov_updated_invoke_payload is returned for testing purposes. "
    lo_writer_json = cl_sxml_string_writer=>create( type =
if_sxml=>co_xt_json ).
    CALL TRANSFORMATION id SOURCE XML ov_updated_invoke_payload RESULT
XML lo_writer_json.
    lv_result = cl_abap_codepage=>convert_from( lo_writer_json-
>get_output( ) ).
    MESSAGE 'Lambda function invoked.' TYPE 'I'.
    CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
    CATCH /aws1/cx_lmdinvrequestctx.
    MESSAGE 'Unable to parse request body as JSON.' TYPE 'E'.
    CATCH /aws1/cx_lmdresourcenotfoundex.
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.
    CATCH /aws1/cx_lmdunsuppedmediatyp00.
    MESSAGE 'Invoke request body does not have JSON as its content type.'
TYPE 'E'.
    ENDTRY.

    " List the functions for your account. "
    TRY.
        DATA(lo_list_output) = lo_lmd->listfunctions( ).
        DATA(lt_functions) = lo_list_output->get_functions( ).
        MESSAGE 'Retrieved list of Lambda functions.' TYPE 'I'.
        CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
    ENDTRY.

    " Delete the Lambda function. "
    TRY.
        lo_lmd->deletefunction( iv_functionname = iv_function_name ).
        MESSAGE 'Lambda function deleted.' TYPE 'I'.
        CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
        CATCH /aws1/cx_lmdresourcenotfoundex.
        MESSAGE 'The requested resource does not exist.' TYPE 'W'.
    ENDTRY.

```

```

" Detach role policy. "
TRY.
    lo_iam->detachrolepolicy(
        iv_rolename = iv_role_name
        iv_policyarn = 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole' ).
    MESSAGE 'Detached policy from the IAM role.' TYPE 'I'.
CATCH /aws1/cx_iaminvalidinputex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_iamnosuchentityex.
    MESSAGE 'The requested resource entity does not exist.' TYPE 'W'.
CATCH /aws1/cx_iamplynotattachableex.
    MESSAGE 'Service role policies can only be attached to the service-
linked role for their service.' TYPE 'E'.
CATCH /aws1/cx_iamunmodableentityex.
    MESSAGE 'Service that depends on the service-linked role is not
modifiable.' TYPE 'E'.
ENDTRY.

" Delete the IAM role. "
TRY.
    lo_iam->deleterole( iv_rolename = iv_role_name ).
    MESSAGE 'IAM role deleted.' TYPE 'I'.
CATCH /aws1/cx_iamnosuchentityex.
    MESSAGE 'The requested resource entity does not exist.' TYPE 'W'.
CATCH /aws1/cx_iamunmodableentityex.
    MESSAGE 'Service that depends on the service-linked role is not
modifiable.' TYPE 'E'.
ENDTRY.

CATCH /aws1/cx_rt_service_generic INTO lo_exception.
    DATA(lv_error) = lo_exception->get_longtext( ).
    MESSAGE lv_error TYPE 'E'.
ENDTRY.

```

- Per informazioni dettagliate sull'API, consulta gli argomenti seguenti nella documentazione di riferimento dell'API AWS SDK per SAP ABAP.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)

- [ListFunctions](#)
- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)

## Swift

### SDK per Swift

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Definisci la prima funzione Lambda, che incrementa semplicemente il valore specificato.

```
// swift-tools-version: 5.9
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
//
// The swift-tools-version declares the minimum version of Swift required to
// build this package.

import PackageDescription

let package = Package(
    name: "increment",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/swift-server/swift-aws-lambda-runtime.git",
            branch: "main"),
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
        // or a test suite.
        // Targets can depend on other targets in this package and products
```

```

        // from dependencies.
        .executableTarget(
            name: "increment",
            dependencies: [
                .product(name: "AWSLambdaRuntime", package: "swift-aws-lambda-
runtime"),
            ],
            path: "Sources"
        )
    ]
)

import Foundation
import AWSLambdaRuntime

/// Represents the contents of the requests being received from the client.
/// This structure must be `Decodable` to indicate that its initializer
/// converts an external representation into this type.
struct Request: Decodable, Sendable {
    /// The action to perform.
    let action: String
    /// The number to act upon.
    let number: Int
}

/// The contents of the response sent back to the client. This must be
/// `Encodable`.
struct Response: Encodable, Sendable {
    /// The resulting value after performing the action.
    let answer: Int?
}

/// The Lambda function body.
///
/// - Parameters:
///   - event: The `Request` describing the request made by the
///     client.
///   - context: A `LambdaContext` describing the context in
///     which the lambda function is running.
///
/// - Returns: A `Response` object that will be encoded to JSON and sent
///   to the client by the Lambda runtime.
let incrementLambdaRuntime = LambdaRuntime {

```



```

        (event: Request, context: LambdaContext) -> Response in
    let action = event.action
    var answer: Int?

    if action != "increment" {
        context.logger.error("Unrecognized operation: \"\$(action)\". The only
supported action is \"increment\".")
    } else {
        answer = event.number + 1
        context.logger.info("The calculated answer is \"\$(answer!)\".")
    }

    let response = Response(answer: answer)
    return response
}

// Run the Lambda runtime code.

try await incrementLambdaRuntime.run()

```

Definisce la seconda funzione Lambda, che esegue un'operazione aritmetica su due numeri.

```

// swift-tools-version: 5.9
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
//
// The swift-tools-version declares the minimum version of Swift required to
// build this package.

import PackageDescription

let package = Package(
    name: "calculator",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/swift-server/swift-aws-lambda-runtime.git",
            branch: "main"),
    ]
)

```

```

    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
        // or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "calculator",
            dependencies: [
                .product(name: "AWSLambdaRuntime", package: "swift-aws-lambda-
runtime"),
            ],
            path: "Sources"
        )
    ]
)

import Foundation
import AWSLambdaRuntime

/// Represents the contents of the requests being received from the client.
/// This structure must be `Decodable` to indicate that its initializer
/// converts an external representation into this type.
struct Request: Decodable, Sendable {
    /// The action to perform.
    let action: String
    /// The first number to act upon.
    let x: Int
    /// The second number to act upon.
    let y: Int
}

/// A dictionary mapping operation names to closures that perform that
/// operation and return the result.
let actions = [
    "plus": { (x: Int, y: Int) -> Int in
        return x + y
    },
    "minus": { (x: Int, y: Int) -> Int in
        return x - y
    },
    "times": { (x: Int, y: Int) -> Int in
        return x * y
    },

```

```
    "divided-by": { (x: Int, y: Int) -> Int in
        return x / y
    }
]

/// The contents of the response sent back to the client. This must be
/// `Encodable`.
struct Response: Encodable, Sendable {
    /// The resulting value after performing the action.
    let answer: Int?
}

/// The Lambda function's entry point. Called by the Lambda runtime.
///
/// - Parameters:
///   - event: The `Request` describing the request made by the
///     client.
///   - context: A `LambdaContext` describing the context in
///     which the lambda function is running.
///
/// - Returns: A `Response` object that will be encoded to JSON and sent
///   to the client by the Lambda runtime.
let calculatorLambdaRuntime = LambdaRuntime {
    (_ event: Request, context: LambdaContext) -> Response in
    let action = event.action
    var answer: Int?
    var actionFunc: ((Int, Int) -> Int)?

    // Get the closure to run to perform the calculation.

    actionFunc = await actions[action]

    guard let actionFunc else {
        context.logger.error("Unrecognized operation '\(action)'\")
        return Response(answer: nil)
    }

    // Perform the calculation and return the answer.

    answer = actionFunc(event.x, event.y)

    guard let answer else {
        context.logger.error("Error computing \((event.x) \((action) \((event.y))")
```

```

    }
    context.logger.info("\(event.x) \(action) \(event.y) = \(answer)")

    return Response(answer: answer)
}

try await calculatorLambdaRuntime.run()

```

Definisce il programma principale, che invocherà le due funzioni Lambda.

```

// swift-tools-version: 5.9
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
//
// The swift-tools-version declares the minimum version of Swift required to
// build this package.

import PackageDescription

let package = Package(
    name: "lambda-basics",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/aws-labs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
        // or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "lambda-basics",

```

```

        dependencies: [
            .product(name: "AWSLambda", package: "aws-sdk-swift"),
            .product(name: "AWSIAM", package: "aws-sdk-swift"),
            .product(name: "ArgumentParser", package: "swift-argument-
parser")
        ],
        path: "Sources"
    )
]
)

//
/// An example demonstrating a variety of important AWS Lambda functions.

import ArgumentParser
import AWSIAM
import SmithyWaitersAPI
import AWSClientRuntime
import AWSLambda
import Foundation

/// Represents the contents of the requests being received from the client.
/// This structure must be `Decodable` to indicate that its initializer
/// converts an external representation into this type.
struct IncrementRequest: Encodable, Decodable, Sendable {
    /// The action to perform.
    let action: String
    /// The number to act upon.
    let number: Int
}

struct Response: Encodable, Decodable, Sendable {
    /// The resulting value after performing the action.
    let answer: Int?
}

struct CalculatorRequest: Encodable, Decodable, Sendable {
    /// The action to perform.
    let action: String
    /// The first number to act upon.
    let x: Int
    /// The second number to act upon.
    let y: Int
}

```

```

let exampleName = "SwiftLambdaRoleExample"
let basicsFunctionName = "lambda-basics-function"

/// The ARN of the standard IAM policy for execution of Lambda functions.
let policyARN = "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"

struct ExampleCommand: ParsableCommand {
    // -MARK: Command arguments
    @Option(help: "Name of the IAM Role to use for the Lambda functions")
    var role = exampleName
    @Option(help: "Zip archive containing the 'increment' lambda function")
    var incpath: String
    @Option(help: "Zip archive containing the 'calculator' lambda function")
    var calcpath: String
    @Option(help: "Name of the Amazon S3 Region to use (default: us-east-1)")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "lambda-basics",
        abstract: ""
        This example demonstrates several common operations using AWS Lambda.
        "",
        discussion: ""
        ""
    )

    /// Returns the specified IAM role object.
    ///
    /// - Parameters:
    ///   - iamClient: `IAMClient` to use when looking for the role.
    ///   - roleName: The name of the role to check.
    ///
    /// - Returns: The `IAMClientTypes.Role` representing the specified role.
    func getRole(iamClient: IAMClient, roleName: String) async throws
        -> IAMClientTypes.Role {
        do {
            let roleOutput = try await iamClient.getRole(
                input: GetRoleInput(
                    roleName: roleName
                )
            )
        }
    }
}

```

```

        guard let role = roleOutput.role else {
            throw ExampleError.roleNotFound
        }
        return role
    } catch {
        throw ExampleError.roleNotFound
    }
}

/// Create the AWS IAM role that will be used to access AWS Lambda.
///
/// - Parameters:
///   - iamClient: The AWS `IAMClient` to use.
///   - roleName: The name of the AWS IAM role to use for Lambda.
///
/// - Throws: `ExampleError.roleCreateError`
///
/// - Returns: The `IAMClientTypes.Role` struct that describes the new role.
func createRoleForLambda(iamClient: IAMClient, roleName: String) async throws
-> IAMClientTypes.Role {
    let output = try await iamClient.createRole(
        input: CreateRoleInput(
            assumeRolePolicyDocument:
                """
                {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {"Service": "lambda.amazonaws.com"},
                            "Action": "sts:AssumeRole"
                        }
                    ]
                }
                """,
            roleName: roleName
        )
    )

    guard let role = output.role else {
        throw ExampleError.roleCreateError
    }

    // Wait for the role to be ready for use.

```

```

        _ = try await iamClient.waitForRoleExists(
            options: WaiterOptions(
                maxWaitTime: 20,
                minDelay: 0.5,
                maxDelay: 2
            ),
            input: GetRoleInput(roleName: roleName)
        )

        return role
    }

    /// Detect whether or not the AWS Lambda function with the specified name
    /// exists, by requesting its function information.
    ///
    /// - Parameters:
    ///   - lambdaClient: The `LambdaClient` to use.
    ///   - name: The name of the AWS Lambda function to find.
    ///
    /// - Returns: `true` if the Lambda function exists. Otherwise `false`.
    func doesLambdaFunctionExist(lambdaClient: LambdaClient, name: String) async
    -> Bool {
        do {
            _ = try await lambdaClient.getFunction(
                input: GetFunctionInput(functionName: name)
            )
        } catch {
            return false
        }

        return true
    }

    /// Create the specified AWS Lambda function.
    ///
    /// - Parameters:
    ///   - lambdaClient: The `LambdaClient` to use.
    ///   - functionName: The name of the AWS Lambda function to create.
    ///   - roleArn: The ARN of the role to apply to the function.
    ///   - path: The path of the Zip archive containing the function.
    ///
    /// - Returns: `true` if the AWS Lambda was successfully created; `false`
    ///   if it wasn't.

```



```
func createFunction(lambdaClient: LambdaClient, functionName: String,
                    roleArn: String?, path: String) async throws ->
Bool {
    do {
        // Read the Zip archive containing the AWS Lambda function.

        let zipUrl = URL(fileURLWithPath: path)
        let zipData = try Data(contentsOf: zipUrl)

        // Create the AWS Lambda function that runs the specified code,
        // using the name given on the command line. The Lambda function
        // will run using the Amazon Linux 2 runtime.

        _ = try await lambdaClient.createFunction(
            input: CreateFunctionInput(
                code: LambdaClientTypes.FunctionCode(zipFile: zipData),
                functionName: functionName,
                handler: "handle",
                role: roleArn,
                runtime: .providedal2
            )
        )
    } catch {
        print("*** Error creating Lambda function:")
        dump(error)
        return false
    }

    // Wait for a while to be sure the function is done being created.

    let output = try await lambdaClient.waitUntilFunctionActiveV2(
        options: WaiterOptions(
            maxWaitTime: 20,
            minDelay: 0.5,
            maxDelay: 2
        ),
        input: GetFunctionInput(functionName: functionName)
    )

    switch output.result {
    case .success:
        return true
    case .failure:
        return false
    }
```

```
    }  
  }  
  
  /// Update the AWS Lambda function with new code to run when the function  
  /// is invoked.  
  ///  
  /// - Parameters:  
  ///   - lambdaClient: The `LambdaClient` to use.  
  ///   - functionName: The name of the AWS Lambda function to update.  
  ///   - path: The pathname of the Zip file containing the packaged Lambda  
  ///     function.  
  /// - Throws: `ExampleError.zipFileReadError`  
  /// - Returns: `true` if the function's code is updated successfully.  
  ///   Otherwise, returns `false`.  
  func updateFunctionCode(lambdaClient: LambdaClient, functionName: String,  
                           path: String) async throws -> Bool {  
    let zipUrl = URL(fileURLWithPath: path)  
    let zipData: Data  
  
    // Read the function's Zip file.  
  
    do {  
      zipData = try Data(contentsOf: zipUrl)  
    } catch {  
      throw ExampleError.zipFileReadError  
    }  
  
    // Update the function's code and wait for the updated version to be  
    // ready for use.  
  
    do {  
      _ = try await lambdaClient.updateFunctionCode(  
        input: UpdateFunctionCodeInput(  
          functionName: functionName,  
          zipFile: zipData  
        )  
      )  
    } catch {  
      return false  
    }  
  
    let output = try await lambdaClient.waitUntilFunctionUpdatedV2(  
      options: WaiterOptions(  
        maxWaitTime: 20,
```

```

        minDelay: 0.5,
        maxDelay: 2
    ),
    input: GetFunctionInput(
        functionName: functionName
    )
)

switch output.result {
    case .success:
        return true
    case .failure:
        return false
}
}

/// Tell the server-side component to log debug output by setting its
/// environment's `LOG_LEVEL` to `DEBUG`.
///
/// - Parameters:
///   - lambdaClient: The `LambdaClient` to use.
///   - functionName: The name of the AWS Lambda function to enable debug
///     logging for.
///
/// - Throws: `ExampleError.environmentResponseMissingError`,
///   `ExampleError.updateFunctionConfigurationError`,
///   `ExampleError.environmentVariablesMissingError`,
///   `ExampleError.logLevelIncorrectError`,
///   `ExampleError.updateFunctionConfigurationError`
func enableDebugLogging(lambdaClient: LambdaClient, functionName: String)
async throws {
    let envVariables = [
        "LOG_LEVEL": "DEBUG"
    ]
    let environment = LambdaClientTypes.Environment(variables: envVariables)

    do {
        let output = try await lambdaClient.updateFunctionConfiguration(
            input: UpdateFunctionConfigurationInput(
                environment: environment,
                functionName: functionName
            )
        )
    }
}

```

```

        guard let response = output.environment else {
            throw ExampleError.environmentResponseMissingError
        }

        if response.error != nil {
            throw ExampleError.updateFunctionConfigurationError
        }

        guard let retVariables = response.variables else {
            throw ExampleError.environmentVariablesMissingError
        }

        for envVar in retVariables {
            if envVar.key == "LOG_LEVEL" && envVar.value != "DEBUG" {
                print("*** Log level is not set to DEBUG!")
                throw ExampleError.logLevelIncorrectError
            }
        }
    } catch {
        throw ExampleError.updateFunctionConfigurationError
    }
}

/// Returns an array containing the names of all AWS Lambda functions
/// available to the user.
///
/// - Parameter lambdaClient: The `IAMClient` to use.
///
/// - Throws: `ExampleError.listFunctionsError`
///
/// - Returns: An array of lambda function name strings.
func getFunctionNames(lambdaClient: LambdaClient) async throws -> [String] {
    let pages = lambdaClient.listFunctionsPaginated(
        input: ListFunctionsInput()
    )

    var functionNames: [String] = []

    for try await page in pages {
        guard let functions = page.functions else {
            throw ExampleError.listFunctionsError
        }

        for function in functions {

```

```

        functionNames.append(function.functionName ?? "<unknown>")
    }
}

return functionNames
}

/// Invoke the Lambda function to increment a value.
///
/// - Parameters:
///   - lambdaClient: The `IAMClient` to use.
///   - number: The number to increment.
///
/// - Throws: `ExampleError.noAnswerReceived`, `ExampleError.invokeError`
///
/// - Returns: An integer number containing the incremented value.
func invokeIncrement(lambdaClient: LambdaClient, number: Int) async throws ->
Int {
    do {
        let incRequest = IncrementRequest(action: "increment", number:
number)

        let incData = try! JSONEncoder().encode(incRequest)

        // Invoke the lambda function.

        let invokeOutput = try await lambdaClient.invoke(
            input: InvokeInput(
                functionName: "lambda-basics-function",
                payload: incData
            )
        )

        let response = try! JSONDecoder().decode(Response.self,
from:invokeOutput.payload!)

        guard let answer = response.answer else {
            throw ExampleError.noAnswerReceived
        }
        return answer

    } catch {
        throw ExampleError.invokeError
    }
}

```

```
/// Invoke the calculator Lambda function.
///
/// - Parameters:
///   - lambdaClient: The `IAMClient` to use.
///   - action: Which arithmetic operation to perform: "plus", "minus",
///     "times", or "divided-by".
///   - x: The first number to use in the computation.
///   - y: The second number to use in the computation.
///
/// - Throws: `ExampleError.noAnswerReceived`, `ExampleError.invokeError`
///
/// - Returns: The computed answer as an `Int`.
func invokeCalculator(lambdaClient: LambdaClient, action: String, x: Int, y:
Int) async throws -> Int {
    do {
        let calcRequest = CalculatorRequest(action: action, x: x, y: y)
        let calcData = try! JSONEncoder().encode(calcRequest)

        // Invoke the lambda function.

        let invokeOutput = try await lambdaClient.invoke(
            input: InvokeInput(
                functionName: "lambda-basics-function",
                payload: calcData
            )
        )

        let response = try! JSONDecoder().decode(Response.self,
from:invokeOutput.payload!)

        guard let answer = response.answer else {
            throw ExampleError.noAnswerReceived
        }
        return answer

    } catch {
        throw ExampleError.invokeError
    }
}

/// Perform the example's tasks.
func basics() async throws {
```

```
let iamClient = try await IAMClient(
    config: IAMClient.IAMClientConfiguration(region: region)
)

let lambdaClient = try await LambdaClient(
    config: LambdaClient.LambdaClientConfiguration(region: region)
)

/// The IAM role to use for the example.
var iamRole: IAMClientTypes.Role

// Look for the specified role. If it already exists, use it. If not,
// create it and attach the desired policy to it.

do {
    iamRole = try await getRole(iamClient: iamClient, roleName: role)
} catch ExampleError.roleNotFound {
    // The role wasn't found, so create it and attach the needed
    // policy.

    iamRole = try await createRoleForLambda(iamClient: iamClient,
roleName: role)

    do {
        _ = try await iamClient.attachRolePolicy(
            input: AttachRolePolicyInput(policyArn: policyARN, roleName:
role)
        )
    } catch {
        throw ExampleError.policyError
    }
}

// Give the policy time to attach to the role.

sleep(5)

// Look to see if the function already exists. If it does, throw an
// error.

if await doesLambdaFunctionExist(lambdaClient: lambdaClient, name:
basicsFunctionName) {
    throw ExampleError.functionAlreadyExists
}
```

```
// Create, then invoke, the "increment" version of the calculator
// function.

print("Creating the increment Lambda function...")
if try await createFunction(lambdaClient: lambdaClient, functionName:
basicsFunctionName,
                            roleArn: iamRole.arn, path: incpath) {
    print("Running increment function calls...")
    for number in 0...4 {
        do {
            let answer = try await invokeIncrement(lambdaClient:
lambdaClient, number: number)
            print("Increment \ \(number) = \ \(answer)")
        } catch {
            print("Error incrementing \ \(number): ",
error.localizedDescription)
        }
    }
} else {
    print("*** Failed to create the increment function.")
}

// Enable debug logging.

print("\nEnabling debug logging...")
try await enableDebugLogging(lambdaClient: lambdaClient, functionName:
basicsFunctionName)

// Change it to a basic arithmetic calculator. Then invoke it a few
// times.

print("\nReplacing the Lambda function with a calculator...")

if try await updateFunctionCode(lambdaClient: lambdaClient, functionName:
basicsFunctionName,
                                path: calcpath) {
    print("Running calculator function calls...")
    for x in [6, 10] {
        for y in [2, 4] {
            for action in ["plus", "minus", "times", "divided-by"] {
                do {
                    let answer = try await invokeCalculator(lambdaClient:
lambdaClient, action: action, x: x, y: y)
```



```

        print("\n(x) \n(action) \n(y) = \n(answer)")
    } catch {
        print("Error calculating \n(x) \n(action) \n(y): ",
error.localizedDescription)
    }
}
}
}

// List all lambda functions.

let functionNames = try await getFunctionNames(lambdaClient:
lambdaClient)

if functionNames.count > 0 {
    print("\nAWS Lambda functions available on your account:")
    for name in functionNames {
        print("  \n(name)")
    }
}

// Delete the lambda function.

print("Deleting lambda function...")

do {
    _ = try await lambdaClient.deleteFunction(
        input: DeleteFunctionInput(
            functionName: "lambda-basics-function"
        )
    )
} catch {
    print("Error: Unable to delete the function.")
}

// Detach the role from the policy, then delete the role.

print("Deleting the AWS IAM role...")

do {
    _ = try await iamClient.detachRolePolicy(
        input: DetachRolePolicyInput(
            policyArn: policyARN,

```

```

        roleName: role
    )
)
_ = try await iamClient.deleteRole(
    input: DeleteRoleInput(
        roleName: role
    )
)
} catch {
    throw ExampleError.deleteRoleError
}
}
}

// -MARK: - Entry point

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.basics()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}

/// Errors thrown by the example's functions.
enum ExampleError: Error {
    /// An AWS Lambda function with the specified name already exists.
    case functionAlreadyExists
    /// The specified role doesn't exist.
    case roleNotFound
    /// Unable to create the role.
    case roleCreateError
    /// Unable to delete the role.
    case deleteRoleError
    /// Unable to attach a policy to the role.
    case policyError

```

```
/// Unable to get the executable directory.
case executableNotFound
/// An error occurred creating a lambda function.
case createLambdaError
/// An error occurred invoking the lambda function.
case invokeError
/// No answer received from the invocation.
case noAnswerReceived
/// Unable to list the AWS Lambda functions.
case listFunctionsError
/// Unable to update the AWS Lambda function.
case updateFunctionError
/// Unable to update the function configuration.
case updateFunctionConfigurationError
/// The environment response is missing after an
/// UpdateEnvironmentConfiguration attempt.
case environmentResponseMissingError
/// The environment variables are missing from the EnvironmentResponse and
/// no errors occurred.
case environmentVariablesMissingError
/// The log level is incorrect after attempting to set it.
case logLevelIncorrectError
/// Unable to load the AWS Lambda function's Zip file.
case zipFileReadError

var errorDescription: String? {
    switch self {
    case .functionAlreadyExists:
        return "An AWS Lambda function with that name already exists."
    case .roleNotFound:
        return "The specified role doesn't exist."
    case .deleteRoleError:
        return "Unable to delete the AWS IAM role."
    case .roleCreateError:
        return "Unable to create the specified role."
    case .policyError:
        return "An error occurred attaching the policy to the role."
    case .executableNotFound:
        return "Unable to find the executable program directory."
    case .createLambdaError:
        return "An error occurred creating a lambda function."
    case .invokeError:
        return "An error occurred invoking a lambda function."
    case .noAnswerReceived:
```

```
        return "No answer received from the lambda function."
    case .listFunctionsError:
        return "Unable to list the AWS Lambda functions."
    case .updateFunctionError:
        return "Unable to update the AWS lambda function."
    case .updateFunctionConfigurationError:
        return "Unable to update the AWS lambda function configuration."
    case .environmentResponseMissingError:
        return "The environment is missing from the response after updating
the function configuration."
    case .environmentVariablesMissingError:
        return "While no error occurred, no environment variables were
returned following function configuration."
    case .logLevelIncorrectError:
        return "The log level is incorrect after attempting to set it to
DEBUG."
    case .zipFileReadError:
        return "Unable to read the AWS Lambda function."
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API SDK AWS per Swift.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Azioni per l'utilizzo di Lambda AWS SDK

I seguenti esempi di codice mostrano come eseguire singole azioni Lambda con AWS gli SDK. Ogni esempio include un collegamento a GitHub, dove è possibile trovare le istruzioni per la configurazione e l'esecuzione del codice.

Questi estratti chiamano l'API Lambda e sono estratti di codice da programmi più grandi che devono essere eseguiti in modo contestuale. È possibile visualizzare le azioni nel contesto in [Scenari per l'utilizzo di Lambda AWS SDK](#).

Gli esempi seguenti includono solo le azioni più comunemente utilizzate. Per un elenco completo, consulta la [documentazione di riferimento dell'API AWS Lambda](#).

### Esempi

- [Utilizzare CreateAlias con una CLI](#)
- [Utilizzare CreateFunction con un AWS SDK o CLI](#)
- [Utilizzare DeleteAlias con una CLI](#)
- [Utilizzare DeleteFunction con un AWS SDK o CLI](#)
- [Utilizzare DeleteFunctionConcurrency con una CLI](#)
- [Utilizzare DeleteProvisionedConcurrencyConfig con una CLI](#)
- [Utilizzare GetAccountSettings con una CLI](#)
- [Utilizzare GetAlias con una CLI](#)
- [Utilizzare GetFunction con un AWS SDK o CLI](#)
- [Utilizzare GetFunctionConcurrency con una CLI](#)
- [Utilizzare GetFunctionConfiguration con una CLI](#)
- [Utilizzare GetPolicy con una CLI](#)
- [Utilizzare GetProvisionedConcurrencyConfig con una CLI](#)
- [Usa Invoke con un AWS SDK o CLI](#)
- [Utilizzare ListFunctions con un AWS SDK o CLI](#)
- [Utilizzare ListProvisionedConcurrencyConfigs con una CLI](#)
- [Utilizzare ListTags con una CLI](#)
- [Utilizzare ListVersionsByFunction con una CLI](#)
- [Utilizzare PublishVersion con una CLI](#)

- [Utilizzare PutFunctionConcurrency con una CLI](#)
- [Utilizzare PutProvisionedConcurrencyConfig con una CLI](#)
- [Utilizzare RemovePermission con una CLI](#)
- [Utilizzare TagResource con una CLI](#)
- [Utilizzare UntagResource con una CLI](#)
- [Utilizzare UpdateAlias con una CLI](#)
- [Utilizzare UpdateFunctionCodecon un AWS SDK o CLI](#)
- [Utilizzare UpdateFunctionConfigurationcon un AWS SDK o CLI](#)

## Utilizzare **CreateAlias** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare CreateAlias.

### CLI

#### AWS CLI

Per creare un alias per una funzione Lambda

L'esempio create-alias seguente crea un alias denominato LIVE che punta alla versione 1 della funzione Lambda my-function.

```
aws lambda create-alias \  
  --function-name my-function \  
  --description "alias for live version of function" \  
  --function-version 1 \  
  --name LIVE
```

Output:

```
{  
  "FunctionVersion": "1",  
  "Name": "LIVE",  
  "AliasArn": "arn:aws:lambda:us-west-2:123456789012:function:my-  
function:LIVE",  
  "RevisionId": "873282ed-4cd3-4dc8-a069-d0c647e470c6",  
  "Description": "alias for live version of function"  
}
```

Per ulteriori informazioni, consulta [Configurazione degli alias delle funzioni AWS Lambda](#) nella AWS Lambda Developer Guide.

- Per i dettagli sull'API, consulta [CreateAlias](#) Command Reference. AWS CLI

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio crea un nuovo alias Lambda per la versione e la configurazione di routing specificate per specificare la percentuale di richieste di invocazione ricevute.

```
New-LMAlias -FunctionName "MylambdaFunction123" -  
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6"} -Description "Alias  
for version 4" -FunctionVersion 4 -Name "PowershellAlias"
```

- Per i dettagli sull'API, vedere [CreateAlias](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: questo esempio crea un nuovo alias Lambda per la versione e la configurazione di routing specificate per specificare la percentuale di richieste di invocazione ricevute.

```
New-LMAlias -FunctionName "MylambdaFunction123" -  
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6"} -Description "Alias  
for version 4" -FunctionVersion 4 -Name "PowershellAlias"
```

- Per i dettagli sull'API, vedere [CreateAlias](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **CreateFunction** con un AWS SDK o CLI

Gli esempi di codice seguenti mostrano come utilizzare `CreateFunction`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- [Informazioni di base](#)
- [Crea un'API REST con integrazione con proxy di funzioni](#)
- [Creazione di una dashboard di monitoraggio con il nome della funzione come variabile](#)
- [Creazione della tua prima funzione serverless](#)
- [Utilizzo delle variabili di proprietà nei dashboard di monitoraggio per monitorare più funzioni serverless](#)

## .NET

### SDK per .NET

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Creates a new Lambda function.
/// </summary>
/// <param name="functionName">The name of the function.</param>
/// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
/// bucket where the zip file containing the code is located.</param>
/// <param name="s3Key">The Amazon S3 key of the zip file.</param>
/// <param name="role">The Amazon Resource Name (ARN) of a role with the
/// appropriate Lambda permissions.</param>
/// <param name="handler">The name of the handler function.</param>
/// <returns>The Amazon Resource Name (ARN) of the newly created
/// Lambda function.</returns>
public async Task<string> CreateLambdaFunctionAsync(
    string functionName,
    string s3Bucket,
    string s3Key,
    string role,
    string handler)
{
    // Defines the location for the function code.
    // S3Bucket - The S3 bucket where the file containing
    //           the source code is stored.
    // S3Key     - The name of the file containing the code.
```



```
var functionCode = new FunctionCode
{
    S3Bucket = s3Bucket,
    S3Key = s3Key,
};

var createFunctionRequest = new CreateFunctionRequest
{
    FunctionName = functionName,
    Description = "Created by the Lambda .NET API",
    Code = functionCode,
    Handler = handler,
    Runtime = Runtime.Dotnet6,
    Role = role,
};

var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
return reponse.FunctionArn;
}
```

- Per i dettagli sull'API, consulta la [CreateFunction](#) sezione AWS SDK per .NET API Reference.

## C++

### SDK per C++

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
// (overrides config file).
// clientConfig.region = "us-east-1";
```

```

    Aws::Lambda::LambdaClient client(clientConfig);

    Aws::Lambda::Model::CreateFunctionRequest request;
    request.SetFunctionName(LAMBDA_NAME);
    request.SetDescription(LAMBDA_DESCRIPTION); // Optional.
#if USE_CPP_LAMBDA_FUNCTION
    request.SetRuntime(Aws::Lambda::Model::Runtime::provided_al2);
    request.SetTimeout(15);
    request.SetMemorySize(128);

    // Assume the AWS Lambda function was built in Docker with same
    architecture
    // as this code.
#if defined(__x86_64__)
    request.SetArchitectures({Aws::Lambda::Model::Architecture::x86_64});
#elif defined(__aarch64__)
    request.SetArchitectures({Aws::Lambda::Model::Architecture::arm64});
#else
#error "Unimplemented architecture"
#endif // defined(architecture)
#else
    request.SetRuntime(Aws::Lambda::Model::Runtime::python3_9);
#endif

    request.SetRole(roleArn);
    request.SetHandler(LAMBDA_HANDLER_NAME);
    request.SetPublish(true);
    Aws::Lambda::Model::FunctionCode code;
    std::ifstream ifstream(INCREMENT_LAMBDA_CODE.c_str(),
                          std::ios_base::in | std::ios_base::binary);
    if (!ifstream.is_open()) {
        std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
std::endl;

#if USE_CPP_LAMBDA_FUNCTION
        std::cerr
            << "The cpp Lambda function must be built following the
instructions in the cpp_lambda/README.md file. "
            << std::endl;
#endif

        deleteIamRole(clientConfig);
        return false;
    }

    Aws::StringStream buffer;

```

```

        buffer << ifstream.rdbuf();

        code.SetZipFile(Aws::Utils::ByteBuffer((unsigned char *)
buffer.str().c_str(),
                                                    buffer.str().length()));

        request.SetCode(code);

        Aws::Lambda::Model::CreateFunctionOutcome outcome =
client.CreateFunction(
            request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda function was successfully created. " <<
seconds
                        << " seconds elapsed." << std::endl;
            break;
        }

        else {
            std::cerr << "Error with CreateFunction. "
                        << outcome.GetError().GetMessage()
                        << std::endl;
            deleteIamRole(clientConfig);
            return false;
        }
    }
}

```

- Per i dettagli sull'API, consulta la [CreateFunction](#) sezione AWS SDK per C++ API Reference.

## CLI

### AWS CLI

Come creare una funzione Lambda

L'esempio di `create-function` seguente crea una funzione Lambda denominata `my-function`.

```

aws lambda create-function \
  --function-name my-function \
  --runtime nodejs22.x \
  --zip-file fileb://my-function.zip \
  --handler my-function.handler \

```

```
--role arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-tges6bf4
```

Contenuto di my-function.zip:

This file is a deployment package that contains your function code and any dependencies.

Output:

```
{
  "TracingConfig": {
    "Mode": "PassThrough"
  },
  "CodeSha256": "PFn4S+er27qk+UuZSTKEQfNKG/XNn7QJs90mJgq6oH8=",
  "FunctionName": "my-function",
  "CodeSize": 308,
  "RevisionId": "873282ed-4cd3-4dc8-a069-d0c647e470c6",
  "MemorySize": 128,
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
  "Version": "$LATEST",
  "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-zgur6bf4",
  "Timeout": 3,
  "LastModified": "2025-10-14T22:26:11.234+0000",
  "Handler": "my-function.handler",
  "Runtime": "nodejs22.x",
  "Description": ""
}
```

Per ulteriori informazioni, consulta [Configurazione della memoria della funzione Lambda](#) nella Guida per gli sviluppatori Lambda AWS .

- Per i dettagli sull'API, consulta [CreateFunction AWS CLI](#) Command Reference.

## Go

## SDK per Go V2

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
import (  
    "bytes"  
    "context"  
    "encoding/json"  
    "errors"  
    "log"  
    "time"  
  
    "github.com/aws/aws-sdk-go-v2/aws"  
    "github.com/aws/aws-sdk-go-v2/service/lambda"  
    "github.com/aws/aws-sdk-go-v2/service/lambda/types"  
)  
  
// FunctionWrapper encapsulates function actions used in the examples.  
// It contains an AWS Lambda service client that is used to perform user actions.  
type FunctionWrapper struct {  
    LambdaClient *lambda.Client  
}  
  
// CreateFunction creates a new Lambda function from code contained in the  
// zipPackage  
// buffer. The specified handlerName must match the name of the file and function  
// contained in the uploaded code. The role specified by iamRoleArn is assumed by  
// Lambda and grants specific permissions.  
// When the function already exists, types.StateActive is returned.  
// When the function is created, a lambda.FunctionActiveV2Waiter is used to wait  
// until the  
// function is active.
```

```

func (wrapper FunctionWrapper) CreateFunction(ctx context.Context, functionName
string, handlerName string,
iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
var state types.State
_, err := wrapper.LambdaClient.CreateFunction(ctx, &lambda.CreateFunctionInput{
    Code:          &types.FunctionCode{ZipFile: zipPackage.Bytes()},
    FunctionName:  aws.String(functionName),
    Role:          iamRoleArn,
    Handler:       aws.String(handlerName),
    Publish:       true,
    Runtime:       types.RuntimePython39,
})
if err != nil {
    var resConflict *types.ResourceConflictException
    if errors.As(err, &resConflict) {
        log.Printf("Function %v already exists.\n", functionName)
        state = types.StateActive
    } else {
        log.Panicf("Couldn't create function %v. Here's why: %v\n", functionName, err)
    }
} else {
    waiter := lambda.NewFunctionActiveV2Waiter(wrapper.LambdaClient)
    funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
        FunctionName: aws.String(functionName)}, 1*time.Minute)
    if err != nil {
        log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
functionName, err)
    } else {
        state = funcOutput.Configuration.State
    }
}
return state
}

```

- Per i dettagli sull'API, consulta la [CreateFunction](#) sezione AWS SDK per Go API Reference.

## Java

### SDK per Java 2.x

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/**
 * Creates a new Lambda function in AWS using the AWS Lambda Java API.
 *
 * @param awsLambda the AWS Lambda client used to interact with the AWS
Lambda service
 * @param functionName the name of the Lambda function to create
 * @param key the S3 key of the function code
 * @param bucketName the name of the S3 bucket containing the function code
 * @param role the IAM role to assign to the Lambda function
 * @param handler the fully qualified class name of the function handler
 * @return the Amazon Resource Name (ARN) of the created Lambda function
 */
public static String createLambdaFunction(LambdaClient awsLambda,
                                         String functionName,
                                         String key,
                                         String bucketName,
                                         String role,
                                         String handler) {

    try {
        LambdaWaiter waiter = awsLambda.waiter();
        FunctionCode code = FunctionCode.builder()
            .s3Key(key)
            .s3Bucket(bucketName)
            .build();

        CreateFunctionRequest functionRequest =
CreateFunctionRequest.builder()
            .functionName(functionName)
            .description("Created by the Lambda Java API")
            .code(code)
            .handler(handler)
```

```

        .runtime(Runtime.JAVA17)
        .role(role)
        .build();

        // Create a Lambda function using a waiter
        CreateFunctionResponse functionResponse =
awsLambda.createFunction(functionRequest);
        GetFunctionRequest getFunctionRequest = GetFunctionRequest.builder()
            .functionName(functionName)
            .build();
        WaiterResponse<GetFunctionResponse> waiterResponse =
waiter.waitUntilFunctionExists(getFunctionRequest);
        waiterResponse.matched().response().ifPresent(System.out::println);
        return functionResponse.functionArn();

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
    return "";
}

```

- Per i dettagli sull'API, consulta la [CreateFunction](#) sezione AWS SDK for Java 2.x API Reference.

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

const createFunction = async (funcName, roleArn) => {
    const client = new LambdaClient({});
    const code = await readFile(`${dirname}../functions/${funcName}.zip`);

    const command = new CreateFunctionCommand({

```



```
Code: { ZipFile: code },
FunctionName: funcName,
Role: roleArn,
Architectures: [Architecture.arm64],
Handler: "index.handler", // Required when sending a .zip file
PackageType: PackageType.Zip, // Required when sending a .zip file
Runtime: Runtime.nodejs16x, // Required when sending a .zip file
});

return client.send(command);
};
```

- Per i dettagli sull'API, consulta la [CreateFunction](#) sezione AWS SDK per JavaScript API Reference.

## Kotlin

### SDK per Kotlin

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
suspend fun createNewFunction(
    myFunctionName: String,
    s3BucketName: String,
    myS3Key: String,
    myHandler: String,
    myRole: String,
): String? {
    val functionCode =
        FunctionCode {
            s3Bucket = s3BucketName
            s3Key = myS3Key
        }

    val request =
        CreateFunctionRequest {
```

```

        functionName = myFunctionName
        code = functionCode
        description = "Created by the Lambda Kotlin API"
        handler = myHandler
        role = myRole
        runtime = Runtime.Java17
    }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        val functionResponse = awsLambda.createFunction(request)
        awsLambda.waitForFunctionActive {
            functionName = myFunctionName
        }
        return functionResponse.functionArn
    }
}

```

- Per i dettagli sull'API, [CreateFunction](#) consulta AWS SDK for Kotlin API reference.

## PHP

### SDK per PHP

#### Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

public function createFunction($functionName, $role, $bucketName, $handler)
{
    //This assumes the Lambda function is in an S3 bucket.
    return $this->customWaiter(function () use ($functionName, $role,
$bucketName, $handler) {
        return $this->lambdaClient->createFunction([
            'Code' => [
                'S3Bucket' => $bucketName,
                'S3Key' => $functionName,
            ],
            'FunctionName' => $functionName,
            'Role' => $role['Arn'],

```

```

        'Runtime' => 'python3.9',
        'Handler' => "$handler.lambda_handler",
    });
});
}

```

- Per i dettagli sull'API, consulta la [CreateFunction](#) sezione AWS SDK per PHP API Reference.

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: Questo esempio crea una nuova funzione C# (dotnetcore1.0 runtime) denominata in MyFunction AWS Lambda, che fornisce i file binari compilati per la funzione da un file zip sul file system locale (è possibile utilizzare percorsi relativi o assoluti). Le funzioni C# Lambda specificano il gestore per la funzione utilizzando la AssemblyName designazione:::Namespace.ClassName.MethodName È necessario sostituire in modo appropriato le parti relative al nome dell'assembly (senza il suffisso .dll), al namespace, al nome della classe e al nome del metodo delle specifiche dell'handler. La nuova funzione avrà le variabili di ambiente 'envvar1' e 'envvar2' impostate in base ai valori forniti.

```

Publish-LMFunction -Description "My C# Lambda Function" `
    -FunctionName MyFunction `
    -ZipFilename .\MyFunctionBinaries.zip `
    -Handler "AssemblyName::Namespace.ClassName::MethodName" `
    -Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" `
    -Runtime dotnetcore1.0 `
    -Environment_Variable @{ "envvar1"="value";"envvar2"="value" }

```

### Output:

```

CodeSha256      : /NgBMd...gq71I=
CodeSize        : 214784
DeadLetterConfig :
Description      : My C# Lambda Function
Environment      : Amazon.Lambda.Model.EnvironmentResponse
FunctionArn      : arn:aws:lambda:us-west-2:123456789012:function:ToUpper
FunctionName     : MyFunction
Handler         : AssemblyName::Namespace.ClassName::MethodName

```

```

KMSKeyArn      :
LastModified   : 2016-12-29T23:50:14.207+0000
MemorySize     : 128
Role           : arn:aws:iam::123456789012:role/LambdaFullExecRole
Runtime        : dotnetcore1.0
Timeout        : 3
Version        : $LATEST
VpcConfig      :

```

Esempio 2: questo esempio è simile a quello precedente, tranne per il fatto che i binari delle funzioni vengono prima caricati in un bucket Amazon S3 (che deve trovarsi nella stessa regione della funzione Lambda prevista) e l'oggetto S3 risultante viene quindi referenziato durante la creazione della funzione.

```

Write-S3Object -BucketName amzn-s3-demo-bucket -Key MyFunctionBinaries.zip -
File .\MyFunctionBinaries.zip
Publish-LMFunction -Description "My C# Lambda Function" `
    -FunctionName MyFunction `
    -BucketName amzn-s3-demo-bucket `
    -Key MyFunctionBinaries.zip `
    -Handler "AssemblyName::Namespace.ClassName::MethodName" `
    -Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" `
    -Runtime dotnetcore1.0 `
    -Environment_Variable @{ "envvar1"="value";"envvar2"="value" }

```

- Per i dettagli sull'API, vedere [CreateFunction](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

## Strumenti per V5 PowerShell

Esempio 1: Questo esempio crea una nuova funzione C# (dotnetcore1.0 runtime) denominata in MyFunction AWS Lambda, che fornisce i file binari compilati per la funzione da un file zip sul file system locale (è possibile utilizzare percorsi relativi o assoluti). Le funzioni C# Lambda specificano il gestore per la funzione utilizzando la AssemblyName designazione:::.. Namespace.ClassName MethodName È necessario sostituire in modo appropriato le parti relative al nome dell'assembly (senza il suffisso .dll), al namespace, al nome della classe e al nome del metodo delle specifiche dell'handler. La nuova funzione avrà le variabili di ambiente 'envvar1' e 'envvar2' impostate in base ai valori forniti.

```

Publish-LMFunction -Description "My C# Lambda Function" `
    -FunctionName MyFunction `
    -ZipFilename .\MyFunctionBinaries.zip `

```

```
-Handler "AssemblyName::Namespace.ClassName::MethodName" `
-Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" `
-Runtime dotnetcore1.0 `
-Environment_Variable @{ "envvar1"="value";"envvar2"="value" }
```

### Output:

```
CodeSha256      : /NgBMd...gq71I=
CodeSize       : 214784
DeadLetterConfig :
Description     : My C# Lambda Function
Environment     : Amazon.Lambda.Model.EnvironmentResponse
FunctionArn     : arn:aws:lambda:us-west-2:123456789012:function:ToUpper
FunctionName    : MyFunction
Handler        : AssemblyName::Namespace.ClassName::MethodName
KMSKeyArn      :
LastModified    : 2016-12-29T23:50:14.207+0000
MemorySize     : 128
Role           : arn:aws:iam::123456789012:role/LambdaFullExecRole
Runtime        : dotnetcore1.0
Timeout        : 3
Version        : $LATEST
VpcConfig      :
```

Esempio 2: questo esempio è simile a quello precedente, tranne per il fatto che i binari delle funzioni vengono prima caricati in un bucket Amazon S3 (che deve trovarsi nella stessa regione della funzione Lambda prevista) e l'oggetto S3 risultante viene quindi referenziato durante la creazione della funzione.

```
Write-S3Object -BucketName amzn-s3-demo-bucket -Key MyFunctionBinaries.zip -
File .\MyFunctionBinaries.zip
Publish-LMFunction -Description "My C# Lambda Function" `
-FunctionName MyFunction `
-BucketName amzn-s3-demo-bucket `
-Key MyFunctionBinaries.zip `
-Handler "AssemblyName::Namespace.ClassName::MethodName" `
-Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" `
-Runtime dotnetcore1.0 `
-Environment_Variable @{ "envvar1"="value";"envvar2"="value" }
```

- Per i dettagli sull'API, vedere [CreateFunction](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

## Python

### SDK per Python (Boto3)

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def create_function(
        self, function_name, handler_name, iam_role, deployment_package
    ):
        """
        Deploys a Lambda function.

        :param function_name: The name of the Lambda function.
        :param handler_name: The fully qualified name of the handler function.
        This
                               must include the file name and the function name.
        :param iam_role: The IAM role to use for the function.
        :param deployment_package: The deployment package that contains the
        function
                               code in .zip format.
        :return: The Amazon Resource Name (ARN) of the newly created function.
        """
        try:
            response = self.lambda_client.create_function(
                FunctionName=function_name,
                Description="AWS Lambda doc example",
                Runtime="python3.9",
                Role=iam_role.arn,
                Handler=handler_name,
                Code={"ZipFile": deployment_package},
                Publish=True,
            )
```

```

        function_arn = response["FunctionArn"]
        waiter = self.lambda_client.get_waiter("function_active_v2")
        waiter.wait(FunctionName=function_name)
        logger.info(
            "Created function '%s' with ARN: '%s'.",
            function_name,
            response["FunctionArn"],
        )
    except ClientError:
        logger.error("Couldn't create function %s.", function_name)
        raise
    else:
        return function_arn

```

- Per i dettagli sull'API, consulta [CreateFunction AWS SDK for Python \(Boto3\) API Reference](#).

## Ruby

### SDK per Ruby

#### Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Deploys a Lambda function.
  #
  # @param function_name: The name of the Lambda function.

```

```

# @param handler_name: The fully qualified name of the handler function.
# @param role_arn: The IAM role to use for the function.
# @param deployment_package: The deployment package that contains the function
code in .zip format.
# @return: The Amazon Resource Name (ARN) of the newly created function.
def create_function(function_name, handler_name, role_arn, deployment_package)
  response = @lambda_client.create_function({
    role: role_arn.to_s,
    function_name: function_name,
    handler: handler_name,
    runtime: 'ruby2.7',
    code: {
      zip_file: deployment_package
    },
    environment: {
      variables: {
        'LOG_LEVEL' => 'info'
      }
    }
  })

  @lambda_client.wait_until(:function_active_v2, { function_name:
function_name }) do |w|
    w.max_attempts = 5
    w.delay = 5
  end
  response
rescue Aws::Lambda::Errors::ServiceException => e
  @logger.error("There was an error creating #{function_name}:\n #{e.message}")
rescue Aws::Waiters::Errors::WaiterFailed => e
  @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
end

```

- Per i dettagli sull'API, consulta la [CreateFunction](#) sezione AWS SDK per Ruby API Reference.



## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/**
 * Create a function, uploading from a zip file.
 */
pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
anyhow::Error> {
    let code = self.prepare_function(zip_file, None).await?;

    let key = code.s3_key().unwrap().to_string();

    let role = self.create_role().await.map_err(|e| anyhow!(e))?;

    info!("Created iam role, waiting 15s for it to become active");
    tokio::time::sleep(Duration::from_secs(15)).await;

    info!("Creating lambda function {}", self.lambda_name);
    let _ = self
        .lambda_client
        .create_function()
        .function_name(self.lambda_name.clone())
        .code(code)
        .role(role.arn())
        .runtime(aws_sdk_lambda::types::Runtime::ProvidedAl2)
        .handler("_unused")
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;

    self.lambda_client
        .publish_version()
        .function_name(self.lambda_name.clone())
```

```

        .send()
        .await?;

    Ok(key)
}

/**
 * Upload function code from a path to a zip file.
 * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
 * The easiest way to create such a zip is to use `cargo lambda build --
output-format Zip`.
 */
async fn prepare_function(
    &self,
    zip_file: PathBuf,
    key: Option<String>,
) -> Result<FunctionCode, anyhow::Error> {
    let body = ByteStream::from_path(zip_file).await?;

    let key = key.unwrap_or_else(|| format!("{}",_code", self.lambda_name));

    info!("Uploading function code to s3://{}/{}", self.bucket, key);
    let _ = self
        .s3_client
        .put_object()
        .bucket(self.bucket.clone())
        .key(key.clone())
        .body(body)
        .send()
        .await?;

    Ok(FunctionCode::builder()
        .s3_bucket(self.bucket.clone())
        .s3_key(key)
        .build())
}

```

- Per i dettagli sulle API, consulta la [CreateFunction](#) guida di riferimento all'API AWS SDK for Rust.

## SAP ABAP

## SDK per SAP ABAP

 Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

TRY.
    lo_lmd->createfunction(
        iv_functionname = iv_function_name
        iv_runtime = `python3.9`
        iv_role = iv_role_arn
        iv_handler = iv_handler
        io_code = io_zip_file
        iv_description = 'AWS Lambda code example' ).
    MESSAGE 'Lambda function created.' TYPE 'I'.
CATCH /aws1/cx_lmdcodesigningcfn00.
    MESSAGE 'Code signing configuration does not exist.' TYPE 'E'.
CATCH /aws1/cx_lmdcodestorageexcdex.
    MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
CATCH /aws1/cx_lmdcodeverification00.
    MESSAGE 'Code signature failed one or more validation checks for
signature mismatch or expiration.' TYPE 'E'.
CATCH /aws1/cx_lmdinvalidcodesigex.
    MESSAGE 'Code signature failed the integrity check.' TYPE 'E'.
CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_lmdresourceconflictex.
    MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
CATCH /aws1/cx_lmdresourcenotfoundex.
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.
CATCH /aws1/cx_lmdserviceexception.
    MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
CATCH /aws1/cx_lmdtoomanyrequestsex.
    MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
ENDTRY.

```

- Per i dettagli sulle API, [CreateFunction](#) consulta AWS SDK for SAP ABAP API reference.

## Swift

### SDK per Swift

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
import AWSClientRuntime
import AWSLambda
import Foundation

do {
    // Read the Zip archive containing the AWS Lambda function.

    let zipUrl = URL(fileURLWithPath: path)
    let zipData = try Data(contentsOf: zipUrl)

    // Create the AWS Lambda function that runs the specified code,
    // using the name given on the command line. The Lambda function
    // will run using the Amazon Linux 2 runtime.

    _ = try await lambdaClient.createFunction(
        input: CreateFunctionInput(
            code: LambdaClientTypes.FunctionCode(zipFile: zipData),
            functionName: functionName,
            handler: "handle",
            role: roleArn,
            runtime: .providedal2
        )
    )
} catch {
    print("*** Error creating Lambda function:")
    dump(error)
    return false
}
```

- Per i dettagli sull'API, consulta la [CreateFunction](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **DeleteAlias** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare DeleteAlias.

### CLI

#### AWS CLI

Per eliminare un alias di una funzione Lambda

L'esempio delete-alias seguente elimina l'alias denominato LIVE dalla funzione Lambda my-function.

```
aws lambda delete-alias \  
  --function-name my-function \  
  --name LIVE
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta [Configurazione degli alias delle funzioni AWS Lambda](#) nella AWS Lambda Developer Guide.

- Per i dettagli sull'API, consulta [DeleteAlias](#) Command Reference.AWS CLI

### PowerShell

#### Strumenti per PowerShell V4

Esempio 1: questo esempio elimina l'alias della funzione Lambda menzionato nel comando.

```
Remove-LMAlias -FunctionName "MyLambdaFunction123" -Name "NewAlias"
```

- Per i dettagli sull'API, vedere [DeleteAlias](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

## Strumenti per V5 PowerShell

Esempio 1: questo esempio elimina l'alias della funzione Lambda menzionato nel comando.

```
Remove-LMAlias -FunctionName "MyLambdaFunction123" -Name "NewAlias"
```

- Per i dettagli sull'API, vedere [DeleteAlias](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **DeleteFunction** con un AWS SDK o CLI

Gli esempi di codice seguenti mostrano come utilizzare `DeleteFunction`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- [Informazioni di base](#)
- [Crea un'API REST con integrazione con proxy di funzioni](#)
- [Creazione di una dashboard di monitoraggio con il nome della funzione come variabile](#)
- [Creazione della tua prima funzione serverless](#)
- [Utilizzo delle variabili di proprietà nei dashboard di monitoraggio per monitorare più funzioni serverless](#)

## .NET

### SDK per .NET

#### Note

C'è altro su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
```

```

    /// Delete an AWS Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// delete.</param>
    /// <returns>A Boolean value that indicates the success of the action.</
returns>
    public async Task<bool> DeleteFunctionAsync(string functionName)
    {
        var request = new DeleteFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.DeleteFunctionAsync(request);

        // A return value of NoContent means that the request was processed.
        // In this case, the function was deleted, and the return value
        // is intentionally blank.
        return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
    }

```

- Per i dettagli sull'API, consulta la [DeleteFunction](#) sezione AWS SDK per .NET API Reference.

## C++

### SDK per C++

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region in which the bucket was created
    (overrides config file).
    // clientConfig.region = "us-east-1";

```

```
Aws::Lambda::LambdaClient client(clientConfig);

Aws::Lambda::Model::DeleteFunctionRequest request;
request.SetFunctionName(LAMBDA_NAME);

Aws::Lambda::Model::DeleteFunctionOutcome outcome = client.DeleteFunction(
    request);

if (outcome.IsSuccess()) {
    std::cout << "The lambda function was successfully deleted." <<
std::endl;
}
else {
    std::cerr << "Error with Lambda::DeleteFunction. "
        << outcome.GetError().GetMessage()
        << std::endl;
}
```

- Per i dettagli sull'API, consulta la [DeleteFunction](#) sezione AWS SDK per C++ API Reference.

## CLI

### AWS CLI

Esempio 1: come eliminare una funzione Lambda in base al nome della funzione

L'esempio di `delete-function` seguente elimina la funzione Lambda denominata `my-function` specificandone il nome.

```
aws lambda delete-function \
    --function-name my-function
```

Questo comando non produce alcun output.

Esempio 2: eliminazione di una funzione Lambda in base all'ARN della funzione

L'esempio di `delete-function` seguente elimina la funzione Lambda denominata `my-function` specificando l'ARN della funzione.

```
aws lambda delete-function \
```



```
--function-name arn:aws:lambda:us-west-2:123456789012:function:my-function
```

Questo comando non produce alcun output.

Esempio 3: eliminazione di una funzione Lambda in base all'ARN parziale della funzione

L'esempio di `delete-function` seguente elimina la funzione Lambda denominata `my-function` specificando l'ARN parziale della funzione.

```
aws lambda delete-function \  
  --function-name 123456789012:function:my-function
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta [Configurazione delle funzioni Lambda AWS](#) nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta [DeleteFunction AWS CLI](#) Command Reference.

Go

SDK per Go V2

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
import (  
  "bytes"  
  "context"  
  "encoding/json"  
  "errors"  
  "log"  
  "time"  
  
  "github.com/aws/aws-sdk-go-v2/aws"  
  "github.com/aws/aws-sdk-go-v2/service/lambda"
```

```

    "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
    LambdaClient *lambda.Client
}

// DeleteFunction deletes the Lambda function specified by functionName.
func (wrapper FunctionWrapper) DeleteFunction(ctx context.Context, functionName
    string) {
    _, err := wrapper.LambdaClient.DeleteFunction(ctx, &lambda.DeleteFunctionInput{
        FunctionName: aws.String(functionName),
    })
    if err != nil {
        log.Panicf("Couldn't delete function %v. Here's why: %v\n", functionName, err)
    }
}

```

- Per i dettagli sull'API, consulta la [DeleteFunction](#) sezione AWS SDK per Go API Reference.

## Java

### SDK per Java 2.x

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/**
 * Deletes an AWS Lambda function.
 *
 * @param awsLambda      an instance of the {@link LambdaClient} class, which
 * is used to interact with the AWS Lambda service
 * @param functionName  the name of the Lambda function to be deleted

```

```

    *
    * @throws LambdaException if an error occurs while deleting the Lambda
function
    */
    public static void deleteLambdaFunction(LambdaClient awsLambda, String
functionName) {
        try {
            DeleteFunctionRequest request = DeleteFunctionRequest.builder()
                .functionName(functionName)
                .build();

            awsLambda.deleteFunction(request);
            System.out.println("The " + functionName + " function was deleted");

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}

```

- Per i dettagli sull'API, consulta la [DeleteFunction](#) sezione AWS SDK for Java 2.x API Reference.

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
    const client = new LambdaClient({});
    const command = new DeleteFunctionCommand({ FunctionName: funcName });
    return client.send(command);
}

```

```
};
```

- Per i dettagli sull'API, consulta la [DeleteFunction](#) sezione AWS SDK per JavaScript API Reference.

## Kotlin

### SDK per Kotlin

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
suspend fun dellLambdaFunction(myFunctionName: String) {  
    val request =  
        DeleteFunctionRequest {  
            functionName = myFunctionName  
        }  
  
    LambdaClient { region = "us-east-1" }.use { awsLambda ->  
        awsLambda.deleteFunction(request)  
        println("$myFunctionName was deleted")  
    }  
}
```

- Per i dettagli sull'API, [DeleteFunction](#) consulta AWS SDK for Kotlin API reference.

## PHP

### SDK per PHP

#### Note

C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
public function deleteFunction($functionName)
{
    return $this->lambdaClient->deleteFunction([
        'FunctionName' => $functionName,
    ]);
}
```

- Per i dettagli sull'API, consulta la [DeleteFunction](#) sezione AWS SDK per PHP API Reference.

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio elimina una versione specifica di una funzione Lambda

```
Remove-LMFunction -FunctionName "MylambdaFunction123" -Qualifier '3'
```

- Per i dettagli sull'API, vedere [DeleteFunction](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: questo esempio elimina una versione specifica di una funzione Lambda

```
Remove-LMFunction -FunctionName "MylambdaFunction123" -Qualifier '3'
```

- Per i dettagli sull'API, vedere [DeleteFunction](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

## Python

### SDK per Python (Boto3)

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def delete_function(self, function_name):
        """
        Deletes a Lambda function.

        :param function_name: The name of the function to delete.
        """
        try:
            self.lambda_client.delete_function(FunctionName=function_name)
        except ClientError:
            logger.exception("Couldn't delete function %s.", function_name)
            raise
```

- Per i dettagli sull'API, consulta [DeleteFunction AWS SDK for Python \(Boto3\) API Reference](#).

## Ruby

### SDK per Ruby

#### Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end
end
```

```

end

# Deletes a Lambda function.
# @param function_name: The name of the function to delete.
def delete_function(function_name)
  print "Deleting function: #{function_name}..."
  @lambda_client.delete_function(
    function_name: function_name
  )
  print 'Done!'.green
rescue Aws::Lambda::Errors::ServiceException => e
  @logger.error("There was an error deleting #{function_name}:\n #{e.message}")
end

```

- Per i dettagli sull'API, consulta la [DeleteFunction](#) sezione AWS SDK per Ruby API Reference.

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/** Delete a function and its role, and if possible or necessary, its
associated code object and bucket. */
pub async fn delete_function(
  &self,
  location: Option<String>,
) -> (
  Result<DeleteFunctionOutput, anyhow::Error>,
  Result<DeleteRoleOutput, anyhow::Error>,
  Option<Result<DeleteObjectOutput, anyhow::Error>>,
) {
  info!("Deleting lambda function {}", self.lambda_name);
  let delete_function = self
    .lambda_client

```

```

        .delete_function()
        .function_name(self.lambda_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from);

info!("Deleting iam role {}", self.role_name);
let delete_role = self
    .iam_client
    .delete_role()
    .role_name(self.role_name.clone())
    .send()
    .await
    .map_err(anyhow::Error::from);

let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
    if let Some(location) = location {
        info!("Deleting object {location}");
        Some(
            self.s3_client
                .delete_object()
                .bucket(self.bucket.clone())
                .key(location)
                .send()
                .await
                .map_err(anyhow::Error::from),
        )
    } else {
        info!(?location, "Skipping delete object");
        None
    };

(delete_function, delete_role, delete_object)
}

```

- Per i dettagli sulle API, consulta la [DeleteFunction](#) guida di riferimento all'API AWS SDK for Rust.



## SAP ABAP

### SDK per SAP ABAP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
TRY.  
    lo_lmd->deletefunction( iv_functionname = iv_function_name ).  
    MESSAGE 'Lambda function deleted.' TYPE 'I'.  
CATCH /aws1/cx_lmdinvparamvalueex.  
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.  
CATCH /aws1/cx_lmdresourceconflictex.  
    MESSAGE 'Resource already exists or another operation is in progress.'  
TYPE 'E'.  
CATCH /aws1/cx_lmdresourcenotfoundex.  
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.  
CATCH /aws1/cx_lmdserviceexception.  
    MESSAGE 'An internal problem was encountered by the AWS Lambda service.'  
TYPE 'E'.  
CATCH /aws1/cx_lmdtoomanyrequestsex.  
    MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.  
ENDTRY.
```

- Per i dettagli sulle API, [DeleteFunction](#) consulta AWS SDK for SAP ABAP API reference.

## Swift

### SDK per Swift

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
import AWSClientRuntime
import AWSLambda
import Foundation

do {
    _ = try await lambdaClient.deleteFunction(
        input: DeleteFunctionInput(
            functionName: "lambda-basics-function"
        )
    )
} catch {
    print("Error: Unable to delete the function.")
}
```

- Per i dettagli sull'API, consulta la [DeleteFunction](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **DeleteFunctionConcurrency** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare DeleteFunctionConcurrency.

### CLI

#### AWS CLI

Per rimuovere un limite di esecuzione simultanea riservata da una funzione

L'esempio delete-function-concurrency seguente elimina il limite di esecuzione simultanea riservato dalla funzione my-function.

```
aws lambda delete-function-concurrency \
    --function-name my-function
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta [Prenotazione della simultaneità per una funzione Lambda](#) nella Guida per gli sviluppatori di AWS Lambda.

- Per i dettagli sull'API, consulta [DeleteFunctionConcurrency AWS CLI](#) Command Reference.

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio rimuove la simultaneità della funzione dalla funzione Lambda.

```
Remove-LMFunctionConcurrency -FunctionName "MylambdaFunction123"
```

- Per i dettagli sull'API, vedere [DeleteFunctionConcurrency](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: questo esempio rimuove la simultaneità della funzione dalla funzione Lambda.

```
Remove-LMFunctionConcurrency -FunctionName "MylambdaFunction123"
```

- Per i dettagli sull'API, vedere [DeleteFunctionConcurrency](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **DeleteProvisionedConcurrencyConfig** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `DeleteProvisionedConcurrencyConfig`.

## CLI

### AWS CLI

Per eliminare una configurazione della simultaneità fornita

L'esempio `delete-provisioned-concurrency-config` seguente elimina la configurazione di simultaneità fornita per l'alias GREEN della funzione specificata.

```
aws lambda delete-provisioned-concurrency-config \  
  --function-name my-function \  
  --qualifier GREEN
```

- Per i dettagli sull'API, consulta [DeleteProvisionedConcurrencyConfig AWS CLI Command Reference](#).

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio rimuove la configurazione della simultaneità fornita per un alias specifico.

```
Remove-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -  
Qualifier "NewAlias1"
```

- Per i dettagli sull'API, vedere [DeleteProvisionedConcurrencyConfig](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: questo esempio rimuove la configurazione della simultaneità fornita per un alias specifico.

```
Remove-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -  
Qualifier "NewAlias1"
```

- Per i dettagli sull'API, vedere [DeleteProvisionedConcurrencyConfig](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **GetAccountSettings** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetAccountSettings`.

## CLI

### AWS CLI

Per recuperare i dettagli sul tuo account in una regione AWS

L'esempio `get-account-settings` seguente mostra i limiti Lambda e le informazioni sull'utilizzo per il tuo account.

```
aws lambda get-account-settings
```

Output:

```
{
  "AccountLimit": {
    "CodeSizeUnzipped": 262144000,
    "UnreservedConcurrentExecutions": 1000,
    "ConcurrentExecutions": 1000,
    "CodeSizeZipped": 52428800,
    "TotalCodeSize": 80530636800
  },
  "AccountUsage": {
    "FunctionCount": 4,
    "TotalCodeSize": 9426
  }
}
```

Per ulteriori informazioni, consulta [Limiti di AWS Lambda](#) nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta [GetAccountSettings AWS CLI](#) Command Reference.

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio viene visualizzato per confrontare il limite dell'account e l'utilizzo dell'account

```
Get-LMAccountSetting | Select-Object
@{Name="TotalCodeSizeLimit";Expression={$_.AccountLimit.TotalCodeSize}},
@{Name="TotalCodeSizeUsed";Expression={$_.AccountUsage.TotalCodeSize}}
```

Output:

```
TotalCodeSizeLimit TotalCodeSizeUsed
-----
      80530636800      15078795
```

- Per i dettagli sull'API, vedere [GetAccountSettings](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

## Strumenti per V5 PowerShell

Esempio 1: questo esempio viene visualizzato per confrontare il limite dell'account e l'utilizzo dell'account

```
Get-LMAccountSetting | Select-Object
@{Name="TotalCodeSizeLimit";Expression={$_.AccountLimit.TotalCodeSize}},
@{Name="TotalCodeSizeUsed";Expression={$_.AccountUsage.TotalCodeSize}}
```

Output:

```
TotalCodeSizeLimit TotalCodeSizeUsed
-----
      80530636800      15078795
```

- Per i dettagli sull'API, vedere [GetAccountSettings](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **GetAlias** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare GetAlias.

### CLI

#### AWS CLI

Per recuperare i dettagli dell'alias di una funzione

L'esempio `get-alias` seguente visualizza i dettagli per l'alias denominato `LIVE` dalla funzione Lambda `my-function`.

```
aws lambda get-alias \  
  --function-name my-function \  
  --name LIVE
```

Output:

```
{  
  "FunctionVersion": "3",  
  "Name": "LIVE",  
  "AliasArn": "arn:aws:lambda:us-west-2:123456789012:function:my-  
function:LIVE",  
  "RevisionId": "594f41fb-b85f-4c20-95c7-6ca5f2a92c93",  
  "Description": "alias for live version of function"  
}
```

Per ulteriori informazioni, consulta [Configurazione degli alias delle funzioni AWS Lambda](#) nella AWS Lambda Developer Guide.

- Per i dettagli sull'API, consulta [GetAlias](#) Command Reference.AWS CLI

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio recupera i pesi di configurazione del routing per uno specifico alias della funzione Lambda.

```
Get-LMAlias -FunctionName "MylambdaFunction123" -Name "newlabel1" -Select  
RoutingConfig
```

Output:

```
AdditionalVersionWeights  
-----  
{[1, 0.6]}
```

- Per i dettagli sull'API, vedere [GetAlias](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

## Strumenti per V5 PowerShell

Esempio 1: questo esempio recupera i pesi di configurazione del routing per uno specifico alias della funzione Lambda.

```
Get-LMAlias -FunctionName "MyLambdaFunction123" -Name "newlabel1" -Select  
RoutingConfig
```

Output:

```
AdditionalVersionWeights  
-----  
{[1, 0.6]}
```

- Per i dettagli sull'API, vedere [GetAlias](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **GetFunction** con un AWS SDK o CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetFunction`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- [Informazioni di base](#)
- [Creazione della tua prima funzione serverless](#)

## .NET

### SDK per .NET

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).



```
/// <summary>
/// Gets information about a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function for
/// which to retrieve information.</param>
/// <returns>Async Task.</returns>
public async Task<FunctionConfiguration> GetFunctionAsync(string
functionName)
{
    var functionRequest = new GetFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.GetFunctionAsync(functionRequest);
    return response.Configuration;
}
```

- Per i dettagli sull'API, consulta la [GetFunction](#) sezione AWS SDK per .NET API Reference.

## C++

### SDK per C++

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
// (overrides config file).
// clientConfig.region = "us-east-1";

Aws::Lambda::LambdaClient client(clientConfig);

Aws::Lambda::Model::GetFunctionRequest request;
request.SetFunctionName(functionName);
```

```
Aws::Lambda::Model::GetFunctionOutcome outcome =
client.GetFunction(request);

if (outcome.IsSuccess()) {
    std::cout << "Function retrieve.\n" <<

outcome.GetResult().GetConfiguration().Jsonize().View().WriteReadable()
    << std::endl;
}
else {
    std::cerr << "Error with Lambda::GetFunction. "
    << outcome.GetError().GetMessage()
    << std::endl;
}
```

- Per i dettagli sull'API, consulta la [GetFunction](#) sezione AWS SDK per C++ API Reference.

## CLI

### AWS CLI

Come recuperare le informazioni relative a una funzione

Nell'esempio di `get-function` seguente vengono visualizzate informazioni sulla funzione `my-function`.

```
aws lambda get-function \
    --function-name my-function
```

Output:

```
{
  "Concurrency": {
    "ReservedConcurrentExecutions": 100
  },
  "Code": {
    "RepositoryType": "S3",
    "Location": "https://awslambda-us-west-2-tasks.s3.us-
west-2.amazonaws.com/snapshots/123456789012/my-function..."
  },
```

```
"Configuration": {
  "TracingConfig": {
    "Mode": "PassThrough"
  },
  "Version": "$LATEST",
  "CodeSha256": "5tT2qgzYUHoqwR616pZ2dpkn/0J1FrzJmLKidWaaCgk=",
  "FunctionName": "my-function",
  "VpcConfig": {
    "SubnetIds": [],
    "VpcId": "",
    "SecurityGroupIds": []
  },
  "MemorySize": 128,
  "RevisionId": "28f0fb31-5c5c-43d3-8955-03e76c5c1075",
  "CodeSize": 304,
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
  "Handler": "index.handler",
  "Role": "arn:aws:iam::123456789012:role/service-role/helloWorldPython-role-uy3l9qyq",
  "Timeout": 3,
  "LastModified": "2025-09-24T18:20:35.054+0000",
  "Runtime": "nodejs22.x",
  "Description": ""
}
```

Per ulteriori informazioni, consulta [Configurazione della memoria della funzione Lambda](#) nella Guida per gli sviluppatori Lambda AWS .

- Per i dettagli sull'API, consulta [GetFunction AWS CLI](#) Command Reference.

Go

SDK per Go V2

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
import (
    "bytes"
    "context"
    "encoding/json"
    "errors"
    "log"
    "time"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/lambda"
    "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
    LambdaClient *lambda.Client
}

// GetFunction gets data about the Lambda function specified by functionName.
func (wrapper FunctionWrapper) GetFunction(ctx context.Context, functionName
    string) types.State {
    var state types.State
    funcOutput, err := wrapper.LambdaClient.GetFunction(ctx,
        &lambda.GetFunctionInput{
            FunctionName: aws.String(functionName),
        })
    if err != nil {
        log.Panicf("Couldn't get function %v. Here's why: %v\n", functionName, err)
    } else {
        state = funcOutput.Configuration.State
    }
    return state
}
```

- Per i dettagli sull'API, consulta la [GetFunction](#) sezione AWS SDK per Go API Reference.

## Java

### SDK per Java 2.x

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/**
 * Retrieves information about an AWS Lambda function.
 *
 * @param awsLambda an instance of the {@link LambdaClient} class, which
is used to interact with the AWS Lambda service
 * @param functionName the name of the AWS Lambda function to retrieve
information about
 */
public static void getFunction(LambdaClient awsLambda, String functionName) {
    try {
        GetFunctionRequest functionRequest = GetFunctionRequest.builder()
            .functionName(functionName)
            .build();

        GetFunctionResponse response =
awsLambda.getFunction(functionRequest);
        System.out.println("The runtime of this Lambda function is " +
response.configuration().runtime());

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- Per i dettagli sull'API, consulta la [GetFunction](#) sezione AWS SDK for Java 2.x API Reference.

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
const getFunction = (funcName) => {  
  const client = new LambdaClient({});  
  const command = new GetFunctionCommand({ FunctionName: funcName });  
  return client.send(command);  
};
```

- Per i dettagli sull'API, consulta la [GetFunction](#) sezione AWS SDK per JavaScript API Reference.

## PHP

### SDK per PHP

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
public function getFunction($functionName)  
{  
    return $this->lambdaClient->getFunction([  
        'FunctionName' => $functionName,  
    ]);  
}
```

- Per i dettagli sull'API, consulta la [GetFunction](#) sezione AWS SDK per PHP API Reference.

## Python

### SDK per Python (Boto3)

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def get_function(self, function_name):
        """
        Gets data about a Lambda function.

        :param function_name: The name of the function.
        :return: The function data.
        """
        response = None
        try:
            response =
self.lambda_client.get_function(FunctionName=function_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Function %s does not exist.", function_name)
            else:
                logger.error(
                    "Couldn't get function %s. Here's why: %s: %s",
                    function_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return response
```

- Per i dettagli sull'API, consulta [GetFunction](#) AWS SDK for Python (Boto3) API Reference.

## Ruby

### SDK per Ruby

#### Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Gets data about a Lambda function.
  #
  # @param function_name: The name of the function.
  # @return response: The function data, or nil if no such function exists.
  def get_function(function_name)
    @lambda_client.get_function(
      {
        function_name: function_name
      }
    )
  rescue Aws::Lambda::Errors::ResourceNotFoundException => e
    @logger.debug("Could not find function: #{function_name}: \n #{e.message}")
    nil
  end
end
```

- Per i dettagli sull'API, consulta la [GetFunction](#) sezione AWS SDK per Ruby API Reference.



## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/** Get the Lambda function with this Manager's name. */
pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error>
{
    info!("Getting lambda function");
    self.lambda_client
        .get_function()
        .function_name(self.lambda_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from)
}
```

- Per i dettagli sulle API, consulta la [GetFunction](#) guida di riferimento all'API AWS SDK for Rust.

## SAP ABAP

### SDK per SAP ABAP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
TRY.
    oo_result = lo_lmd->getfunction( iv_functionname = iv_function_name ).
    " oo_result is returned for testing purposes. "
    MESSAGE 'Lambda function information retrieved.' TYPE 'I'.
```

```

CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_lmdserviceexception.
    MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
CATCH /aws1/cx_lmdtoomanyrequestsex.
    MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
ENDTRY.

```

- Per i dettagli sulle API, [GetFunction](#) consulta AWS SDK for SAP ABAP API reference.

## Swift

### SDK per Swift

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

import AWSClientRuntime
import AWSLambda
import Foundation

/// Detect whether or not the AWS Lambda function with the specified name
/// exists, by requesting its function information.
///
/// - Parameters:
///   - lambdaClient: The `LambdaClient` to use.
///   - name: The name of the AWS Lambda function to find.
///
/// - Returns: `true` if the Lambda function exists. Otherwise `false`.
func doesLambdaFunctionExist(lambdaClient: LambdaClient, name: String) async
-> Bool {
    do {
        _ = try await lambdaClient.getFunction(
            input: GetFunctionInput(functionName: name)
        )
    } catch {
        return false
    }
}

```

```
    }  
  
    return true  
}
```

- Per i dettagli sull'API, consulta la [GetFunction](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **GetFunctionConcurrency** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare GetFunctionConcurrency.

### CLI

#### AWS CLI

Per visualizzare l'impostazione di simultaneità riservata per una funzione

L'esempio `get-function-concurrency` seguente recupera l'impostazione della simultaneità riservata per la funzione specificata.

```
aws lambda get-function-concurrency \  
  --function-name my-function
```

Output:

```
{  
  "ReservedConcurrentExecutions": 250  
}
```

- Per i dettagli sull'API, consulta [GetFunctionConcurrency AWS CLI](#) Command Reference.

### PowerShell

#### Strumenti per PowerShell V4

Esempio 1: questo esempio ottiene la simultaneità riservata per la funzione Lambda

```
Get-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -Select *
```

Output:

```
ReservedConcurrentExecutions
-----
100
```

- Per i dettagli sull'API, vedere [GetFunctionConcurrency](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio ottiene la simultaneità riservata per la funzione Lambda

```
Get-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -Select *
```

Output:

```
ReservedConcurrentExecutions
-----
100
```

- Per i dettagli sull'API, vedere [GetFunctionConcurrency](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **GetFunctionConfiguration** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare GetFunctionConfiguration.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- [Creazione della tua prima funzione serverless](#)

## CLI

### AWS CLI

Come recuperare le impostazioni specifiche della versione di una funzione Lambda

L'esempio `get-function-configuration` seguente visualizza le impostazioni per la versione 2 della funzione `my-function`.

```
aws lambda get-function-configuration \  
  --function-name my-function:2
```

Output:

```
{  
  "FunctionName": "my-function",  
  "LastModified": "2019-09-26T20:28:40.438+0000",  
  "RevisionId": "e52502d4-9320-4688-9cd6-152a6ab7490d",  
  "MemorySize": 256,  
  "Version": "2",  
  "Role": "arn:aws:iam::123456789012:role/service-role/my-function-role-uy3l9qq",  
  "Timeout": 3,  
  "Runtime": "nodejs10.x",  
  "TracingConfig": {  
    "Mode": "PassThrough"  
  },  
  "CodeSha256": "5tT2qgzYUHaqwR716pZ2dpkn/0J1FrzJmLKidWoaCgk=",  
  "Description": "",  
  "VpcConfig": {  
    "SubnetIds": [],  
    "VpcId": "",  
    "SecurityGroupIds": []  
  },  
  "CodeSize": 304,  
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function:2",  
  "Handler": "index.handler"  
}
```

Per ulteriori informazioni, consulta [Configurazione delle funzioni Lambda AWS](#) nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta [GetFunctionConfiguration AWS CLI Command Reference](#).

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio recupera la configurazione specifica della versione di una funzione Lambda.

```
Get-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Qualifier
"PowershellAlias"
```

Output:

```
CodeSha256           : uW0W0R7z+f0VyLuUg7+/D08hkMFsq0SF4seuyUZJ/R8=
CodeSize             : 1426
DeadLetterConfig     : Amazon.Lambda.Model.DeadLetterConfig
Description          : Verson 3 to test Aliases
Environment          : Amazon.Lambda.Model.EnvironmentResponse
FunctionArn          : arn:aws:lambda:us-
east-1:123456789012:function:MylambdaFunction123
                     :PowershellAlias
FunctionName         : MylambdaFunction123
Handler              : lambda_function.launch_instance
KMSKeyArn            :
LastModified         : 2019-12-25T09:52:59.872+0000
LastUpdateStatus     : Successful
LastUpdateStatusReason :
LastUpdateStatusReasonCode :
Layers               : {}
MasterArn            :
MemorySize           : 128
RevisionId           : 5d7de38b-87f2-4260-8f8a-e87280e10c33
Role                 : arn:aws:iam::123456789012:role/service-role/lambda
Runtime              : python3.8
State                : Active
StateReason          :
StateReasonCode      :
Timeout              : 600
TracingConfig        : Amazon.Lambda.Model.TracingConfigResponse
Version              : 4
VpcConfig            : Amazon.Lambda.Model.VpcConfigDetail
```

- Per i dettagli sull'API, vedere [GetFunctionConfiguration](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

## Strumenti per V5 PowerShell

Esempio 1: questo esempio recupera la configurazione specifica della versione di una funzione Lambda.

```
Get-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Qualifier
"PowershellAlias"
```

### Output:

```
CodeSha256           : uW0W0R7z+f0VyLuUg7+/D08hkMFsq0SF4seuyUZJ/R8=
CodeSize             : 1426
DeadLetterConfig     : Amazon.Lambda.Model.DeadLetterConfig
Description          : Version 3 to test Aliases
Environment          : Amazon.Lambda.Model.EnvironmentResponse
FunctionArn          : arn:aws:lambda:us-
east-1:123456789012:function:MylambdaFunction123
                    :PowershellAlias
FunctionName         : MylambdaFunction123
Handler              : lambda_function.launch_instance
KMSKeyArn            :
LastModified         : 2019-12-25T09:52:59.872+0000
LastUpdateStatus     : Successful
LastUpdateStatusReason :
LastUpdateStatusReasonCode :
Layers               : {}
MasterArn            :
MemorySize           : 128
RevisionId           : 5d7de38b-87f2-4260-8f8a-e87280e10c33
Role                 : arn:aws:iam::123456789012:role/service-role/lambda
Runtime              : python3.8
State                : Active
StateReason          :
StateReasonCode       :
Timeout              : 600
TracingConfig        : Amazon.Lambda.Model.TracingConfigResponse
Version              : 4
VpcConfig             : Amazon.Lambda.Model.VpcConfigDetail
```

- Per i dettagli sull'API, vedere [GetFunctionConfiguration](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **GetPolicy** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare GetPolicy.

### CLI

#### AWS CLI

Per recuperare la policy IAM basata sulle risorse per una funzione, una versione o un alias

L'esempio get-policy seguente mostra le informazioni sulla policy sulla funzione Lambda my-function.

```
aws lambda get-policy \  
  --function-name my-function
```

Output:

```
{  
  "Policy": {  
    "Version": "2012-10-17",  
    "Id": "default",  
    "Statement": [  
      {  
        "Sid": "iot-events",  
        "Effect": "Allow",  
        "Principal": {"Service": "iotevents.amazonaws.com"},  
        "Action": "lambda:InvokeFunction",  
        "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-  
function"  
      }  
    ],  
  },  
}
```



```
"RevisionId": "93017fc9-59cb-41dc-901b-4845ce4bf668"
}
```

Per ulteriori informazioni, consulta [Using Resource-based Policies for AWS Lambda nella Lambda Developer Guide AWS](#).

- Per i dettagli sull'API, consulta AWS CLI Command [GetPolicy](#) Reference.

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio mostra la policy delle funzioni della funzione Lambda

```
Get-LMPolicy -FunctionName test -Select Policy
```

Output:

```
{"Version":"2012-10-17",      "Id":"default","Statement":
[{"Sid":"xxxx","Effect":"Allow","Principal":
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-east-1:123456789102:function:test"]}]}
```

- Per i dettagli sull'API, vedere [GetPolicy](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: questo esempio mostra la policy delle funzioni della funzione Lambda

```
Get-LMPolicy -FunctionName test -Select Policy
```

Output:

```
{"Version":"2012-10-17",      "Id":"default","Statement":
[{"Sid":"xxxx","Effect":"Allow","Principal":
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-east-1:123456789102:function:test"]}]}
```

- Per i dettagli sull'API, vedere [GetPolicy](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **GetProvisionedConcurrencyConfig** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetProvisionedConcurrencyConfig`.

### CLI

#### AWS CLI

Per visualizzare una configurazione della simultaneità fornita

L'esempio `get-provisioned-concurrency-config` seguente visualizza i dettagli della configurazione di simultaneità fornita per l'alias `BLUE` della funzione specificata.

```
aws lambda get-provisioned-concurrency-config \
  --function-name my-function \
  --qualifier BLUE
```

Output:

```
{
  "RequestedProvisionedConcurrentExecutions": 100,
  "AvailableProvisionedConcurrentExecutions": 100,
  "AllocatedProvisionedConcurrentExecutions": 100,
  "Status": "READY",
  "LastModified": "2019-12-31T20:28:49+0000"
}
```

- Per i dettagli sull'API, consulta [GetProvisionedConcurrencyConfig AWS CLI Command Reference](#).

### PowerShell

#### Strumenti per PowerShell V4

Esempio 1: questo esempio ottiene la configurazione di simultaneità fornita per l'alias specificato della funzione Lambda.

```
C:\>Get-LMProvisionedConcurrencyConfig -FunctionName "MyLambdaFunction123" -
Qualifier "NewAlias1"
```

Output:

```
AllocatedProvisionedConcurrentExecutions : 0
AvailableProvisionedConcurrentExecutions : 0
LastModified                             : 2020-01-15T03:21:26+0000
RequestedProvisionedConcurrentExecutions : 70
Status                                    : IN_PROGRESS
StatusReason                             :
```

- Per i dettagli sull'API, vedere [GetProvisionedConcurrencyConfig](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio ottiene la configurazione di simultaneità fornita per l'alias specificato della funzione Lambda.

```
C:\>Get-LMProvisionedConcurrencyConfig -FunctionName "MyLambdaFunction123" -
Qualifier "NewAlias1"
```

Output:

```
AllocatedProvisionedConcurrentExecutions : 0
AvailableProvisionedConcurrentExecutions : 0
LastModified                             : 2020-01-15T03:21:26+0000
RequestedProvisionedConcurrentExecutions : 70
Status                                    : IN_PROGRESS
StatusReason                             :
```

- Per i dettagli sull'API, vedere [GetProvisionedConcurrencyConfig](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Usa **Invoke** con un AWS SDK o CLI

Gli esempi di codice seguenti mostrano come utilizzare Invoke.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- [Informazioni di base](#)
- [Creazione di una dashboard di monitoraggio con il nome della funzione come variabile](#)
- [Creazione della tua prima funzione serverless](#)
- [Utilizzo delle variabili di proprietà nei dashboard di monitoraggio per monitorare più funzioni serverless](#)

### .NET

#### SDK per .NET

##### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Invoke a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// invoke.</param>
/// <param name="parameters">The parameter values that will be passed to the
function.</param>
/// <returns>A System Threading Task.</returns>
public async Task<string> InvokeFunctionAsync(
    string functionName,
    string parameters)
{
    var payload = parameters;
    var request = new InvokeRequest
    {
        FunctionName = functionName,
        Payload = payload,
```

```
};

var response = await _lambdaService.InvokeAsync(request);
MemoryStream stream = response.Payload;
string returnValue =
System.Text.Encoding.UTF8.GetString(stream.ToArray());
return returnValue;
}
```

- Per informazioni dettagliate sull'API, consulta [Invoke](#) nella documentazione di riferimento dell'API AWS SDK per .NET .

## C++

### SDK per C++

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Lambda::LambdaClient client(clientConfig);

Aws::Lambda::Model::InvokeRequest request;
request.SetFunctionName(LAMBDA_NAME);
request.SetLogType(logType);
std::shared_ptr<Aws::IOStream> payload =
Aws::MakeShared<Aws::StringStream>(
    "FunctionTest");
*payload << jsonPayload.View().WriteReadable();
request.SetBody(payload);
request.SetContentType("application/json");
Aws::Lambda::Model::InvokeOutcome outcome = client.Invoke(request);
```

```
if (outcome.IsSuccess()) {
    invokeResult = std::move(outcome.GetResult());
    result = true;
    break;
}

else {
    std::cerr << "Error with Lambda::InvokeRequest. "
               << outcome.GetError().GetMessage()
               << std::endl;
    break;
}
```

- Per informazioni dettagliate sull'API, consulta [Invoke](#) nella documentazione di riferimento dell'API AWS SDK per C++ .

## CLI

### AWS CLI

Esempio 1: come invocare una funzione Lambda in modo sincrono

L'esempio di `invoke` seguente invoca la funzione `my-function` in modo sincrono. L'opzione `cli-binary-format` è obbligatoria se utilizzi la versione 2 della AWS CLI. Per ulteriori informazioni, consulta [Opzioni della riga di comando globali supportate da AWS CLI](#) nella Guida per l'utente dell'Interfaccia della linea di comando AWS (AWS CLI).

```
aws lambda invoke \
    --function-name my-function \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "name": "Bob" }' \
    response.json
```

Output:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

Per ulteriori informazioni, consulta [Invocare una funzione Lambda in modo sincrono](#) nella Guida per gli sviluppatori di AWS Lambda.

Esempio 2: come invocare una funzione Lambda in modo asincrono

L'esempio di `invoke` seguente invoca la funzione `my-function` in modo asincrono. L'opzione `cli-binary-format` è obbligatoria se utilizzi la versione 2 della AWS CLI. Per ulteriori informazioni, consulta [Opzioni della riga di comando globali supportate da AWS CLI](#) nella Guida per l'utente dell'Interfaccia della linea di comando AWS (AWS CLI).

```
aws lambda invoke \
  --function-name my-function \
  --invocation-type Event \
  --cli-binary-format raw-in-base64-out \
  --payload '{ "name": "Bob" }' \
  response.json
```

Output:

```
{
  "StatusCode": 202
}
```

Per ulteriori informazioni, consulta [Invocare una funzione Lambda in modo asincrono](#) nella Guida per gli sviluppatori di AWS Lambda.

- Per informazioni dettagliate sull'API, consulta [Invoke](#) nella documentazione di riferimento dei comandi della AWS CLI .

Go

SDK per Go V2

#### Note

C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
import (
    "bytes"
    "context"
    "encoding/json"
    "errors"
    "log"
    "time"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/lambda"
    "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
    LambdaClient *lambda.Client
}

// Invoke invokes the Lambda function specified by functionName, passing the
// parameters
// as a JSON payload. When getLog is true, types.LogTypeTail is specified, which
// tells
// Lambda to include the last few log lines in the returned result.
func (wrapper FunctionWrapper) Invoke(ctx context.Context, functionName string,
    parameters any, getLog bool) *lambda.InvokeOutput {
    logType := types.LogTypeNone
    if getLog {
        logType = types.LogTypeTail
    }
    payload, err := json.Marshal(parameters)
    if err != nil {
        log.Panicf("Couldn't marshal parameters to JSON. Here's why %v\n", err)
    }
    invokeOutput, err := wrapper.LambdaClient.Invoke(ctx, &lambda.InvokeInput{
        FunctionName: aws.String(functionName),
        LogType:      logType,
        Payload:      payload,
    })
    if err != nil {
        log.Panicf("Couldn't invoke function %v. Here's why: %v\n", functionName, err)
    }
}
```



```
    return invokeOutput
}
```

- Per informazioni dettagliate sull'API, consulta [Invoke](#) nella documentazione di riferimento dell'API AWS SDK per Go .

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/**
 * Invokes a specific AWS Lambda function.
 *
 * @param awsLambda    an instance of {@link LambdaClient} to interact with
the AWS Lambda service
 * @param functionName the name of the AWS Lambda function to be invoked
 */
public static void invokeFunction(LambdaClient awsLambda, String
functionName) {
    InvokeResponse res;
    try {
        // Need a SdkBytes instance for the payload.
        JSONObject jsonObj = new JSONObject();
        jsonObj.put("inputValue", "2000");
        String json = jsonObj.toString();
        SdkBytes payload = SdkBytes.fromUtf8String(json);

        InvokeRequest request = InvokeRequest.builder()
            .functionName(functionName)
            .payload(payload)
            .build();

        res = awsLambda.invoke(request);
    }
```

```
        String value = res.payload().asUtf8String();
        System.out.println(value);

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- Per informazioni dettagliate sull'API, consulta [Invoke](#) nella documentazione di riferimento dell'API AWS SDK for Java 2.x .

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
const invoke = async (funcName, payload) => {
    const client = new LambdaClient({});
    const command = new InvokeCommand({
        FunctionName: funcName,
        Payload: JSON.stringify(payload),
        LogType: LogType.Tail,
    });

    const { Payload, LogResult } = await client.send(command);
    const result = Buffer.from(Payload).toString();
    const logs = Buffer.from(LogResult, "base64").toString();
    return { logs, result };
};
```

- Per informazioni dettagliate sull'API, consulta [Invoke](#) nella documentazione di riferimento dell'API AWS SDK per JavaScript .

## Kotlin

### SDK per Kotlin

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
suspend fun invokeFunction(functionNameVal: String) {  
    val json = """"{"inputValue":"1000"}""""  
    val byteArray = json.trimIndent().encodeToByteArray()  
    val request =  
        InvokeRequest {  
            functionName = functionNameVal  
            logType = LogType.Tail  
            payload = byteArray  
        }  
  
    LambdaClient { region = "us-west-2" }.use { awsLambda ->  
        val res = awsLambda.invoke(request)  
        println("${res.payload?.toString(Charsets.UTF_8)}")  
        println("The log result is ${res.logResult}")  
    }  
}
```

- Per informazioni dettagliate sull'API, consulta [Invoke](#) nella documentazione di riferimento dell'API AWS SDK per Kotlin.

## PHP

### SDK per PHP

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

public function invoke($functionName, $params, $logType = 'None')
{
    return $this->lambdaClient->invoke([
        'FunctionName' => $functionName,
        'Payload' => json_encode($params),
        'LogType' => $logType,
    ]);
}

```

- Per informazioni dettagliate sull'API, consulta [Invoke](#) nella documentazione di riferimento dell'API AWS SDK per PHP .

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def invoke_function(self, function_name, function_params, get_log=False):
        """
        Invokes a Lambda function.

        :param function_name: The name of the function to invoke.
        :param function_params: The parameters of the function as a dict. This
        dict
                               is serialized to JSON before it is sent to
        Lambda.
        :param get_log: When true, the last 4 KB of the execution log are
        included in
                       the response.

```

```

        :return: The response from the function invocation.
        """
        try:
            response = self.lambda_client.invoke(
                FunctionName=function_name,
                Payload=json.dumps(function_params),
                LogType="Tail" if get_log else "None",
            )
            logger.info("Invoked function %s.", function_name)
        except ClientError:
            logger.exception("Couldn't invoke function %s.", function_name)
            raise
        return response

```

- Per informazioni dettagliate sull'API, consulta [Invoke](#) nella documentazione di riferimento dell'API AWS SDK per Python (Boto3).

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Invokes a Lambda function.

```

```
# @param function_name [String] The name of the function to invoke.
# @param payload [nil] Payload containing runtime parameters.
# @return [Object] The response from the function invocation.
def invoke_function(function_name, payload = nil)
  params = { function_name: function_name }
  params[:payload] = payload unless payload.nil?
  @lambda_client.invoke(params)
rescue Aws::Lambda::Errors::ServiceException => e
  @logger.error("There was an error executing #{function_name}:\n
#{e.message}")
end
```

- Per informazioni dettagliate sull'API, consulta [Invoke](#) nella documentazione di riferimento dell'API AWS SDK per Ruby .

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/** Invoke the lambda function using calculator InvokeArgs. */
pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
  info!(?args, "Invoking {}", self.lambda_name);
  let payload = serde_json::to_string(&args)?;
  debug!(?payload, "Sending payload");
  self.lambda_client
    .invoke()
    .function_name(self.lambda_name.clone())
    .payload(Blob::new(payload))
    .send()
    .await
    .map_err(anyhow::Error::from)
}
```

```
fn log_invoke_output(invoke: &InvokeOutput, message: &str) {
    if let Some(payload) = invoke.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
        info!("Could not extract payload")
    }
    if let Some(logs) = invoke.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}
```

- Per informazioni dettagliate sull'API, consulta la pagina [Invoke](#) nella documentazione di riferimento dell'API SDK AWS per Rust.

## SAP ABAP

### SDK per SAP ABAP

#### Note

C'è altro da fare. [GitHub Trova l'esempio completo](#) e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
TRY.
    DATA(lv_json) = /aws1/cl_rt_util=>string_to_xstring(
        `{` &&
        `"action": "increment",` &&
        `"number": 10` &&
        `}` ).
    oo_result = lo_lmd->invoke(                                " oo_result is returned for
testing purposes. "
        iv_functionname = iv_function_name
        iv_payload = lv_json ).
    MESSAGE 'Lambda function invoked.' TYPE 'I'.
CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
```

```

CATCH /aws1/cx_lmdinvrequestctx.
    MESSAGE 'Unable to parse request body as JSON.' TYPE 'E'.
CATCH /aws1/cx_lmdinvalidzipfileex.
    MESSAGE 'The deployment package could not be unzipped.' TYPE 'E'.
CATCH /aws1/cx_lmdrequesttoolargeex.
    MESSAGE 'Invoke request body JSON input limit was exceeded by the request
payload.' TYPE 'E'.
CATCH /aws1/cx_lmdresourceconflictex.
    MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
CATCH /aws1/cx_lmdresourcenotfoundex.
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.
CATCH /aws1/cx_lmdserviceexception.
    MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
CATCH /aws1/cx_lmdtoomanyrequestsex.
    MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
CATCH /aws1/cx_lmdunsuppedmediatyp00.
    MESSAGE 'Invoke request body does not have JSON as its content type.'
TYPE 'E'.
ENDTRY.

```

- Per informazioni dettagliate sull'API, consulta [Invoke](#) nella documentazione di riferimento dell'API AWS SDK per SAP ABAP.

## Swift

### SDK per Swift

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

import AWSClientRuntime
import AWSLambda
import Foundation

/// Invoke the Lambda function to increment a value.

```



```

///
/// - Parameters:
///   - lambdaClient: The `IAMClient` to use.
///   - number: The number to increment.
///
/// - Throws: `ExampleError.noAnswerReceived`, `ExampleError.invokeError`
///
/// - Returns: An integer number containing the incremented value.
func invokeIncrement(lambdaClient: LambdaClient, number: Int) async throws ->
Int {
    do {
        let incRequest = IncrementRequest(action: "increment", number:
number)

        let incData = try! JSONEncoder().encode(incRequest)

        // Invoke the lambda function.

        let invokeOutput = try await lambdaClient.invoke(
            input: InvokeInput(
                functionName: "lambda-basics-function",
                payload: incData
            )
        )

        let response = try! JSONDecoder().decode(Response.self,
from:invokeOutput.payload!)

        guard let answer = response.answer else {
            throw ExampleError.noAnswerReceived
        }
        return answer

    } catch {
        throw ExampleError.invokeError
    }
}

```

- Per informazioni dettagliate sull'API, consulta [Invoke](#) nella documentazione di riferimento dell'API AWS SDK per Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **ListFunctions** con un AWS SDK o CLI

Gli esempi di codice seguenti mostrano come utilizzare `ListFunctions`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nei seguenti esempi di codice:

- [Informazioni di base](#)
- [Creazione di una dashboard di monitoraggio con il nome della funzione come variabile](#)

### .NET

#### SDK per .NET

##### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Get a list of Lambda functions.
/// </summary>
/// <returns>A list of FunctionConfiguration objects.</returns>
public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
{
    var functionList = new List<FunctionConfiguration>();

    var functionPaginator =
        _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
    await foreach (var function in functionPaginator.Functions)
    {
        functionList.Add(function);
    }

    return functionList;
}
```

- Per i dettagli sull'API, consulta la [ListFunctions](#) sezione AWS SDK per .NET API Reference.

## C++

### SDK per C++

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Lambda::LambdaClient client(clientConfig);

std::vector<Aws::String> functions;
Aws::String marker;

do {
    Aws::Lambda::Model::ListFunctionsRequest request;
    if (!marker.empty()) {
        request.SetMarker(marker);
    }

    Aws::Lambda::Model::ListFunctionsOutcome outcome = client.ListFunctions(
        request);

    if (outcome.IsSuccess()) {
        const Aws::Lambda::Model::ListFunctionsResult &result =
outcome.GetResult();
        std::cout << result.GetFunctions().size()
            << " lambda functions were retrieved." << std::endl;

        for (const Aws::Lambda::Model::FunctionConfiguration
&functionConfiguration: result.GetFunctions()) {
```

```

        functions.push_back(functionConfiguration.GetFunctionName());
        std::cout << functions.size() << " "
                  << functionConfiguration.GetDescription() << std::endl;
        std::cout << " "
                  <<
        Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
            functionConfiguration.GetRuntime()) << ": "
                  << functionConfiguration.GetHandler()
                  << std::endl;
    }
    marker = result.GetNextMarker();
}
else {
    std::cerr << "Error with Lambda::ListFunctions. "
              << outcome.GetError().GetMessage()
              << std::endl;
}
} while (!marker.empty());

```

- Per i dettagli sull'API, consulta la [ListFunctions](#) sezione AWS SDK per C++ API Reference.

## CLI

### AWS CLI

Come recuperare un elenco di funzioni Lambda

L'esempio di `list-functions` seguente visualizza un elenco di tutte le funzioni per l'utente attuale.

```
aws lambda list-functions
```

Output:

```

{
  "Functions": [
    {
      "TracingConfig": {
        "Mode": "PassThrough"
      },
      "Version": "$LATEST",

```

```

        "CodeSha256": "dBG9m8SGdmlEjw/JYXlhhvCrAv5TxvXsbl/RMr0fT/I=",
        "FunctionName": "helloworld",
        "MemorySize": 128,
        "RevisionId": "1718e831-badf-4253-9518-d0644210af7b",
        "CodeSize": 294,
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:helloworld",
        "Handler": "helloworld.handler",
        "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-zgur6bf4",
        "Timeout": 3,
        "LastModified": "2025-09-23T18:32:33.857+0000",
        "Runtime": "nodejs22.x",
        "Description": ""
    },
    {
        "TracingConfig": {
            "Mode": "PassThrough"
        },
        "Version": "$LATEST",
        "CodeSha256": "sU0cJ2/h0ZevwV/1TxCuQqK3gDZP3i8gUoqUUVRmY6E=",
        "FunctionName": "my-function",
        "VpcConfig": {
            "SubnetIds": [],
            "VpcId": "",
            "SecurityGroupIds": []
        },
        "MemorySize": 256,
        "RevisionId": "93017fc9-59cb-41dc-901b-4845ce4bf668",
        "CodeSize": 266,
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
        "Handler": "index.handler",
        "Role": "arn:aws:iam::123456789012:role/service-role/helloWorldPython-role-uy3l9qq",
        "Timeout": 3,
        "LastModified": "2025-10-01T16:47:28.490+0000",
        "Runtime": "nodejs22.x",
        "Description": ""
    },
    {
        "Layers": [
            {
                "CodeSize": 41784542,

```

```

        "Arn": "arn:aws:lambda:us-
west-2:420165488524:layer:AWSLambda-Python37-SciPy1x:2"
    },
    {
        "CodeSize": 4121,
        "Arn": "arn:aws:lambda:us-
west-2:123456789012:layer:pythonLayer:1"
    }
],
"TracingConfig": {
    "Mode": "PassThrough"
},
"Version": "$LATEST",
"CodeSha256": "ZQukCqxtkqFgyF2cU41Avj99TKQ/hNihPtDtRcc08mI=",
"FunctionName": "my-python-function",
"VpcConfig": {
    "SubnetIds": [],
    "VpcId": "",
    "SecurityGroupIds": []
},
"MemorySize": 128,
"RevisionId": "80b4eabc-acf7-4ea8-919a-e874c213707d",
"CodeSize": 299,
"FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
python-function",
"Handler": "lambda_function.lambda_handler",
"Role": "arn:aws:iam::123456789012:role/service-role/my-python-
function-role-z5g7dr6n",
"Timeout": 3,
"LastModified": "2025-10-01T19:40:41.643+0000",
"Runtime": "python3.11",
"Description": ""
}
]
}

```

Per ulteriori informazioni, consulta [Configurazione della memoria della funzione Lambda](#) nella Guida per gli sviluppatori Lambda AWS .

- Per i dettagli sull'API, consulta [ListFunctions AWS CLI](#) Command Reference.

## Go

## SDK per Go V2

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
import (
    "bytes"
    "context"
    "encoding/json"
    "errors"
    "log"
    "time"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/lambda"
    "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
    LambdaClient *lambda.Client
}

// ListFunctions lists up to maxItems functions for the account. This function
// uses a
// lambda.ListFunctionsPaginator to paginate the results.
func (wrapper FunctionWrapper) ListFunctions(ctx context.Context, maxItems int)
    []types.FunctionConfiguration {
    var functions []types.FunctionConfiguration
    paginator := lambda.NewListFunctionsPaginator(wrapper.LambdaClient,
        &lambda.ListFunctionsInput{
            MaxItems: aws.Int32(int32(maxItems)),
        })
}
```

```
for paginator.HasMorePages() && len(functions) < maxItems {
    pageOutput, err := paginator.NextPage(ctx)
    if err != nil {
        log.Panicf("Couldn't list functions for your account. Here's why: %v\n", err)
    }
    functions = append(functions, pageOutput.Functions...)
}
return functions
}
```

- Per i dettagli sull'API, consulta la [ListFunctions](#) sezione AWS SDK per Go API Reference.

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
const listFunctions = () => {
    const client = new LambdaClient({});
    const command = new ListFunctionsCommand({});

    return client.send(command);
};
```

- Per i dettagli sull'API, consulta la [ListFunctions](#) sezione AWS SDK per JavaScript API Reference.



## PHP

### SDK per PHP

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
public function listFunctions($maxItems = 50, $marker = null)
{
    if (is_null($marker)) {
        return $this->lambdaClient->listFunctions([
            'MaxItems' => $maxItems,
        ]);
    }

    return $this->lambdaClient->listFunctions([
        'Marker' => $marker,
        'MaxItems' => $maxItems,
    ]);
}
```

- Per i dettagli sull'API, consulta la [ListFunctions](#) sezione AWS SDK per PHP API Reference.

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio mostra tutte le funzioni Lambda con dimensioni di codice ordinate

```
Get-LMFunctionList | Sort-Object -Property CodeSize | Select-Object FunctionName,
    RunTime, Timeout, CodeSize
```

Output:

| FunctionName | Runtime | Timeout |
|--------------|---------|---------|
| CodeSize     |         |         |

| -----               | -----     | ----- |
|---------------------|-----------|-------|
| -----               |           |       |
| test                | python2.7 | 3     |
| 243                 |           |       |
| MylambdaFunction123 | python3.8 | 600   |
| 659                 |           |       |
| myfuncpython1       | python3.8 | 303   |
| 675                 |           |       |

- Per i dettagli sull'API, vedere [ListFunctions](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: questo esempio mostra tutte le funzioni Lambda con dimensioni di codice ordinate

```
Get-LMFunctionList | Sort-Object -Property CodeSize | Select-Object FunctionName,
RunTime, Timeout, CodeSize
```

Output:

| FunctionName        | Runtime   | Timeout |
|---------------------|-----------|---------|
| CodeSize            |           |         |
| -----               | -----     | -----   |
| -----               |           |         |
| test                | python2.7 | 3       |
| 243                 |           |         |
| MylambdaFunction123 | python3.8 | 600     |
| 659                 |           |         |
| myfuncpython1       | python3.8 | 303     |
| 675                 |           |         |

- Per i dettagli sull'API, vedere [ListFunctions](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

## Python

### SDK per Python (Boto3)

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def list_functions(self):
        """
        Lists the Lambda functions for the current account.
        """
        try:
            func_paginator = self.lambda_client.get_paginator("list_functions")
            for func_page in func_paginator.paginate():
                for func in func_page["Functions"]:
                    print(func["FunctionName"])
                    desc = func.get("Description")
                    if desc:
                        print(f"\t{desc}")
                        print(f"\t{func['Runtime']}: {func['Handler']}")
        except ClientError as err:
            logger.error(
                "Couldn't list functions. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Per i dettagli sull'API, consulta [ListFunctions AWS SDK for Python \(Boto3\) API Reference](#).

## Ruby

### SDK per Ruby

#### Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Lists the Lambda functions for the current account.
  def list_functions
    functions = []
    @lambda_client.list_functions.each do |response|
      response['functions'].each do |function|
        functions.append(function['function_name'])
      end
    end
    functions
  end

  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error listing functions:\n #{e.message}")
  end
end
```

- Per i dettagli sull'API, consulta la [ListFunctions](#) sezione AWS SDK per Ruby API Reference.

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/** List all Lambda functions in the current Region. */
pub async fn list_functions(&self) -> Result<ListFunctionsOutput,
anyhow::Error> {
    info!("Listing lambda functions");
    self.lambda_client
        .list_functions()
        .send()
        .await
        .map_err(anyhow::Error::from)
}
```

- Per i dettagli sulle API, consulta la [ListFunctions](#) guida di riferimento all'API AWS SDK for Rust.

## SAP ABAP

### SDK per SAP ABAP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
TRY.
    oo_result = lo_lmd->listfunctions( ).      " oo_result is returned for
testing purposes. "
    DATA(lt_functions) = oo_result->get_functions( ).
    MESSAGE 'Retrieved list of Lambda functions.' TYPE 'I'.
```

```

CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_lmdserviceexception.
    MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
CATCH /aws1/cx_lmdtoomanyrequestsex.
    MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
ENDTRY.

```

- Per i dettagli sulle API, [ListFunctions](#) consulta AWS SDK for SAP ABAP API reference.

## Swift

### SDK per Swift

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

import AWSClientRuntime
import AWSLambda
import Foundation

/// Returns an array containing the names of all AWS Lambda functions
/// available to the user.
///
/// - Parameter lambdaClient: The `IAMClient` to use.
///
/// - Throws: `ExampleError.listFunctionsError`
///
/// - Returns: An array of lambda function name strings.
func getFunctionNames(lambdaClient: LambdaClient) async throws -> [String] {
    let pages = lambdaClient.listFunctionsPaginated(
        input: ListFunctionsInput()
    )

    var functionNames: [String] = []

    for try await page in pages {

```

```

        guard let functions = page.functions else {
            throw ExampleError.listFunctionsError
        }

        for function in functions {
            functionNames.append(function.functionName ?? "<unknown>")
        }
    }

    return functionNames
}

```

- Per i dettagli sull'API, consulta la [ListFunctions](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **ListProvisionedConcurrencyConfigs** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare ListProvisionedConcurrencyConfigs.

### CLI

#### AWS CLI

Per ottenere un elenco di configurazioni di simultaneità fornita

L'esempio `list-provisioned-concurrency-configs` seguente elenca le configurazioni di simultaneità fornita per la funzione specificata.

```

aws lambda list-provisioned-concurrency-configs \
    --function-name my-function

```

Output:

```

{
  "ProvisionedConcurrencyConfigs": [
    {

```

```

        "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-
function:GREEN",
        "RequestedProvisionedConcurrentExecutions": 100,
        "AvailableProvisionedConcurrentExecutions": 100,
        "AllocatedProvisionedConcurrentExecutions": 100,
        "Status": "READY",
        "LastModified": "2019-12-31T20:29:00+0000"
    },
    {
        "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-
function:BLUE",
        "RequestedProvisionedConcurrentExecutions": 100,
        "AvailableProvisionedConcurrentExecutions": 100,
        "AllocatedProvisionedConcurrentExecutions": 100,
        "Status": "READY",
        "LastModified": "2019-12-31T20:28:49+0000"
    }
]
}

```

- Per i dettagli sull'API, consulta [ListProvisionedConcurrencyConfigs AWS CLI Command Reference](#).

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio recupera l'elenco di configurazioni di simultaneità fornita per una funzione Lambda.

```
Get-LMProvisionedConcurrencyConfigList -FunctionName "MylambdaFunction123"
```

- Per i dettagli sull'API, vedere [ListProvisionedConcurrencyConfigs](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: questo esempio recupera l'elenco di configurazioni di simultaneità fornita per una funzione Lambda.

```
Get-LMProvisionedConcurrencyConfigList -FunctionName "MylambdaFunction123"
```



- Per i dettagli sull'API, vedere [ListProvisionedConcurrencyConfigs](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **ListTags** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare ListTags.

### CLI

#### AWS CLI

Per recuperare l'elenco di tag per una funzione Lambda

L'esempio list-tags seguente visualizza i tag collegati alla funzione Lambda my-function.

```
aws lambda list-tags \
  --resource arn:aws:lambda:us-west-2:123456789012:function:my-function
```

Output:

```
{
  "Tags": {
    "Category": "Web Tools",
    "Department": "Sales"
  }
}
```

Per ulteriori informazioni, consulta [Tagging delle funzioni Lambda](#) nella Guida per gli sviluppatori di AWS Lambda.

- Per i dettagli sull'API, consulta [ListTags AWS CLI](#) Command Reference.

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: recupera i tag e i relativi valori attualmente impostati sulla funzione specificata.

```
Get-LMResourceTag -Resource "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
```

Output:

| Key        | Value      |
|------------|------------|
| ---        | -----      |
| California | Sacramento |
| Oregon     | Salem      |
| Washington | Olympia    |

- Per i dettagli sull'API, vedere [ListTags](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: recupera i tag e i relativi valori attualmente impostati sulla funzione specificata.

```
Get-LMResourceTag -Resource "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
```

Output:

| Key        | Value      |
|------------|------------|
| ---        | -----      |
| California | Sacramento |
| Oregon     | Salem      |
| Washington | Olympia    |

- Per i dettagli sull'API, vedere [ListTags](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **ListVersionsByFunction** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare ListVersionsByFunction.

### CLI

#### AWS CLI

Per recuperare un elenco di versioni di una funzione

L'esempio `list-versions-by-function` seguente visualizza l'elenco di versioni per la funzione Lambda `my-function`.

```
aws lambda list-versions-by-function \
  --function-name my-function
```

Output:

```
{
  "Versions": [
    {
      "TracingConfig": {
        "Mode": "PassThrough"
      },
      "Version": "$LATEST",
      "CodeSha256": "sU0cJ2/h0ZevwV/1TxCuQqK3gDZP3i8gUoqUUVRmY6E=",
      "FunctionName": "my-function",
      "VpcConfig": {
        "SubnetIds": [],
        "VpcId": "",
        "SecurityGroupIds": []
      },
      "MemorySize": 256,
      "RevisionId": "93017fc9-59cb-41dc-901b-4845ce4bf668",
      "CodeSize": 266,
      "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:$LATEST",
      "Handler": "index.handler",
      "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qq",
      "Timeout": 3,
      "LastModified": "2019-10-01T16:47:28.490+0000",
      "Runtime": "nodejs10.x",
    }
  ]
}
```

```

        "Description": ""
    },
    {
        "TracingConfig": {
            "Mode": "PassThrough"
        },
        "Version": "1",
        "CodeSha256": "5tT2qgzYUHoqWR616pZ2dpkn/0J1FrzJmLKidWaaCgk=",
        "FunctionName": "my-function",
        "VpcConfig": {
            "SubnetIds": [],
            "VpcId": "",
            "SecurityGroupIds": []
        },
        "MemorySize": 256,
        "RevisionId": "949c8914-012e-4795-998c-e467121951b1",
        "CodeSize": 304,
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:1",
        "Handler": "index.handler",
        "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qq",
        "Timeout": 3,
        "LastModified": "2019-09-26T20:28:40.438+0000",
        "Runtime": "nodejs10.x",
        "Description": "new version"
    },
    {
        "TracingConfig": {
            "Mode": "PassThrough"
        },
        "Version": "2",
        "CodeSha256": "sU0cJ2/h0ZevwV/1TxCuQqK3gDZP3i8gUoqUUVrmY6E=",
        "FunctionName": "my-function",
        "VpcConfig": {
            "SubnetIds": [],
            "VpcId": "",
            "SecurityGroupIds": []
        },
        "MemorySize": 256,
        "RevisionId": "cd669f21-0f3d-4e1c-9566-948837f2e2ea",
        "CodeSize": 266,
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:2",

```

```

        "Handler": "index.handler",
        "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qq",
        "Timeout": 3,
        "LastModified": "2019-10-01T16:47:28.490+0000",
        "Runtime": "nodejs10.x",
        "Description": "newer version"
    }
]
}

```

Per ulteriori informazioni, consulta [Configurazione degli alias delle funzioni AWS Lambda](#) nella AWS Lambda Developer Guide.

- Per i dettagli sull'API, consulta [ListVersionsByFunction](#) Command Reference.AWS CLI

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio restituisce l'elenco di configurazioni specifiche della versione per ogni versione della funzione Lambda.

```
Get-LMVersionsByFunction -FunctionName "MylambdaFunction123"
```

Output:

| FunctionName<br>RoleName                                      | Runtime   | MemorySize | Timeout | CodeSize | LastModified |
|---------------------------------------------------------------|-----------|------------|---------|----------|--------------|
| -----                                                         | -----     | -----      | -----   | -----    | -----        |
| MylambdaFunction123<br>2020-01-10T03:20:56.390+0000<br>lambda | python3.8 | 128        | 600     | 659      |              |
| MylambdaFunction123<br>2019-12-25T09:19:02.238+0000<br>lambda | python3.8 | 128        | 5       | 1426     |              |
| MylambdaFunction123<br>2019-12-25T09:39:36.779+0000<br>lambda | python3.8 | 128        | 5       | 1426     |              |
| MylambdaFunction123<br>2019-12-25T09:52:59.872+0000<br>lambda | python3.8 | 128        | 600     | 1426     |              |

- Per i dettagli sull'API, vedere [ListVersionsByFunction](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

## Strumenti per V5 PowerShell

Esempio 1: questo esempio restituisce l'elenco di configurazioni specifiche della versione per ogni versione della funzione Lambda.

```
Get-LMVersionsByFunction -FunctionName "MylambdaFunction123"
```

Output:

| FunctionName<br>RoleName | Runtime   | MemorySize | Timeout | CodeSize | LastModified                 |
|--------------------------|-----------|------------|---------|----------|------------------------------|
| -----                    | -----     | -----      | -----   | -----    | -----                        |
| MylambdaFunction123      | python3.8 | 128        | 600     | 659      | 2020-01-10T03:20:56.390+0000 |
|                          |           |            |         |          | lambda                       |
| MylambdaFunction123      | python3.8 | 128        | 5       | 1426     | 2019-12-25T09:19:02.238+0000 |
|                          |           |            |         |          | lambda                       |
| MylambdaFunction123      | python3.8 | 128        | 5       | 1426     | 2019-12-25T09:39:36.779+0000 |
|                          |           |            |         |          | lambda                       |
| MylambdaFunction123      | python3.8 | 128        | 600     | 1426     | 2019-12-25T09:52:59.872+0000 |
|                          |           |            |         |          | lambda                       |

- Per i dettagli sull'API, vedere [ListVersionsByFunction](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **PublishVersion** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare PublishVersion.

### CLI

#### AWS CLI

Per pubblicare una nuova versione di una funzione

L'esempio publish-version seguente illustra la pubblicazione di una nuova versione della funzione Lambda my-function.

```
aws lambda publish-version \  
  --function-name my-function
```

Output:

```
{  
  "TracingConfig": {  
    "Mode": "PassThrough"  
  },  
  "CodeSha256": "dBG9m8SGdmlEjw/JYXlhhvCrAv5TxvXsbl/RMr0fT/I=",  
  "FunctionName": "my-function",  
  "CodeSize": 294,  
  "RevisionId": "f31d3d39-cc63-4520-97d4-43cd44c94c20",  
  "MemorySize": 128,  
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-  
function:3",  
  "Version": "2",  
  "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-  
zgur6bf4",  
  "Timeout": 3,  
  "LastModified": "2019-09-23T18:32:33.857+0000",  
  "Handler": "my-function.handler",  
  "Runtime": "nodejs10.x",  
  "Description": ""  
}
```

Per ulteriori informazioni, consulta [Configurazione degli alias delle funzioni AWS Lambda](#) nella AWS Lambda Developer Guide.

- Per i dettagli sull'API, consulta [PublishVersion](#) Command Reference.AWS CLI

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio crea una versione per lo snapshot esistente del codice funzione Lambda

```
Publish-LMVersion -FunctionName "MylambdaFunction123" -Description "Publishing  
Existing Snapshot of function code as a new version through Powershell"
```

- Per i dettagli sull'API, vedere [PublishVersion](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: questo esempio crea una versione per lo snapshot esistente del codice funzione Lambda

```
Publish-LMVersion -FunctionName "MylambdaFunction123" -Description "Publishing Existing Snapshot of function code as a new version through Powershell"
```

- Per i dettagli sull'API, vedere [PublishVersion](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **PutFunctionConcurrency** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare PutFunctionConcurrency.

### CLI

#### AWS CLI

Per configurare un limite di simultaneità riservata per una funzione

L'esempio put-function-concurrency seguente configura 100 esecuzioni simultanee riservate per la funzione my-function.

```
aws lambda put-function-concurrency \  
  --function-name my-function \  
  --reserved-concurrent-executions 100
```

Output:

```
{  
  "ReservedConcurrentExecutions": 100  
}
```



Per ulteriori informazioni, consulta [Reserving Concurrency for a Lambda Function](#) nella Guida per gli sviluppatori di AWS Lambda.

- Per i dettagli sull'API, consulta [PutFunctionConcurrency AWS CLI](#) Command Reference.

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio applica le impostazioni di simultaneità per l'intera funzione.

```
Write-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -  
ReservedConcurrentExecution 100
```

- Per i dettagli sull'API, vedere [PutFunctionConcurrency](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: questo esempio applica le impostazioni di simultaneità per l'intera funzione.

```
Write-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -  
ReservedConcurrentExecution 100
```

- Per i dettagli sull'API, vedere [PutFunctionConcurrency](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **PutProvisionedConcurrencyConfig** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare PutProvisionedConcurrencyConfig.

### CLI

#### AWS CLI

Per allocare la simultaneità fornita

L'esempio `put-provisioned-concurrency-config` seguente alloca 100 simultaneità fornita per l'alias `BLUE` della funzione specificata.

```
aws lambda put-provisioned-concurrency-config \  
  --function-name my-function \  
  --qualifier BLUE \  
  --provisioned-concurrent-executions 100
```

Output:

```
{  
  "Requested ProvisionedConcurrentExecutions": 100,  
  "Allocated ProvisionedConcurrentExecutions": 0,  
  "Status": "IN_PROGRESS",  
  "LastModified": "2019-11-21T19:32:12+0000"  
}
```

- Per i dettagli sull'API, consulta [PutProvisionedConcurrencyConfig AWS CLI Command Reference](#).

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio aggiunge una configurazione di simultaneità fornita all'alias di una funzione

```
Write-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -  
ProvisionedConcurrentExecution 20 -Qualifier "NewAlias1"
```

- Per i dettagli sull'API, vedere [PutProvisionedConcurrencyConfig](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: questo esempio aggiunge una configurazione di simultaneità fornita all'alias di una funzione

```
Write-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -  
ProvisionedConcurrentExecution 20 -Qualifier "NewAlias1"
```

- Per i dettagli sull'API, vedere [PutProvisionedConcurrencyConfig](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **RemovePermission** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `RemovePermission`.

### CLI

#### AWS CLI

Per rimuovere le autorizzazioni da una funzione Lambda esistente

L'esempio `remove-permission` seguente concede rimuove l'autorizzazione per invocare una funzione denominata `my-function`.

```
aws lambda remove-permission \  
  --function-name my-function \  
  --statement-id sns
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta [Using Resource-based Policies for AWS Lambda nella Lambda Developer Guide AWS](#).

- Per i dettagli sull'API, consulta AWS CLI Command [RemovePermission](#) Reference.

### PowerShell

#### Strumenti per PowerShell V4

Esempio 1: questo esempio rimuove la politica `StatementId` della funzione per la funzione Lambda specificata.

```
$policy = Get-LMPolicy -FunctionName "MylambdaFunction123" -Select Policy |  
  ConvertFrom-Json | Select-Object -ExpandProperty Statement
```

```
Remove-LMPermission -FunctionName "MylambdaFunction123" -StatementId  
$policy[0].Sid
```

- Per i dettagli sull'API, vedere [RemovePermission](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: questo esempio rimuove la politica StatementId della funzione per la funzione Lambda specificata.

```
$policy = Get-LMPolicy -FunctionName "MylambdaFunction123" -Select Policy |  
ConvertFrom-Json | Select-Object -ExpandProperty Statement  
Remove-LMPermission -FunctionName "MylambdaFunction123" -StatementId  
$policy[0].Sid
```

- Per i dettagli sull'API, vedere [RemovePermission](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **TagResource** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare TagResource.

### CLI

#### AWS CLI

Per aggiungere tag a una funzione Lambda esistente

L'esempio tag-resource seguente aggiunge un tag con il nome della chiave DEPARTMENT e un valore di Department A alla funzione Lambda specificata.

```
aws lambda tag-resource \  
--resource arn:aws:lambda:us-west-2:123456789012:function:my-function \  
--tags "DEPARTMENT=Department A"
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta [Tagging Lambda Functions](#) nella Guida per gli sviluppatori di AWS Lambda.

- Per i dettagli sull'API, consulta [TagResource AWS CLI](#) Command Reference.

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: aggiunge i tre tag (Washington, Oregon e California) e i relativi valori associati alla funzione specificata identificata dal relativo ARN.

```
Add-LMResourceTag -Resource "arn:aws:lambda:us-west-2:123456789012:function:MyFunction" -Tag @{ "Washington" = "Olympia";  
  "Oregon" = "Salem"; "California" = "Sacramento" }
```

- Per i dettagli sull'API, vedere [TagResource](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: aggiunge i tre tag (Washington, Oregon e California) e i relativi valori associati alla funzione specificata identificata dal relativo ARN.

```
Add-LMResourceTag -Resource "arn:aws:lambda:us-west-2:123456789012:function:MyFunction" -Tag @{ "Washington" = "Olympia";  
  "Oregon" = "Salem"; "California" = "Sacramento" }
```

- Per i dettagli sull'API, vedere [TagResource](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **UntagResource** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare UntagResource.

## CLI

### AWS CLI

Per rimuovere i tag da una funzione Lambda esistente

L'esempio `untag-resource` seguente rimuove il tag con il tag `DEPARTMENT` per il nome della chiave dalla funzione Lambda `my-function`.

```
aws lambda untag-resource \  
  --resource arn:aws:lambda:us-west-2:123456789012:function:my-function \  
  --tag-keys DEPARTMENT
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta [Tagging Lambda Functions](#) nella Guida per gli sviluppatori di AWS Lambda.

- Per i dettagli sull'API, consulta [UntagResource AWS CLI](#) Command Reference.

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: rimuove i tag forniti da una funzione. Il cmdlet richiederà una conferma prima di procedere, a meno che non venga specificato lo switch `-Force`. Viene effettuata una sola chiamata al servizio per rimuovere i tag.

```
Remove-LMResourceTag -Resource "arn:aws:lambda:us-  
west-2:123456789012:function:MyFunction" -TagKey  
  "Washington","Oregon","California"
```

Esempio 2: rimuove i tag forniti da una funzione. Il cmdlet richiederà una conferma prima di procedere, a meno che non venga specificato lo switch `-Force`. Una volta effettuata la chiamata al servizio per tag fornito.

```
"Washington","Oregon","California" | Remove-LMResourceTag -Resource  
  "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
```

- Per i dettagli sull'API, vedere [UntagResource](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

## Strumenti per V5 PowerShell

Esempio 1: rimuove i tag forniti da una funzione. Il cmdlet richiederà una conferma prima di procedere, a meno che non venga specificato lo switch -Force. Viene effettuata una sola chiamata al servizio per rimuovere i tag.

```
Remove-LMResourceTag -Resource "arn:aws:lambda:us-west-2:123456789012:function:MyFunction" -TagKey "Washington","Oregon","California"
```

Esempio 2: rimuove i tag forniti da una funzione. Il cmdlet richiederà una conferma prima di procedere, a meno che non venga specificato lo switch -Force. Una volta effettuata la chiamata al servizio per tag fornito.

```
"Washington","Oregon","California" | Remove-LMResourceTag -Resource "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
```

- Per i dettagli sull'API, vedere [UntagResource](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **UpdateAlias** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare UpdateAlias.

### CLI

#### AWS CLI

Per aggiornare l'alias di una funzione

L'esempio update-alias seguente aggiorna l'alias denominato LIVE che punti alla versione 3 della funzione Lambda my-function.

```
aws lambda update-alias \  
  --function-name my-function \  
  --version 3
```

```
--function-version 3 \  
--name LIVE
```

Output:

```
{  
  "FunctionVersion": "3",  
  "Name": "LIVE",  
  "AliasArn": "arn:aws:lambda:us-west-2:123456789012:function:my-  
function:LIVE",  
  "RevisionId": "594f41fb-b85f-4c20-95c7-6ca5f2a92c93",  
  "Description": "alias for live version of function"  
}
```

Per ulteriori informazioni, consulta [Configurazione degli alias delle funzioni AWS Lambda](#) nella AWS Lambda Developer Guide.

- Per i dettagli sull'API, consulta [UpdateAlias](#) Command Reference.AWS CLI

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio aggiorna la configurazione di un alias di funzione Lambda esistente. Aggiorna il RoutingConfiguration valore per spostare il 60% (0,6) del traffico alla versione 1

```
Update-LMAlias -FunctionName "MylambdaFunction123" -Description  
  " Alias for version 2" -FunctionVersion 2 -Name "newlabel1" -  
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6"}
```

- Per i dettagli sull'API, vedere [UpdateAlias](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: questo esempio aggiorna la configurazione di un alias di funzione Lambda esistente. Aggiorna il RoutingConfiguration valore per spostare il 60% (0,6) del traffico alla versione 1



```
Update-LMAlias -FunctionName "MyLambdaFunction123" -Description
" Alias for version 2" -FunctionVersion 2 -Name "newlabel1" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6"}
```

- Per i dettagli sull'API, vedere [UpdateAlias](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **UpdateFunctionCode** con un AWS SDK o CLI

Gli esempi di codice seguenti mostrano come utilizzare UpdateFunctionCode.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- [Informazioni di base](#)

### .NET

#### SDK per .NET

##### Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>
/// Update an existing Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to update.</
param>
/// <param name="bucketName">The bucket where the zip file containing
/// the Lambda function code is stored.</param>
/// <param name="key">The key name of the source code file.</param>
/// <returns>Async Task.</returns>
public async Task UpdateFunctionCodeAsync(
```

```
string functionName,  
string bucketName,  
string key)  
{  
    var functionCodeRequest = new UpdateFunctionCodeRequest  
    {  
        FunctionName = functionName,  
        Publish = true,  
        S3Bucket = bucketName,  
        S3Key = key,  
    };  
  
    var response = await  
_lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);  
    Console.WriteLine($"The Function was last modified at  
{response.LastModified}.");  
}
```

- Per i dettagli sull'API, consulta la [UpdateFunctionCode](#) sezione AWS SDK per .NET API Reference.

## C++

### SDK per C++

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
Aws::Client::ClientConfiguration clientConfig;  
    // Optional: Set to the AWS Region in which the bucket was created  
(overrides config file).  
    // clientConfig.region = "us-east-1";  
  
    Aws::Lambda::LambdaClient client(clientConfig);  
  
    Aws::Lambda::Model::UpdateFunctionCodeRequest request;
```

```

        request.SetFunctionName(LAMBDA_NAME);
        std::ifstream ifstream(CALCULATOR_LAMBDA_CODE.c_str(),
                                std::ios_base::in | std::ios_base::binary);
        if (!ifstream.is_open()) {
            std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
std::endl;

#ifdef USE_CPP_LAMBDA_FUNCTION
            std::cerr
                << "The cpp Lambda function must be built following the
instructions in the cpp_lambda/README.md file. "
                << std::endl;
#endif

            deleteLambdaFunction(client);
            deleteIamRole(clientConfig);
            return false;
        }

        Aws::StringStream buffer;
        buffer << ifstream.rdbuf();
        request.SetZipFile(
            Aws::Utils::ByteBuffer((unsigned char *) buffer.str().c_str(),
                                    buffer.str().length()));

        request.SetPublish(true);

        Aws::Lambda::Model::UpdateFunctionCodeOutcome outcome =
client.UpdateFunctionCode(
    request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda code was successfully updated." <<
std::endl;
        }
        else {
            std::cerr << "Error with Lambda::UpdateFunctionCode. "
                << outcome.GetError().GetMessage()
                << std::endl;
        }
    }

```

- Per i dettagli sull'API, consulta la [UpdateFunctionCode](#) sezione AWS SDK per C++ API Reference.

## CLI

### AWS CLI

Come aggiornare il codice di una funzione Lambda

L'esempio di `update-function-code` seguente sostituisce il codice della versione (\$LATEST) non pubblicata della funzione `my-function` con i contenuti del file zip specificato.

```
aws lambda update-function-code \  
  --function-name my-function \  
  --zip-file fileb://my-function.zip
```

Output:

```
{  
  "FunctionName": "my-function",  
  "LastModified": "2019-09-26T20:28:40.438+0000",  
  "RevisionId": "e52502d4-9320-4688-9cd6-152a6ab7490d",  
  "MemorySize": 256,  
  "Version": "$LATEST",  
  "Role": "arn:aws:iam::123456789012:role/service-role/my-function-role-  
uy3l9qq",  
  "Timeout": 3,  
  "Runtime": "nodejs10.x",  
  "TracingConfig": {  
    "Mode": "PassThrough"  
  },  
  "CodeSha256": "5tT2qgzYUHaqwR716pZ2dpkn/0J1FrzJm1KidWoaCgk=",  
  "Description": "",  
  "VpcConfig": {  
    "SubnetIds": [],  
    "VpcId": "",  
    "SecurityGroupIds": []  
  },  
  "CodeSize": 304,  
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",  
  "Handler": "index.handler"  
}
```

Per ulteriori informazioni, consulta [Configurazione delle funzioni Lambda AWS](#) nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta [UpdateFunctionCode AWS CLI Command Reference](#).

Go

SDK per Go V2

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
import (  
    "bytes"  
    "context"  
    "encoding/json"  
    "errors"  
    "log"  
    "time"  
  
    "github.com/aws/aws-sdk-go-v2/aws"  
    "github.com/aws/aws-sdk-go-v2/service/lambda"  
    "github.com/aws/aws-sdk-go-v2/service/lambda/types"  
)  
  
// FunctionWrapper encapsulates function actions used in the examples.  
// It contains an AWS Lambda service client that is used to perform user actions.  
type FunctionWrapper struct {  
    LambdaClient *lambda.Client  
}  
  
// UpdateFunctionCode updates the code for the Lambda function specified by  
// functionName.  
// The existing code for the Lambda function is entirely replaced by the code in  
// the  
// zipPackage buffer. After the update action is called, a  
// lambda.FunctionUpdatedV2Waiter  
// is used to wait until the update is successful.
```

```
func (wrapper FunctionWrapper) UpdateFunctionCode(ctx context.Context,
    functionName string, zipPackage *bytes.Buffer) types.State {
    var state types.State
    _, err := wrapper.LambdaClient.UpdateFunctionCode(ctx,
        &lambda.UpdateFunctionCodeInput{
            FunctionName: aws.String(functionName), ZipFile: zipPackage.Bytes(),
        })
    if err != nil {
        log.Panicf("Couldn't update code for function %v. Here's why: %v\n",
            functionName, err)
    } else {
        waiter := lambda.NewFunctionUpdatedV2Waiter(wrapper.LambdaClient)
        funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
            FunctionName: aws.String(functionName)}, 1*time.Minute)
        if err != nil {
            log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
                functionName, err)
        } else {
            state = funcOutput.Configuration.State
        }
    }
    return state
}
```

- Per i dettagli sull'API, consulta la [UpdateFunctionCode](#) sezione AWS SDK per Go API Reference.

## Java

### SDK per Java 2.x

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/**
 * Updates the code for an AWS Lambda function.
```

```

*
* @param awsLambda the AWS Lambda client
* @param functionName the name of the Lambda function to update
* @param bucketName the name of the S3 bucket where the function code is
located
* @param key the key (file name) of the function code in the S3 bucket
* @throws LambdaException if there is an error updating the function code
*/
public static void updateFunctionCode(LambdaClient awsLambda, String
functionName, String bucketName, String key) {
    try {
        LambdaWaiter waiter = awsLambda.waiter();
        UpdateFunctionCodeRequest functionCodeRequest =
UpdateFunctionCodeRequest.builder()
            .functionName(functionName)
            .publish(true)
            .s3Bucket(bucketName)
            .s3Key(key)
            .build();

        UpdateFunctionCodeResponse response =
awsLambda.updateFunctionCode(functionCodeRequest);
        GetFunctionConfigurationRequest getFunctionConfigRequest =
GetFunctionConfigurationRequest.builder()
            .functionName(functionName)
            .build();

        WaiterResponse<GetFunctionConfigurationResponse> waiterResponse =
waiter
            .waitUntilFunctionUpdated(getFunctionConfigRequest);
        waiterResponse.matched().response().ifPresent(System.out::println);
        System.out.println("The last modified value is " +
response.lastModified());

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

```

- Per i dettagli sull'API, consulta la [UpdateFunctionCode](#) sezione AWS SDK for Java 2.x API Reference.

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la [UpdateFunctionCode](#) sezione AWS SDK per JavaScript API Reference.

## PHP

### SDK per PHP

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
public function updateFunctionCode($functionName, $s3Bucket, $s3Key)
```



```
{  
    return $this->lambdaClient->updateFunctionCode([  
        'FunctionName' => $functionName,  
        'S3Bucket' => $s3Bucket,  
        'S3Key' => $s3Key,  
    ]);  
}
```

- Per i dettagli sull'API, consulta la [UpdateFunctionCode](#) sezione AWS SDK per PHP API Reference.

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: aggiorna la funzione denominata MyFunction " con il nuovo contenuto contenuto nel file zip specificato. Per una funzione Lambda C# .NET Core, il file zip deve contenere l'assembly compilato.

```
Update-LMFunctionCode -FunctionName MyFunction -ZipFilename .\UpdatedCode.zip
```

Esempio 2: questo esempio è simile a quello precedente ma utilizza un oggetto Amazon S3 contenente il codice aggiornato per aggiornare la funzione.

```
Update-LMFunctionCode -FunctionName MyFunction -BucketName amzn-s3-demo-bucket -  
Key UpdatedCode.zip
```

- Per i dettagli sull'API, vedere [UpdateFunctionCode](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

### Strumenti per V5 PowerShell

Esempio 1: aggiorna la funzione denominata MyFunction " con il nuovo contenuto contenuto nel file zip specificato. Per una funzione Lambda C# .NET Core, il file zip deve contenere l'assembly compilato.

```
Update-LMFunctionCode -FunctionName MyFunction -ZipFilename .\UpdatedCode.zip
```

Esempio 2: questo esempio è simile a quello precedente ma utilizza un oggetto Amazon S3 contenente il codice aggiornato per aggiornare la funzione.

```
Update-LMFunctionCode -FunctionName MyFunction -BucketName amzn-s3-demo-bucket -
Key UpdatedCode.zip
```

- Per i dettagli sull'API, vedere [UpdateFunctionCode](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

## Python

### SDK per Python (Boto3)

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def update_function_code(self, function_name, deployment_package):
        """
        Updates the code for a Lambda function by submitting a .zip archive that
        contains
        the code for the function.

        :param function_name: The name of the function to update.
        :param deployment_package: The function code to update, packaged as bytes
in
                                .zip format.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_code(
                FunctionName=function_name, ZipFile=deployment_package
            )
        except ClientError as err:
            logger.error(
                "Couldn't update function %s. Here's why: %s: %s",

```

```

        function_name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return response

```

- Per i dettagli sull'API, consulta [UpdateFunctionCode AWS SDK for Python \(Boto3\) API Reference](#).

## Ruby

### SDK per Ruby

#### Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Updates the code for a Lambda function by submitting a .zip archive that
  # contains
  # the code for the function.
  #
  # @param function_name: The name of the function to update.
  # @param deployment_package: The function code to update, packaged as bytes in
  #                           .zip format.

```

```
# @return: Data about the update, including the status.
def update_function_code(function_name, deployment_package)
  @lambda_client.update_function_code(
    function_name: function_name,
    zip_file: deployment_package
  )
  @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name }) do |w|
    w.max_attempts = 5
    w.delay = 5
  end
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error updating function code for:
#{function_name}:\n #{e.message}")
    nil
  rescue Aws::Waiters::Errors::WaiterFailed => e
    @logger.error("Failed waiting for #{function_name} to update:\n
#{e.message}")
  end
end
```

- Per i dettagli sull'API, consulta la [UpdateFunctionCode](#) sezione AWS SDK per Ruby API Reference.

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
pub async fn update_function_code(
  &self,
  zip_file: PathBuf,
  key: String,
) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
```

```

        let function_code = self.prepare_function(zip_file, Some(key)).await?;

        info!("Updating code for {}", self.lambda_name);
        let update = self
            .lambda_client
            .update_function_code()
            .function_name(self.lambda_name.clone())
            .s3_bucket(self.bucket.clone())
            .s3_key(function_code.s3_key().unwrap().to_string())
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        Ok(update)
    }

    /**
     * Upload function code from a path to a zip file.
     * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
     * The easiest way to create such a zip is to use `cargo lambda build --
output-format Zip`.
     */
    async fn prepare_function(
        &self,
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()

```

```

        .s3_bucket(self.bucket.clone())
        .s3_key(key)
        .build()
    }

```

- Per i dettagli sulle API, consulta la [UpdateFunctionCode](#) guida di riferimento all'API AWS SDK for Rust.

## SAP ABAP

### SDK per SAP ABAP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

TRY.
    oo_result = lo_lmd->updatefunctioncode(      " oo_result is returned for
testing purposes. "
        iv_functionname = iv_function_name
        iv_zipfile = io_zip_file ).

    MESSAGE 'Lambda function code updated.' TYPE 'I'.
    CATCH /aws1/cx_lmdcodesigningcfn00.
    MESSAGE 'Code signing configuration does not exist.' TYPE 'E'.
    CATCH /aws1/cx_lmdcodestorageexcdex.
    MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
    CATCH /aws1/cx_lmdcodeverification00.
    MESSAGE 'Code signature failed one or more validation checks for
signature mismatch or expiration.' TYPE 'E'.
    CATCH /aws1/cx_lmdinvalidcodesigex.
    MESSAGE 'Code signature failed the integrity check.' TYPE 'E'.
    CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
    CATCH /aws1/cx_lmdresourceconflictex.
    MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
    CATCH /aws1/cx_lmdresourcenotfoundex.

```

```

    MESSAGE 'The requested resource does not exist.' TYPE 'E'.
  CATCH /aws1/cx_lmdserviceexception.
    MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
  TYPE 'E'.
  CATCH /aws1/cx_lmdtoomanyrequestsex.
    MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
  ENDTRY.

```

- Per i dettagli sulle API, [UpdateFunctionCode](#) consulta AWS SDK for SAP ABAP API reference.

## Swift

### SDK per Swift

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

import AWSClientRuntime
import AWSLambda
import Foundation

let zipUrl = URL(fileURLWithPath: path)
let zipData: Data

// Read the function's Zip file.

do {
    zipData = try Data(contentsOf: zipUrl)
} catch {
    throw ExampleError.zipFileReadError
}

// Update the function's code and wait for the updated version to be
// ready for use.

do {

```

```
        _ = try await lambdaClient.updateFunctionCode(  
            input: UpdateFunctionCodeInput(  
                functionName: functionName,  
                zipFile: zipData  
            )  
        )  
    } catch {  
        return false  
    }
```

- Per i dettagli sull'API, consulta la [UpdateFunctionCode](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare **UpdateFunctionConfiguration** con un AWS SDK o CLI

Gli esempi di codice seguenti mostrano come utilizzare UpdateFunctionConfiguration.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- [Informazioni di base](#)

.NET

SDK per .NET

### Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/// <summary>  
/// Update the code of a Lambda function.
```



```
/// </summary>
/// <param name="functionName">The name of the function to update.</param>
/// <param name="functionHandler">The code that performs the function's
actions.</param>
/// <param name="environmentVariables">A dictionary of environment
variables.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> UpdateFunctionConfigurationAsync(
    string functionName,
    string functionHandler,
    Dictionary<string, string> environmentVariables)
{
    var request = new UpdateFunctionConfigurationRequest
    {
        Handler = functionHandler,
        FunctionName = functionName,
        Environment = new Amazon.Lambda.Model.Environment { Variables =
environmentVariables },
    };

    var response = await
_lambdaService.UpdateFunctionConfigurationAsync(request);

    Console.WriteLine(response.LastModified);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Per i dettagli sull'API, consulta la [UpdateFunctionConfiguration](#) sezione AWS SDK per .NET API Reference.

## C++

### SDK per C++

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region in which the bucket was created
    (overrides config file).
    // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

    Aws::Lambda::Model::UpdateFunctionConfigurationRequest request;
    request.SetFunctionName(LAMBDA_NAME);
    Aws::Lambda::Model::Environment environment;
    environment.AddVariables("LOG_LEVEL", "DEBUG");
    request.SetEnvironment(environment);

    Aws::Lambda::Model::UpdateFunctionConfigurationOutcome outcome =
    client.UpdateFunctionConfiguration(
        request);

    if (outcome.IsSuccess()) {
        std::cout << "The lambda configuration was successfully updated."
                  << std::endl;
        break;
    }

    else {
        std::cerr << "Error with Lambda::UpdateFunctionConfiguration. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
}

```

- Per i dettagli sull'API, consulta la [UpdateFunctionConfiguration](#) sezione AWS SDK per C++ API Reference.

## CLI

### AWS CLI

Come modificare la configurazione di una funzione

L'esempio di `update-function-configuration` seguente modifica la dimensione della memoria in 256 MB per la versione non pubblicata (\$LATEST) della funzione `my-function`.

```
aws lambda update-function-configuration \  
  --function-name my-function \  
  --memory-size 256
```

Output:

```
{  
  "FunctionName": "my-function",  
  "LastModified": "2019-09-26T20:28:40.438+0000",  
  "RevisionId": "e52502d4-9320-4688-9cd6-152a6ab7490d",  
  "MemorySize": 256,  
  "Version": "$LATEST",  
  "Role": "arn:aws:iam::123456789012:role/service-role/my-function-role-uy3l9qq",  
  "Timeout": 3,  
  "Runtime": "nodejs10.x",  
  "TracingConfig": {  
    "Mode": "PassThrough"  
  },  
  "CodeSha256": "5tT2qgzYUHaqwR716pZ2dpkn/0J1FrzJmLKidWoaCgk=",  
  "Description": "",  
  "VpcConfig": {  
    "SubnetIds": [],  
    "VpcId": "",  
    "SecurityGroupIds": []  
  },  
  "CodeSize": 304,  
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",  
  "Handler": "index.handler"  
}
```

Per ulteriori informazioni, consulta [Configurazione delle funzioni Lambda AWS](#) nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta [UpdateFunctionConfiguration AWS CLI](#) Command Reference.

## Go

## SDK per Go V2

 Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
import (
    "bytes"
    "context"
    "encoding/json"
    "errors"
    "log"
    "time"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/lambda"
    "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
    LambdaClient *lambda.Client
}

// UpdateFunctionConfiguration updates a map of environment variables configured
// for
// the Lambda function specified by functionName.
func (wrapper FunctionWrapper) UpdateFunctionConfiguration(ctx context.Context,
    functionName string, envVars map[string]string) {
    _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(ctx,
        &lambda.UpdateFunctionConfigurationInput{
            FunctionName: aws.String(functionName),
            Environment: &types.Environment{Variables: envVars},
        })
}
```

```

if err != nil {
    log.Panicf("Couldn't update configuration for %v. Here's why: %v",
        functionName, err)
}
}

```

- Per i dettagli sull'API, consulta la [UpdateFunctionConfiguration](#) sezione AWS SDK per Go API Reference.

## Java

### SDK per Java 2.x

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

/**
 * Updates the configuration of an AWS Lambda function.
 *
 * @param awsLambda      the {@link LambdaClient} instance to use for the AWS
Lambda operation
 * @param functionName  the name of the AWS Lambda function to update
 * @param handler        the new handler for the AWS Lambda function
 *
 * @throws LambdaException if there is an error while updating the function
configuration
 */
public static void updateFunctionConfiguration(LambdaClient awsLambda, String
functionName, String handler) {
    try {
        UpdateFunctionConfigurationRequest configurationRequest =
UpdateFunctionConfigurationRequest.builder()
            .functionName(functionName)
            .handler(handler)
            .runtime(Runtime.JAVA17)
            .build();
    }
}

```

```
        awsLambda.updateFunctionConfiguration(configurationRequest);

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- Per i dettagli sull'API, consulta la [UpdateFunctionConfiguration](#) sezione AWS SDK for Java 2.x API Reference.

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
const updateFunctionConfiguration = async (funcName) => {
    const client = new LambdaClient({});
    const config = readFileSync(`${dirname}../functions/config.json`).toString();
    const command = new UpdateFunctionConfigurationCommand({
        ...JSON.parse(config),
        FunctionName: funcName,
    });
    const result = await client.send(command);
    await waitFunctionUpdated({ FunctionName: funcName });
    return result;
};
```

- Per i dettagli sull'API, consulta la [UpdateFunctionConfiguration](#) sezione AWS SDK per JavaScript API Reference.

## PHP

### SDK per PHP

#### Note

C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
public function updateFunctionConfiguration($functionName, $handler,
$environment = '')
{
    return $this->lambdaClient->updateFunctionConfiguration([
        'FunctionName' => $functionName,
        'Handler' => "$handler.lambda_handler",
        'Environment' => $environment,
    ]);
}
```

- Per i dettagli sull'API, consulta la [UpdateFunctionConfiguration](#) sezione AWS SDK per PHP API Reference.

## PowerShell

### Strumenti per PowerShell V4

Esempio 1: questo esempio aggiorna la configurazione della funzione Lambda esistente

```
Update-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Handler
"lambda_function.launch_instance" -Timeout 600 -Environment_Variable
@{ "envvar1"="value";"envvar2"="value" } -Role arn:aws:iam::123456789101:role/
service-role/lambda -DeadLetterConfig_TargetArn arn:aws:sns:us-east-1:
123456789101:MyfirstTopic
```

- Per i dettagli sull'API, vedere [UpdateFunctionConfiguration](#) in Strumenti AWS per PowerShell Cmdlet Reference (V4).

## Strumenti per V5 PowerShell

Esempio 1: questo esempio aggiorna la configurazione della funzione Lambda esistente

```
Update-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Handler
"lambda_function.launch_instance" -Timeout 600 -Environment_Variable
@{ "envvar1"="value";"envvar2"="value" } -Role arn:aws:iam::123456789101:role/
service-role/lambda -DeadLetterConfig_TargetArn arn:aws:sns:us-east-1:
123456789101:MyfirstTopic
```

- Per i dettagli sull'API, vedere [UpdateFunctionConfiguration](#) in Strumenti AWS per PowerShell Cmdlet Reference (V5).

## Python

### SDK per Python (Boto3)

#### Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def update_function_configuration(self, function_name, env_vars):
        """
        Updates the environment variables for a Lambda function.

        :param function_name: The name of the function to update.
        :param env_vars: A dict of environment variables to update.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_configuration(
                FunctionName=function_name, Environment={"Variables": env_vars}
            )
```



```

except ClientError as err:
    logger.error(
        "Couldn't update function configuration %s. Here's why: %s: %s",
        function_name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return response

```

- Per i dettagli sull'API, consulta [UpdateFunctionConfiguration AWS SDK for Python \(Boto3\) API Reference](#).

## Ruby

### SDK per Ruby

#### Note

C'è di più su. [GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.](#)

```

class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Updates the environment variables for a Lambda function.
  # @param function_name: The name of the function to update.
  # @param log_level: The log level of the function.
  # @return: Data about the update, including the status.

```

```
def update_function_configuration(function_name, log_level)
  @lambda_client.update_function_configuration({
    function_name: function_name,
    environment: {
      variables: {
        'LOG_LEVEL' => log_level
      }
    }
  })

  @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name }) do |w|
    w.max_attempts = 5
    w.delay = 5
  end
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error updating configurations for
#{function_name}:\n #{e.message}")
  rescue Aws::Waiters::Errors::WaiterFailed => e
    @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
  end
end
```

- Per i dettagli sull'API, consulta la [UpdateFunctionConfiguration](#) sezione AWS SDK per Ruby API Reference.

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```
/** Update the environment for a function. */
pub async fn update_function_configuration(
    &self,
    environment: Environment,
) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
```

```

    info!(
        ?environment,
        "Updating environment for {}", self.lambda_name
    );
    let updated = self
        .lambda_client
        .update_function_configuration()
        .function_name(self.lambda_name.clone())
        .environment(environment)
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;

    Ok(updated)
}

```

- Per i dettagli sulle API, consulta la [UpdateFunctionConfiguration](#) guida di riferimento all'API AWS SDK for Rust.

## SAP ABAP

### SDK per SAP ABAP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

TRY.
    oo_result = lo_lmd->updatefunctionconfiguration(      " oo_result is
returned for testing purposes. "
        iv_functionname = iv_function_name
        iv_runtime       = iv_runtime
        iv_description    = 'Updated Lambda function'
        iv_memorysize    = iv_memory_size ).

    MESSAGE 'Lambda function configuration/settings updated.' TYPE 'I'.

```

```

CATCH /aws1/cx_lmdcodesigningcfgno00.
    MESSAGE 'Code signing configuration does not exist.' TYPE 'E'.
CATCH /aws1/cx_lmdcodeverification00.
    MESSAGE 'Code signature failed one or more validation checks for
signature mismatch or expiration.' TYPE 'E'.
CATCH /aws1/cx_lmdinvalidcodesigex.
    MESSAGE 'Code signature failed the integrity check.' TYPE 'E'.
CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_lmdresourceconflictex.
    MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
CATCH /aws1/cx_lmdresourcenotfoundex.
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.
CATCH /aws1/cx_lmdserviceexception.
    MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
CATCH /aws1/cx_lmdtoomanyrequestsex.
    MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
ENDTRY.

```

- Per i dettagli sulle API, [UpdateFunctionConfiguration](#) consulta AWS SDK for SAP ABAP API reference.

## Swift

### SDK per Swift

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

```

import AWSClientRuntime
import AWSLambda
import Foundation

/// Tell the server-side component to log debug output by setting its
/// environment's `LOG_LEVEL` to `DEBUG`.

```

```
///
/// - Parameters:
///   - lambdaClient: The `LambdaClient` to use.
///   - functionName: The name of the AWS Lambda function to enable debug
///     logging for.
///
/// - Throws: `ExampleError.environmentResponseMissingError`,
///   `ExampleError.updateFunctionConfigurationError`,
///   `ExampleError.environmentVariablesMissingError`,
///   `ExampleError.logLevelIncorrectError`,
///   `ExampleError.updateFunctionConfigurationError`
func enableDebugLogging(lambdaClient: LambdaClient, functionName: String)
async throws {
    let envVariables = [
        "LOG_LEVEL": "DEBUG"
    ]
    let environment = LambdaClientTypes.Environment(variables: envVariables)

    do {
        let output = try await lambdaClient.updateFunctionConfiguration(
            input: UpdateFunctionConfigurationInput(
                environment: environment,
                functionName: functionName
            )
        )

        guard let response = output.environment else {
            throw ExampleError.environmentResponseMissingError
        }

        if response.error != nil {
            throw ExampleError.updateFunctionConfigurationError
        }

        guard let retVariables = response.variables else {
            throw ExampleError.environmentVariablesMissingError
        }

        for envVar in retVariables {
            if envVar.key == "LOG_LEVEL" && envVar.value != "DEBUG" {
                print("*** Log level is not set to DEBUG!")
                throw ExampleError.logLevelIncorrectError
            }
        }
    }
}
```

```
        } catch {  
            throw ExampleError.updateFunctionConfigurationError  
        }  
    }  
}
```

- Per i dettagli sull'API, consulta la [UpdateFunctionConfiguration](#) guida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Scenari per l'utilizzo di Lambda AWS SDK

I seguenti esempi di codice mostrano come implementare scenari comuni in Lambda con AWS gli SDK. Questi scenari illustrano come eseguire attività specifiche richiamando più funzioni all'interno di Lambda o combinate con altri Servizi AWS. Ogni scenario include un collegamento al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice.

Gli scenari sono relativi a un livello intermedio di esperienza per aiutarti a comprendere le azioni di servizio nel contesto.

### Esempi

- [Conferma automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda utilizzando un AWS SDK](#)
- [Esegui automaticamente la migrazione di utenti Amazon Cognito noti con una funzione Lambda utilizzando un AWS SDK](#)
- [Crea un'API API Gateway REST per tenere traccia COVID-19 dei dati](#)
- [Creazione di una REST API per la libreria di prestiti](#)
- [Creazione di un'applicazione di messaggistica con Step Functions](#)
- [Crea un'API REST con integrazione di funzioni proxy](#)
- [Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette](#)
- [Creazione di un'applicazione di chat websocket con API Gateway](#)
- [Crea un'applicazione che analizza il feedback dei clienti e sintetizza l'audio](#)

- [Creazione di una dashboard di monitoraggio con il nome della funzione come variabile](#)
- [Creazione della tua prima funzione serverless](#)
- [Richiamo a una funzione Lambda da un browser](#)
- [Trasformare i dati per l'applicazione con S3 Object Lambda](#)
- [Utilizzo di Gateway API per invocare una funzione Lambda](#)
- [Utilizzo di Step Functions per richiamare le funzioni Lambda](#)
- [Utilizzo degli eventi pianificati per invocare una funzione Lambda](#)
- [Utilizzare l'API Amazon Neptune per sviluppare una funzione Lambda che esegua query sui dati del grafico](#)
- [Utilizzo delle variabili di proprietà nei dashboard di monitoraggio per monitorare più funzioni serverless](#)
- [Scrivi dati di attività personalizzati con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito utilizzando un AWS SDK](#)

## Conferma automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda utilizzando un AWS SDK

L'esempio di codice seguente mostra come confermare automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda.

- Configura un pool di utenti per chiamare una funzione Lambda per il trigger PreSignUp.
- Registra un utente con Amazon Cognito.
- La funzione Lambda esegue la scansione di una tabella DynamoDB e conferma automaticamente gli utenti noti.
- Accedi come nuovo utente, quindi elimina le risorse.

Go

SDK per Go V2

### Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
import (
    "context"
    "errors"
    "log"
    "strings"
    "user_pools_and_lambda_triggers/actions"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// AutoConfirm separates the steps of this scenario into individual functions so
// that
// they are simpler to read and understand.
type AutoConfirm struct {
    helper      IScenarioHelper
    questioner  demotools.IQuestioner
    resources   Resources
    cognitoActor *actions.CognitoActions
}

// NewAutoConfirm constructs a new auto confirm runner.
func NewAutoConfirm(sdkConfig aws.Config, questioner demotools.IQuestioner,
    helper IScenarioHelper) AutoConfirm {
    scenario := AutoConfirm{
        helper:      helper,
        questioner:  questioner,
        resources:   Resources{},
        cognitoActor: &actions.CognitoActions{CognitoClient:
            cognitoidentityprovider.NewFromConfig(sdkConfig)},
    }
    scenario.resources.init(scenario.cognitoActor, questioner)
    return scenario
}

// AddPreSignUpTrigger adds a Lambda handler as an invocation target for the
// PreSignUp trigger.
func (runner *AutoConfirm) AddPreSignUpTrigger(ctx context.Context, userPoolId
    string, functionArn string) {
```



```

log.Printf("Let's add a Lambda function to handle the PreSignUp trigger from
Cognito.\n" +
    "This trigger happens when a user signs up, and lets your function take action
before the main Cognito\n" +
    "sign up processing occurs.\n")
err := runner.cognitoActor.UpdateTriggers(
    ctx, userPoolId,
    actions.TriggerInfo{Trigger: actions.PreSignUp, HandlerArn:
aws.String(functionArn)})
if err != nil {
    panic(err)
}
log.Printf("Lambda function %v added to user pool %v to handle the PreSignUp
trigger.\n",
    functionArn, userPoolId)
}

// SignUpUser signs up a user from the known user table with a password you
specify.
func (runner *AutoConfirm) SignUpUser(ctx context.Context, clientId string,
usersTable string) (string, string) {
    log.Println("Let's sign up a user to your Cognito user pool. When the user's
email matches an email in the\n" +
        "DynamoDB known users table, it is automatically verified and the user is
confirmed.")

    knownUsers, err := runner.helper.GetKnownUsers(ctx, usersTable)
    if err != nil {
        panic(err)
    }
    userChoice := runner.questioner.AskChoice("Which user do you want to use?\n",
knownUsers.UserNameList())
    user := knownUsers.Users[userChoice]

    var signedUp bool
    var userConfirmed bool
    password := runner.questioner.AskPassword("Enter a password that has at least
eight characters, uppercase, lowercase, numbers and symbols.\n"+
        "(the password will not display as you type):", 8)
    for !signedUp {
        log.Printf("Signing up user '%v' with email '%v' to Cognito.\n", user.UserName,
user.UserEmail)
        userConfirmed, err = runner.cognitoActor.SignUp(ctx, clientId, user.UserName,
password, user.UserEmail)
    }
}

```

```

    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            password = runner.questioner.AskPassword("Enter another password:", 8)
        } else {
            panic(err)
        }
    } else {
        signedUp = true
    }
}
log.Printf("User %v signed up, confirmed = %v.\n", user.UserName, userConfirmed)

log.Println(strings.Repeat("-", 88))

return user.UserName, password
}

// SignInUser signs in a user.
func (runner *AutoConfirm) SignInUser(ctx context.Context, clientId string,
    userName string, password string) string {
    runner.questioner.Ask("Press Enter when you're ready to continue.")
    log.Printf("Let's sign in as %v...\n", userName)
    authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
    if err != nil {
        panic(err)
    }
    log.Printf("Successfully signed in. Your access token starts with: %v...\n",
        (*authResult.AccessToken)[:10])
    log.Println(strings.Repeat("-", 88))
    return *authResult.AccessToken
}

// Run runs the scenario.
func (runner *AutoConfirm) Run(ctx context.Context, stackName string) {
    defer func() {
        if r := recover(); r != nil {
            log.Println("Something went wrong with the demo.")
            runner.resources.Cleanup(ctx)
        }
    }()

    log.Println(strings.Repeat("-", 88))
    log.Printf("Welcome\n")

```

```

log.Println(strings.Repeat("-", 88))

stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
if err != nil {
    panic(err)
}
runner.resources.userPoolId = stackOutputs["UserPoolId"]
runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])

runner.AddPreSignUpTrigger(ctx, stackOutputs["UserPoolId"],
stackOutputs["AutoConfirmFunctionArn"])
runner.resources.triggers = append(runner.resources.triggers, actions.PreSignUp)
userName, password := runner.SignUpUser(ctx, stackOutputs["UserPoolClientId"],
stackOutputs["TableName"])
runner.helper.ListRecentLogEvents(ctx, stackOutputs["AutoConfirmFunction"])
runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password))

runner.resources.Cleanup(ctx)

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}

```

Gestisci il trigger PreSignUp con una funzione Lambda.

```

import (
    "context"
    "log"
    "os"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    dynamodbtypes "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"

```

```

)

const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
// format.
type UserInfo struct {
    UserName string `dynamodbav:"UserName"`
    UserEmail string `dynamodbav:"UserEmail"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
    userEmail, err := attributevalue.Marshal(user.UserEmail)
    if err != nil {
        panic(err)
    }
    return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
    dynamoClient *dynamodb.Client
}

// HandleRequest handles the PreSignUp event by looking up a user in an Amazon
// DynamoDB table and
// specifying whether they should be confirmed and verified.
func (h *handler) HandleRequest(ctx context.Context, event
events.CognitoEventUserPoolsPreSignup) (events.CognitoEventUserPoolsPreSignup,
error) {
    log.Printf("Received presignup from %v for user '%v'", event.TriggerSource,
event.UserName)
    if event.TriggerSource != "PreSignUp_SignUp" {
        // Other trigger sources, such as PreSignUp_AdminInitiateAuth, ignore the
        // response from this handler.
        return event, nil
    }
    tableName := os.Getenv(TABLE_NAME)
    user := UserInfo{
        UserEmail: event.Request.UserAttributes["email"],
    }
    log.Printf("Looking up email %v in table %v.\n", user.UserEmail, tableName)
    output, err := h.dynamoClient.GetItem(ctx, &dynamodb.GetItemInput{
        Key:      user.GetKey(),

```

```

    TableName: aws.String(tableName),
  })
  if err != nil {
    log.Printf("Error looking up email %v.\n", user.UserEmail)
    return event, err
  }
  if output.Item == nil {
    log.Printf("Email %v not found. Email verification is required.\n",
      user.UserEmail)
    return event, err
  }

  err = attributevalue.UnmarshalMap(output.Item, &user)
  if err != nil {
    log.Printf("Couldn't unmarshal DynamoDB item. Here's why: %v\n", err)
    return event, err
  }

  if user.UserName != event.UserName {
    log.Printf("UserEmail %v found, but stored UserName '%v' does not match
      supplied UserName '%v'. Verification is required.\n",
      user.UserEmail, user.UserName, event.UserName)
  } else {
    log.Printf("UserEmail %v found with matching UserName %v. User is confirmed.
\n", user.UserEmail, user.UserName)
    event.Response.AutoConfirmUser = true
    event.Response.AutoVerifyEmail = true
  }

  return event, err
}

func main() {
  ctx := context.Background()
  sdkConfig, err := config.LoadDefaultConfig(ctx)
  if err != nil {
    log.Panicln(err)
  }
  h := handler{
    dynamoClient: dynamodb.NewFromConfig(sdkConfig),
  }
  lambda.Start(h.HandleRequest)
}

```

Crea una struttura che esegue operazioni comuni.

```
import (
    "context"
    "log"
    "strings"
    "time"
    "user_pools_and_lambda_triggers/actions"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudformation"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this
// example.
type IScenarioHelper interface {
    Pause(secs int)
    GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
        error)
    PopulateUserTable(ctx context.Context, tableName string)
    GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
    AddKnownUser(ctx context.Context, tableName string, user actions.User)
    ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
// example.
type ScenarioHelper struct {
    questioner demotools.IQuestioner
    dynamoActor *actions.DynamoActions
    cfnActor     *actions.CloudFormationActions
    cwActor      *actions.CloudWatchLogsActions
    isTestRun    bool
}

// NewScenarioHelper constructs a new scenario helper.
```

```

func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
ScenarioHelper {
    scenario := ScenarioHelper{
        questioner: questioner,
        dynamoActor: &actions.DynamoActions{DynamoClient:
dynamodb.NewFromConfig(sdkConfig)},
        cfnActor: &actions.CloudFormationActions{CfnClient:
cloudformation.NewFromConfig(sdkConfig)},
        cwlActor: &actions.CloudWatchLogsActions{CwlClient:
cloudwatchlogs.NewFromConfig(sdkConfig)},
    }
    return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
    if !helper.isTestRun {
        time.Sleep(time.Duration(secs) * time.Second)
    }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
string) (actions.StackOutputs, error) {
    return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
string) {
    log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
this example.\n", tableName)
    err := helper.dynamoActor.PopulateTable(ctx, tableName)
    if err != nil {
        panic(err)
    }
}

// GetKnownUsers gets the users from the known users table in a structured
format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
(actions.UserList, error) {
    knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)

```

```
    if err != nil {
        log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
            tableName, err)
    }
    return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
    user actions.User) {
    log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
        table...\n",
        user.UserName, user.UserEmail)
    err := helper.dynamoActor.AddUser(ctx, tableName, user)
    if err != nil {
        panic(err)
    }
}

// ListRecentLogEvents gets the most recent log stream and events for the
// specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
    functionName string) {
    log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
    helper.Pause(10)
    log.Println("Okay, let's check the logs to find what's happened recently with
        your Lambda function.")
    logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
    if err != nil {
        panic(err)
    }
    log.Printf("Getting some recent events from log stream %v\n",
        *logStream.LogStreamName)
    events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
        *logStream.LogStreamName, 10)
    if err != nil {
        panic(err)
    }
    for _, event := range events {
        log.Printf("\t%v", *event.Message)
    }
    log.Println(strings.Repeat("-", 88))
}
```



Crea una struttura che racchiude le azioni di Amazon Cognito.

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
// trigger.
type Trigger int

const (
    PreSignUp Trigger = iota
    UserMigration
    PostAuthentication
)

type TriggerInfo struct {
    Trigger    Trigger
    HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
// is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
    string, triggers ...TriggerInfo) error {
    output, err := actor.CognitoClient.DescribeUserPool(ctx,
        &cognitoidentityprovider.DescribeUserPoolInput{
```

```

    UserPoolId: aws.String(userPoolId),
  })
  if err != nil {
    log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
      userPoolId, err)
    return err
  }
  lambdaConfig := output.UserPool.LambdaConfig
  for _, trigger := range triggers {
    switch trigger.Trigger {
    case PreSignUp:
      lambdaConfig.PreSignUp = trigger.HandlerArn
    case UserMigration:
      lambdaConfig.UserMigration = trigger.HandlerArn
    case PostAuthentication:
      lambdaConfig.PostAuthentication = trigger.HandlerArn
    }
  }
  _, err = actor.CognitoClient.UpdateUserPool(ctx,
    &cognitoidentityprovider.UpdateUserPoolInput{
      UserPoolId:    aws.String(userPoolId),
      LambdaConfig: lambdaConfig,
    })
  if err != nil {
    log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
  }
  return err
}

// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
  string, password string, userEmail string) (bool, error) {
  confirmed := false
  output, err := actor.CognitoClient.SignUp(ctx,
    &cognitoidentityprovider.SignUpInput{
      ClientId: aws.String(clientId),
      Password: aws.String(password),
      Username: aws.String(userName),
      UserAttributes: []types.AttributeType{
        {Name: aws.String("email"), Value: aws.String(userEmail)},
      },
    })
  if err != nil {
    log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
  return confirmed, err
}

```

```

if err != nil {
    var invalidPassword *types.InvalidPasswordException
    if errors.As(err, &invalidPassword) {
        log.Println(*invalidPassword.Message)
    } else {
        log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
    }
} else {
    confirmed = output.UserConfirmed
}
return confirmed, err
}

// SignIn signs in a user to Amazon Cognito using a username and password
// authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
string, password string) (*types.AuthenticationResultType, error) {
    var authResult *types.AuthenticationResultType
    output, err := actor.CognitoClient.InitiateAuth(ctx,
        &cognitoidentityprovider.InitiateAuthInput{
            AuthFlow:      "USER_PASSWORD_AUTH",
            ClientId:      aws.String(clientId),
            AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
        })
    if err != nil {
        var resetRequired *types.PasswordResetRequiredException
        if errors.As(err, &resetRequired) {
            log.Println(*resetRequired.Message)
        } else {
            log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
        }
    } else {
        authResult = output.AuthenticationResult
    }
    return authResult, err
}

// ForgotPassword starts a password recovery flow for a user. This flow typically
// sends a confirmation code
// to the user's configured notification destination, such as email.

```

```

func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
    userName string) (*types.CodeDeliveryDetailsType, error) {
    output, err := actor.CognitoClient.ForgotPassword(ctx,
        &cognitoidentityprovider.ForgotPasswordInput{
            ClientId: aws.String(clientId),
            Username: aws.String(userName),
        })
    if err != nil {
        log.Printf("Couldn't start password reset for user '%v'. Here's why: %v\n",
            userName, err)
    }
    return output.CodeDeliveryDetails, err
}

// ConfirmForgotPassword confirms a user with a confirmation code and a new
// password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
    string, code string, userName string, password string) error {
    _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
        &cognitoidentityprovider.ConfirmForgotPasswordInput{
            ClientId:      aws.String(clientId),
            ConfirmationCode: aws.String(code),
            Password:      aws.String(password),
            Username:      aws.String(userName),
        })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
        }
    }
    return err
}

// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
    string) error {

```

```

_, err := actor.CognitoClient.DeleteUser(ctx,
&cognitoidentityprovider.DeleteUserInput{
    AccessToken: aws.String(userAccessToken),
})
if err != nil {
    log.Printf("Couldn't delete user. Here's why: %v\n", err)
}
return err
}

// AdminCreateUser uses administrator credentials to add a user to a user pool.
// This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
string, userName string, userEmail string) error {
_, err := actor.CognitoClient.AdminCreateUser(ctx,
&cognitoidentityprovider.AdminCreateUserInput{
    UserPoolId:      aws.String(userPoolId),
    Username:        aws.String(userName),
    MessageAction:    types.MessageActionTypeSuppress,
    UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
aws.String(userEmail)}}},
})
if err != nil {
    var userExists *types.UsernameExistsException
    if errors.As(err, &userExists) {
        log.Printf("User %v already exists in the user pool.", userName)
        err = nil
    } else {
        log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
    }
}
return err
}

// AdminSetUserPassword uses administrator credentials to set a password for a
// user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
string, userName string, password string) error {

```

```
_, err := actor.CognitoClient.AdminSetUserPassword(ctx,
&cognitoidentityprovider.AdminSetUserPasswordInput{
    Password:    aws.String(password),
    UserPoolId:  aws.String(userPoolId),
    Username:    aws.String(userName),
    Permanent:   true,
})
if err != nil {
    var invalidPassword *types.InvalidPasswordException
    if errors.As(err, &invalidPassword) {
        log.Println(*invalidPassword.Message)
    } else {
        log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
err)
    }
}
return err
}
```

Crea una struttura che racchiude le azioni di DynamoDB.

```
import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type DynamoActions struct {
    DynamoClient *dynamodb.Client
}

// User defines structured user data.
```

```
type User struct {
    UserName string
    UserEmail string
    LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
    UserPoolId string
    ClientId    string
    Time        string
}

// UserList defines a list of users.
type UserList struct {
    Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
strings.
func (users *UserList) UserNameList() []string {
    names := make([]string, len(users.Users))
    for i := 0; i < len(users.Users); i++ {
        names[i] = users.Users[i].UserName
    }
    return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
error {
    var err error
    var item map[string]types.AttributeValue
    var writeReqs []types.WriteRequest
    for i := 1; i < 4; i++ {
        item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
        if err != nil {
            log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
err)
            return err
        }
        writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
&types.PutRequest{Item: item}}})
    }
```

```

    }
    _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
        RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
    })
    if err != nil {
        log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
            tableName, err)
    }
    return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
    error) {
    var userList UserList
    output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
        TableName: aws.String(tableName),
    })
    if err != nil {
        log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
            err)
    } else {
        err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
        if err != nil {
            log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
        }
    }
    return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user
    User) error {
    userItem, err := attributevalue.MarshalMap(user)
    if err != nil {
        log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
    }
    _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
        Item:      userItem,
        TableName: aws.String(tableName),
    })
    if err != nil {
        log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
    }
}

```



```
    return err
}
```

Crea una struttura che racchiuda le azioni di CloudWatch Logs.

```
import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
    CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
    functionName string) (types.LogStream, error) {
    var logStream types.LogStream
    logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
    output, err := actor.CwlClient.DescribeLogStreams(ctx,
        &cloudwatchlogs.DescribeLogStreamsInput{
            Descending:    aws.Bool(true),
            Limit:         aws.Int32(1),
            LogGroupName:  aws.String(logGroupName),
            OrderBy:      types.OrderByLastEventTime,
        })
    if err != nil {
        log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
            logGroupName, err)
    } else {
        logStream = output.LogStreams[0]
    }
    return logStream, err
}
```

```
// GetLogEvents gets the most recent eventCount events from the specified log
// stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
string, logStreamName string, eventCount int32) (
[]types.OutputLogEvent, error) {
var events []types.OutputLogEvent
logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
output, err := actor.CwlClient.GetLogEvents(ctx,
&cloudwatchlogs.GetLogEventsInput{
    LogStreamName: aws.String(logStreamName),
    Limit:         aws.Int32(eventCount),
    LogGroupName:  aws.String(logGroupName),
})
if err != nil {
    log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
logStreamName, err)
} else {
    events = output.Events
}
return events, err
}
```

Crea una struttura che racchiuda le azioni. CloudFormation

```
import (
    "context"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
    CfnClient *cloudformation.Client
}
```

```
// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
// structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
string) StackOutputs {
    output, err := actor.CfnClient.DescribeStacks(ctx,
        &cloudformation.DescribeStacksInput{
            StackName: aws.String(stackName),
        })
    if err != nil || len(output.Stacks) == 0 {
        log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
            stackName, err)
    }
    stackOutputs := StackOutputs{}
    for _, out := range output.Stacks[0].Outputs {
        stackOutputs[*out.OutputKey] = *out.OutputValue
    }
    return stackOutputs
}
```

Eliminare le risorse.

```
import (
    "context"
    "log"
    "user_pools_and_lambda_triggers/actions"

    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
    userPoolId      string
    userAccessTokens []string
    triggers        []actions.Trigger

    cognitoActor *actions.CognitoActions
    questioner   demotools.IQuestioner
}
```

```

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
demotools.IQuestioner) {
    resources.userAccessTokens = []string{}
    resources.triggers = []actions.Trigger{}
    resources.cognitoActor = cognitoActor
    resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
    defer func() {
        if r := recover(); r != nil {
            log.Printf("Something went wrong during cleanup.\n%v\n", r)
            log.Println("Use the AWS Management Console to remove any remaining resources
\n" +
                "that were created for this scenario.")
        }
    }()

    wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
resources that were created "+
        "during this demo (y/n)?", "y")
    if wantDelete {
        for _, accessToken := range resources.userAccessTokens {
            err := resources.cognitoActor.DeleteUser(ctx, accessToken)
            if err != nil {
                log.Println("Couldn't delete user during cleanup.")
                panic(err)
            }
            log.Println("Deleted user.")
        }
        triggerList := make([]actions.TriggerInfo, len(resources.triggers))
        for i := 0; i < len(resources.triggers); i++ {
            triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
HandlerArn: nil}
        }
        err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
triggerList...)
        if err != nil {
            log.Println("Couldn't update Cognito triggers during cleanup.")
            panic(err)
        }
        log.Println("Removed Cognito triggers from user pool.")
    } else {

```

```
    log.Println("Be sure to remove resources when you're done with them to avoid  
    unexpected charges!")  
  }  
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - [DeleteUser](#)
  - [InitiateAuth](#)
  - [SignUp](#)
  - [UpdateUserPool](#)

## JavaScript

### SDK per (v3 JavaScript )

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Configura un'esecuzione interattiva dello "Scenario". Gli esempi JavaScript (v3) condividono uno Scenario runner per semplificare esempi complessi. Il codice sorgente completo è attivo. [GitHub](#)

```
import { AutoConfirm } from "./scenario-auto-confirm.js";  
  
/**  
 * The context is passed to every scenario. Scenario steps  
 * will modify the context.  
 */  
const context = {  
  errors: [],  
  users: [  
    {  
      UserName: "test_user_1",
```

```

        UserEmail: "test_email_1@example.com",
    },
    {
        UserName: "test_user_2",
        UserEmail: "test_email_2@example.com",
    },
    {
        UserName: "test_user_3",
        UserEmail: "test_email_3@example.com",
    },
  ],
};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 * class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Demonstrate automatically confirming known users in a database.
  "auto-confirm": AutoConfirm(context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { parseScenarioArgs } from "@aws-doc-sdk-examples/lib/scenario/index.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Cognito user pools and triggers",
    description:
      "Demonstrate how to use the AWS SDKs to customize Amazon Cognito authentication behavior.",
  });
}

```

Questo scenario dimostra la conferma automatica di un utente noto. Orchestra i passaggi di esempio.

```

import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  Scenario,

```

```

    ScenarioAction,
    ScenarioInput,
    ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";

import {
    getStackOutputs,
    logCleanUpReminder,
    promptForStackName,
    promptForStackRegion,
    skipWhenErrors,
} from "../steps-common.js";
import { populateTable } from "../actions/dynamodb-actions.js";
import {
    addPreSignUpHandler,
    deleteUser,
    getUser,
    signIn,
    signUpUser,
} from "../actions/cognito-actions.js";
import {
    getLatestLogStreamForLambda,
    getLogEvents,
} from "../actions/cloudwatch-logs-actions.js";

/**
 * @typedef {{
 *   errors: Error[],
 *   password: string,
 *   users: { UserName: string, UserEmail: string }[],
 *   selectedUser?: string,
 *   stackName?: string,
 *   stackRegion?: string,
 *   token?: string,
 *   confirmDeleteSignedInUser?: boolean,
 *   TableName?: string,
 *   UserPoolClientId?: string,
 *   UserPoolId?: string,
 *   UserPoolArn?: string,
 *   AutoConfirmHandlerArn?: string,
 *   AutoConfirmHandlerName?: string
 * }} State
 */

```

```
const greeting = new ScenarioOutput(
  "greeting",
  (/** @type {State} */ state) => `This demo will populate some users into the \
database created as part of the "${state.stackName}" stack. \
Then the AutoConfirmHandler will be linked to the PreSignUp \
trigger from Cognito. Finally, you will choose a user to sign up.` ,
  { skipWhen: skipWhenErrors },
);

const logPopulatingUsers = new ScenarioOutput(
  "logPopulatingUsers",
  "Populating the DynamoDB table with some users.",
  { skipWhenErrors: skipWhenErrors },
);

const logPopulatingUsersComplete = new ScenarioOutput(
  "logPopulatingUsersComplete",
  "Done populating users.",
  { skipWhen: skipWhenErrors },
);

const populateUsers = new ScenarioAction(
  "populateUsers",
  async (/** @type {State} */ state) => {
    const [_, err] = await populateTable({
      region: state.stackRegion,
      tableName: state.TableName,
      items: state.users,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTrigger = new ScenarioOutput(
  "logSetupSignUpTrigger",
  "Setting up the PreSignUp trigger for the Cognito User Pool.",
  { skipWhen: skipWhenErrors },
);
```



```

const setupSignUpTrigger = new ScenarioAction(
  "setupSignUpTrigger",
  async (** @type {State} */ state) => {
    const [_, err] = await addPreSignUpHandler({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      handlerArn: state.AutoConfirmHandlerArn,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTriggerComplete = new ScenarioOutput(
  "logSetupSignUpTriggerComplete",
  (
    /** @type {State} */ state,
  ) => `The lambda function "${state.AutoConfirmHandlerName}" \
has been configured as the PreSignUp trigger handler for the user pool
"${state.UserPoolId}".`,
  { skipWhen: skipWhenErrors },
);

const selectUser = new ScenarioInput(
  "selectedUser",
  "Select a user to sign up.",
  {
    type: "select",
    choices: (** @type {State} */ state) => state.users.map((u) => u.UserName),
    skipWhen: skipWhenErrors,
    default: (** @type {State} */ state) => state.users[0].UserName,
  },
);

const checkIfUserAlreadyExists = new ScenarioAction(
  "checkIfUserAlreadyExists",
  async (** @type {State} */ state) => {
    const [user, err] = await getUser({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,

```

```

        username: state.selectedUser,
    });

    if (err?.name === "UserNotFoundException") {
        // Do nothing. We're not expecting the user to exist before
        // sign up is complete.
        return;
    }

    if (err) {
        state.errors.push(err);
        return;
    }

    if (user) {
        state.errors.push(
            new Error(
                `The user "${state.selectedUser}" already exists in the user pool
                "${state.UserPoolId}".`,
            ),
        );
    }
},
{
    skipWhen: skipWhenErrors,
},
);

const createPassword = new ScenarioInput(
    "password",
    "Enter a password that has at least eight characters, uppercase, lowercase,
    numbers and symbols.",
    { type: "password", skipWhen: skipWhenErrors, default: "Abcd1234!" },
);

const logSignUpExistingUser = new ScenarioOutput(
    "logSignUpExistingUser",
    (/** @type {State} */ state) => `Signing up user "${state.selectedUser}".`,
    { skipWhen: skipWhenErrors },
);

const signUpExistingUser = new ScenarioAction(
    "signUpExistingUser",
    async (/** @type {State} */ state) => {

```

```
const signUp = (password) =>
  signUpUser({
    region: state.stackRegion,
    userPoolClientId: state.UserPoolClientId,
    username: state.selectedUser,
    email: state.users.find((u) => u.UserName === state.selectedUser)
      .UserEmail,
    password,
  });

let [_, err] = await signUp(state.password);

while (err?.name === "InvalidPasswordException") {
  console.warn("The password you entered was invalid.");
  await createPassword.handle(state);
  [_, err] = await signUp(state.password);
}

if (err) {
  state.errors.push(err);
}
},
{ skipWhen: skipWhenErrors },
);

const logSignUpExistingUserComplete = new ScenarioOutput(
  "logSignUpExistingUserComplete",
  (/** @type {State} */ state) =>
    `"${state.selectedUser}" was signed up successfully.`,
  { skipWhen: skipWhenErrors },
);

const logLambdaLogs = new ScenarioAction(
  "logLambdaLogs",
  async (/** @type {State} */ state) => {
    console.log(
      "Waiting a few seconds to let Lambda write to CloudWatch Logs...\n",
    );
    await wait(10);

    const [logStream, logStreamErr] = await getLatestLogStreamForLambda({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
    });
  });
```

```
    if (logStreamErr) {
      state.errors.push(logStreamErr);
      return;
    }

    console.log(
      `Getting some recent events from log stream "${logStream.logStreamName}"`,
    );
    const [logEvents, logEventsErr] = await getLogEvents({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
      eventCount: 10,
      logStreamName: logStream.logStreamName,
    });
    if (logEventsErr) {
      state.errors.push(logEventsErr);
      return;
    }

    console.log(logEvents.map((ev) => `\t${ev.message}`).join(""));
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUser = new ScenarioOutput(
  "logSignInUser",
  (/** @type {State} */ state) => `Let's sign in as ${state.selectedUser}`,
  { skipWhen: skipWhenErrors },
);

const signInUser = new ScenarioAction(
  "signInUser",
  async (/** @type {State} */ state) => {
    const [response, err] = await signIn({
      region: state.stackRegion,
      clientId: state.UserPoolClientId,
      username: state.selectedUser,
      password: state.password,
    });
  });

  if (err?.name === "PasswordResetRequiredException") {
    state.errors.push(new Error("Please reset your password."));
    return;
  }
}
```

```
    if (err) {
      state.errors.push(err);
      return;
    }

    state.token = response?.AuthenticationResult?.AccessToken;
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUserComplete = new ScenarioOutput(
  "logSignInUserComplete",
  (/** @type {State} */ state) =>
    `Successfully signed in. Your access token starts with:
    ${state.token.slice(0, 11)}`,
  { skipWhen: skipWhenErrors },
);

const confirmDeleteSignedInUser = new ScenarioInput(
  "confirmDeleteSignedInUser",
  "Do you want to delete the currently signed in user?",
  { type: "confirm", skipWhen: skipWhenErrors },
);

const deleteSignedInUser = new ScenarioAction(
  "deleteSignedInUser",
  async (/** @type {State} */ state) => {
    const [_, err] = await deleteUser({
      region: state.stackRegion,
      accessToken: state.token,
    });

    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: (/** @type {State} */ state) =>
      skipWhenErrors(state) || !state.confirmDeleteSignedInUser,
  },
);

const logErrors = new ScenarioOutput(
```

```
"logErrors",
(** @type {State}*/ state) => {
  const errorList = state.errors
    .map((err) => ` - ${err.name}: ${err.message}`)
    .join("\n");
  return `Scenario errors found:\n${errorList}`;
},
{
  // Don't log errors when there aren't any!
  skipWhen: (** @type {State} */ state) => state.errors.length === 0,
},
);

export const AutoConfirm = (context) =>
  new Scenario(
    "AutoConfirm",
    [
      promptForStackName,
      promptForStackRegion,
      getStackOutputs,
      greeting,
      logPopulatingUsers,
      populateUsers,
      logPopulatingUsersComplete,
      logSetupSignUpTrigger,
      setupSignUpTrigger,
      logSetupSignUpTriggerComplete,
      selectUser,
      checkIfUserAlreadyExists,
      createPassword,
      logSignUpExistingUser,
      signUpExistingUser,
      logSignUpExistingUserComplete,
      logLambdaLogs,
      logSignInUser,
      signInUser,
      logSignInUserComplete,
      confirmDeleteSignedInUser,
      deleteSignedInUser,
      logCleanUpReminder,
      logErrors,
    ],
    context,
  );
```

Questi sono passaggi condivisi con altri scenari.

```
import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { getCfnOutputs } from "@aws-doc-sdk-examples/lib/sdk/cfn-outputs.js";

export const skipWhenErrors = (state) => state.errors.length > 0;

export const getStackOutputs = new ScenarioAction(
  "getStackOutputs",
  async (state) => {
    if (!state.stackName || !state.stackRegion) {
      state.errors.push(
        new Error(
          "No stack name or region provided. The stack name and \
region are required to fetch CFN outputs relevant to this example.",
        ),
      );
      return;
    }

    const outputs = await getCfnOutputs(state.stackName, state.stackRegion);
    Object.assign(state, outputs);
  },
);

export const promptForStackName = new ScenarioInput(
  "stackName",
  "Enter the name of the stack you deployed earlier.",
  { type: "input", default: "PoolsAndTriggersStack" },
);

export const promptForStackRegion = new ScenarioInput(
  "stackRegion",
  "Enter the region of the stack you deployed earlier.",
  { type: "input", default: "us-east-1" },
);
```

```
export const logCleanUpReminder = new ScenarioOutput(
  "logCleanUpReminder",
  "All done. Remember to run 'cdk destroy' to teardown the stack.",
  { skipWhen: skipWhenErrors },
);
```

Un handler per il trigger PreSignUp con una funzione Lambda.

```
import type { PreSignUpTriggerEvent, Handler } from "aws-lambda";
import type { UserRepository } from "../user-repository";
import { DynamoDBUserRepository } from "../user-repository";

export class PreSignUpHandler {
  private userRepository: UserRepository;

  constructor(userRepository: UserRepository) {
    this.userRepository = userRepository;
  }

  private isPreSignUpTriggerSource(event: PreSignUpTriggerEvent): boolean {
    return event.triggerSource === "PreSignUp_SignUp";
  }

  private getEventUserEmail(event: PreSignUpTriggerEvent): string {
    return event.request.userAttributes.email;
  }

  async handlePreSignUpTriggerEvent(
    event: PreSignUpTriggerEvent,
  ): Promise<PreSignUpTriggerEvent> {
    console.log(
      `Received presignup from ${event.triggerSource} for user` +
      `${event.userName}` +
    );

    if (!this.isPreSignUpTriggerSource(event)) {
      return event;
    }

    const eventEmail = this.getEventUserEmail(event);
    console.log(`Looking up email ${eventEmail}`);
    const storedUserInfo =
```



```
    await this.userRepository.getUserInfoByEmail(eventEmail);

    if (!storedUserInfo) {
        console.log(
            `Email ${eventEmail} not found. Email verification is required.`
        );
        return event;
    }

    if (storedUserInfo.UserName !== event.userName) {
        console.log(
            `UserEmail ${eventEmail} found, but stored UserName
            '${storedUserInfo.UserName}' does not match supplied UserName
            '${event.userName}'. Verification is required.`
        );
    } else {
        console.log(
            `UserEmail ${eventEmail} found with matching UserName
            ${storedUserInfo.UserName}. User is confirmed.`
        );
        event.response.autoConfirmUser = true;
        event.response.autoVerifyEmail = true;
    }
    return event;
}

const createPreSignUpHandler = (): PreSignUpHandler => {
    const tableName = process.env.TABLE_NAME;
    if (!tableName) {
        throw new Error("TABLE_NAME environment variable is not set");
    }

    const userRepository = new DynamoDBUserRepository(tableName);
    return new PreSignUpHandler(userRepository);
};

export const handler: Handler = async (event: PreSignUpTriggerEvent) => {
    const preSignUpHandler = createPreSignUpHandler();
    return preSignUpHandler.handlePreSignUpTriggerEvent(event);
};
```

## Modulo di azioni CloudWatch Logs.

```
import {
  CloudWatchLogsClient,
  GetLogEventsCommand,
  OrderBy,
  paginateDescribeLogStreams,
} from "@aws-sdk/client-cloudwatch-logs";

/**
 * Get the latest log stream for a Lambda function.
 * @param {{ functionName: string, region: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").LogStream | null,
  unknown]>}
 */
export const getLatestLogStreamForLambda = async ({ functionName, region }) => {
  try {
    const logGroupName = `/aws/lambda/${functionName}`;
    const cwlClient = new CloudWatchLogsClient({ region });
    const paginator = paginateDescribeLogStreams(
      { client: cwlClient },
      {
        descending: true,
        limit: 1,
        orderBy: OrderBy.LastEventTime,
        logGroupName,
      },
    );

    for await (const page of paginator) {
      return [page.logStreams[0], null];
    }
  } catch (err) {
    return [null, err];
  }
};

/**
 * Get the log events for a Lambda function's log stream.
 * @param {{
 *   functionName: string,
 *   logStreamName: string,
 *   eventCount: number,

```

```

*   region: string
* }} config
* @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").OutputLogEvent[]
| null, unknown]>}
*/
export const getLogEvents = async ({
  functionName,
  logStreamName,
  eventCount,
  region,
}) => {
  try {
    const cwlClient = new CloudWatchLogsClient({ region });
    const logGroupName = `/aws/lambda/${functionName}`;
    const response = await cwlClient.send(
      new GetLogEventsCommand({
        logStreamName: logStreamName,
        limit: eventCount,
        logGroupName: logGroupName,
      }),
    );

    return [response.events, null];
  } catch (err) {
    return [null, err];
  }
};

```

Modulo di azioni Amazon Cognito.

```

import {
  AdminGetUserCommand,
  CognitoIdentityProviderClient,
  DeleteUserCommand,
  InitiateAuthCommand,
  SignUpCommand,
  UpdateUserPoolCommand,
} from "@aws-sdk/client-cognito-identity-provider";

/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool

```

```

* @param {{ region: string, userPoolId: string, handlerArn: string }} config
* @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").UpdateUserPoolCommandOutput | null, unknown]>}
*/
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
* Attempt to register a user to a user pool with a given username and password.
* @param {{
*   region: string,
*   userPoolClientId: string,
*   username: string,
*   email: string,
*   password: string
* }} config
* @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").SignUpCommandOutput | null, unknown]>}
*/
export const signUpUser = async ({
  region,
  userPoolClientId,
  username,

```

```

    email,
    password,
  }) => {
    try {
      const cognitoClient = new CognitoIdentityProviderClient({
        region,
      });

      const response = await cognitoClient.send(
        new SignUpCommand({
          ClientId: userPoolClientId,
          Username: username,
          Password: password,
          UserAttributes: [{ Name: "email", Value: email }],
        }),
      );
      return [response, null];
    } catch (err) {
      return [null, err];
    }
  };
};

/**
 * Sign in a user to Amazon Cognito using a username and password authentication
 * flow.
 * @param {{ region: string, clientId: string, username: string, password:
 * string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
 * provider").InitiateAuthCommandOutput | null, unknown]>}
 */
export const signIn = async ({ region, clientId, username, password }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new InitiateAuthCommand({
        AuthFlow: "USER_PASSWORD_AUTH",
        ClientId: clientId,
        AuthParameters: { USERNAME: username, PASSWORD: password },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
}

```

```
};

/**
 * Retrieve an existing user from a user pool.
 * @param {{ region: string, userPoolId: string, username: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").AdminGetUserCommandOutput | null, unknown]>}
 */
export const getUser = async ({ region, userPoolId, username }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new AdminGetUserCommand({
        UserPoolId: userPoolId,
        Username: username,
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

## Modulo di azioni DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

/**
 * Populate a DynamoDB table with provide items.
 * @param {{ region: string, tableName: string, items: Record<string,
unknown>[] }} config
 * @returns {Promise<[import("@aws-sdk/lib-dynamodb").BatchWriteCommandOutput |
null, unknown]>}
 */
export const populateTable = async ({ region, tableName, items }) => {
  try {
    const ddbClient = new DynamoDBClient({ region });
    const docClient = DynamoDBDocumentClient.from(ddbClient);
    const response = await docClient.send(
      new BatchWriteCommand({
        RequestItems: {
          [tableName]: items.map((item) => ({
            PutRequest: {
              Item: item,
            },
          })),
        },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per JavaScript .
  - [DeleteUser](#)
  - [InitiateAuth](#)

- [SignUp](#)
- [UpdateUserPool](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Esegui automaticamente la migrazione di utenti Amazon Cognito noti con una funzione Lambda utilizzando un AWS SDK

L'esempio di codice seguente mostra come migrare automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda.

- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `MigrateUser`.
- Accedi ad Amazon Cognito con un nome utente e un indirizzo e-mail non incluso nel pool di utenti.
- La funzione Lambda esegue la scansione di una tabella DynamoDB e migra automaticamente gli utenti noti al pool di utenti.
- Esegui il flusso della password dimenticata per reimpostare la password per l'utente migrato.
- Accedi come nuovo utente, quindi elimina le risorse.

Go

SDK per Go V2

### Note

C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
import (  
    "context"  
    "errors"  
    "fmt"
```



```

    "log"
    "strings"
    "user_pools_and_lambda_triggers/actions"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// MigrateUser separates the steps of this scenario into individual functions so
// that
// they are simpler to read and understand.
type MigrateUser struct {
    helper      IScenarioHelper
    questioner  demotools.IQuestioner
    resources    Resources
    cognitoActor *actions.CognitoActions
}

// NewMigrateUser constructs a new migrate user runner.
func NewMigrateUser(sdkConfig aws.Config, questioner demotools.IQuestioner,
    helper IScenarioHelper) MigrateUser {
    scenario := MigrateUser{
        helper:      helper,
        questioner:  questioner,
        resources:    Resources{},
        cognitoActor: &actions.CognitoActions{CognitoClient:
            cognitoidentityprovider.NewFromConfig(sdkConfig)},
    }
    scenario.resources.init(scenario.cognitoActor, questioner)
    return scenario
}

// AddMigrateUserTrigger adds a Lambda handler as an invocation target for the
// MigrateUser trigger.
func (runner *MigrateUser) AddMigrateUserTrigger(ctx context.Context, userPoolId
    string, functionArn string) {
    log.Printf("Let's add a Lambda function to handle the MigrateUser trigger from
    Cognito.\n" +
        "This trigger happens when an unknown user signs in, and lets your function
    take action before Cognito\n" +
        "rejects the user.\n\n")
    err := runner.cognitoActor.UpdateTriggers(

```

```

    ctx, userPoolId,
    actions.TriggerInfo{Trigger: actions.UserMigration, HandlerArn:
aws.String(functionArn)})
if err != nil {
    panic(err)
}
log.Printf("Lambda function %v added to user pool %v to handle the MigrateUser
trigger.\n",
    functionArn, userPoolId)

log.Println(strings.Repeat("-", 88))
}

// SignInUser adds a new user to the known users table and signs that user in to
Amazon Cognito.
func (runner *MigrateUser) SignInUser(ctx context.Context, usersTable string,
clientId string) (bool, actions.User) {
    log.Println("Let's sign in a user to your Cognito user pool. When the username
and email matches an entry in the\n" +
        "DynamoDB known users table, the email is automatically verified and the user
is migrated to the Cognito user pool.")

    user := actions.User{}
    user.UserName = runner.questioner.Ask("\nEnter a username:")
    user.UserEmail = runner.questioner.Ask("\nEnter an email that you own. This
email will be used to confirm user migration\n" +
        "during this example:")

    runner.helper.AddKnownUser(ctx, usersTable, user)

    var err error
    var resetRequired *types.PasswordResetRequiredException
    var authResult *types.AuthenticationResultType
    signedIn := false
    for !signedIn && resetRequired == nil {
        log.Printf("Signing in to Cognito as user '%v'. The expected result is a
PasswordResetRequiredException.\n\n", user.UserName)
        authResult, err = runner.cognitoActor.SignIn(ctx, clientId, user.UserName, "_")
        if err != nil {
            if errors.As(err, &resetRequired) {
                log.Printf("\nUser '%v' is not in the Cognito user pool but was found in the
DynamoDB known users table.\n"+
                    "User migration is started and a password reset is required.",
                    user.UserName)

```

```

    } else {
        panic(err)
    }
} else {
    log.Printf("User '%v' successfully signed in. This is unexpected and probably
means you have not\n"+
        "cleaned up a previous run of this scenario, so the user exist in the Cognito
user pool.\n"+
        "You can continue this example and select to clean up resources, or manually
remove\n"+
        "the user from your user pool and try again.", user.UserName)
    runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
*authResult.AccessToken)
    signedIn = true
}
}

log.Println(strings.Repeat("-", 88))
return resetRequired != nil, user
}

// ResetPassword starts a password recovery flow.
func (runner *MigrateUser) ResetPassword(ctx context.Context, clientId string,
user actions.User) {
    wantCode := runner.questioner.AskBool(fmt.Sprintf("In order to migrate the user
to Cognito, you must be able to receive a confirmation\n"+
        "code by email at %v. Do you want to send a code (y/n)?", user.UserEmail), "y")
    if !wantCode {
        log.Println("To complete this example and successfully migrate a user to
Cognito, you must enter an email\n" +
            "you own that can receive a confirmation code.")
        return
    }
    codeDelivery, err := runner.cognitoActor.ForgotPassword(ctx, clientId,
user.UserName)
    if err != nil {
        panic(err)
    }
    log.Printf("\nA confirmation code has been sent to %v.",
*codeDelivery.Destination)
    code := runner.questioner.Ask("Check your email and enter it here:")

    confirmed := false

```

```

password := runner.questioner.AskPassword("\nEnter a password that has at least
eight characters, uppercase, lowercase, numbers and symbols.\n"+
"(the password will not display as you type):", 8)
for !confirmed {
    log.Printf("\nConfirming password reset for user '%v'.\n", user.UserName)
    err = runner.cognitoActor.ConfirmForgotPassword(ctx, clientId, code,
user.UserName, password)
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            password = runner.questioner.AskPassword("\nEnter another password:", 8)
        } else {
            panic(err)
        }
    } else {
        confirmed = true
    }
}
log.Printf("User '%v' successfully confirmed and migrated.\n", user.UserName)
log.Println("Signing in with your username and password...")
authResult, err := runner.cognitoActor.SignIn(ctx, clientId, user.UserName,
password)
if err != nil {
    panic(err)
}
log.Printf("Successfully signed in. Your access token starts with: %v...\n",
(*authResult.AccessToken)[:10])
runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
*authResult.AccessToken)

log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *MigrateUser) Run(ctx context.Context, stackName string) {
    defer func() {
        if r := recover(); r != nil {
            log.Println("Something went wrong with the demo.")
            runner.resources.Cleanup(ctx)
        }
    }()

    log.Println(strings.Repeat("-", 88))
    log.Printf("Welcome\n")

```

```

log.Println(strings.Repeat("-", 88))

stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
if err != nil {
    panic(err)
}
runner.resources.userPoolId = stackOutputs["UserPoolId"]

runner.AddMigrateUserTrigger(ctx, stackOutputs["UserPoolId"],
stackOutputs["MigrateUserFunctionArn"])
runner.resources.triggers = append(runner.resources.triggers,
actions.UserMigration)
resetNeeded, user := runner.SignInUser(ctx, stackOutputs["TableName"],
stackOutputs["UserPoolClientId"])
if resetNeeded {
    runner.helper.ListRecentLogEvents(ctx, stackOutputs["MigrateUserFunction"])
    runner.ResetPassword(ctx, stackOutputs["UserPoolClientId"], user)
}

runner.resources.Cleanup(ctx)

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}

```

Gestisci il trigger MigrateUser con una funzione Lambda.

```

import (
    "context"
    "log"
    "os"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"

```

```

    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
)

const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
// format.
type UserInfo struct {
    UserName string `dynamodbav:"UserName"`
    UserEmail string `dynamodbav:"UserEmail"`
}

type handler struct {
    dynamoClient *dynamodb.Client
}

// HandleRequest handles the MigrateUser event by looking up a user in an Amazon
// DynamoDB table and
// specifying whether they should be migrated to the user pool.
func (h *handler) HandleRequest(ctx context.Context, event
events.CognitoEventUserPoolsMigrateUser)
(events.CognitoEventUserPoolsMigrateUser, error) {
    log.Printf("Received migrate trigger from %v for user '%v'",
event.TriggerSource, event.UserName)
    if event.TriggerSource != "UserMigration_Authentication" {
        return event, nil
    }
    tableName := os.Getenv(TABLE_NAME)
    user := UserInfo{
        UserName: event.UserName,
    }
    log.Printf("Looking up user '%v' in table %v.\n", user.UserName, tableName)
    filterEx := expression.Name("UserName").Equal(expression.Value(user.UserName))
    expr, err := expression.NewBuilder().WithFilter(filterEx).Build()
    if err != nil {
        log.Printf("Error building expression to query for user '%v'.\n",
user.UserName)
        return event, err
    }
    output, err := h.dynamoClient.Scan(ctx, &dynamodb.ScanInput{
        TableName:          aws.String(tableName),
        FilterExpression:    expr.Filter(),
        ExpressionAttributeNames: expr.Names(),
        ExpressionAttributeValues: expr.Values(),
    })

```

```

    })
    if err != nil {
        log.Printf("Error looking up user '%v'.\n", user.UserName)
        return event, err
    }
    if len(output.Items) == 0 {
        log.Printf("User '%v' not found, not migrating user.\n", user.UserName)
        return event, err
    }

    var users []UserInfo
    err = attributevalue.UnmarshalListOfMaps(output.Items, &users)
    if err != nil {
        log.Printf("Couldn't unmarshal DynamoDB items. Here's why: %v\n", err)
        return event, err
    }

    user = users[0]
    log.Printf("UserName '%v' found with email %v. User is migrated and must reset password.\n", user.UserName, user.UserEmail)
    event.CognitoEventUserPoolsMigrateUserResponse.UserAttributes =
    map[string]string{
        "email":          user.UserEmail,
        "email_verified": "true", // email_verified is required for the forgot password
        flow.
    }
    event.CognitoEventUserPoolsMigrateUserResponse.FinalUserStatus =
    "RESET_REQUIRED"
    event.CognitoEventUserPoolsMigrateUserResponse.MessageAction = "SUPPRESS"

    return event, err
}

func main() {
    ctx := context.Background()
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        log.Panicln(err)
    }
    h := handler{
        dynamoClient: dynamodb.NewFromConfig(sdkConfig),
    }
    lambda.Start(h.HandleRequest)
}

```

Crea una struttura che esegue operazioni comuni.

```
import (
    "context"
    "log"
    "strings"
    "time"
    "user_pools_and_lambda_triggers/actions"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudformation"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this
// example.
type IScenarioHelper interface {
    Pause(secs int)
    GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
        error)
    PopulateUserTable(ctx context.Context, tableName string)
    GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
    AddKnownUser(ctx context.Context, tableName string, user actions.User)
    ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
// example.
type ScenarioHelper struct {
    questioner  demotools.IQuestioner
    dynamoActor *actions.DynamoActions
    cfnActor     *actions.CloudFormationActions
    cwlActor     *actions.CloudWatchLogsActions
    isTestRun    bool
}

// NewScenarioHelper constructs a new scenario helper.
```



```

func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
ScenarioHelper {
    scenario := ScenarioHelper{
        questioner: questioner,
        dynamoActor: &actions.DynamoActions{DynamoClient:
dynamodb.NewFromConfig(sdkConfig)},
        cfnActor: &actions.CloudFormationActions{CfnClient:
cloudformation.NewFromConfig(sdkConfig)},
        cwlActor: &actions.CloudWatchLogsActions{CwlClient:
cloudwatchlogs.NewFromConfig(sdkConfig)},
    }
    return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
    if !helper.isTestRun {
        time.Sleep(time.Duration(secs) * time.Second)
    }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
string) (actions.StackOutputs, error) {
    return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
string) {
    log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
this example.\n", tableName)
    err := helper.dynamoActor.PopulateTable(ctx, tableName)
    if err != nil {
        panic(err)
    }
}

// GetKnownUsers gets the users from the known users table in a structured
format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
(actions.UserList, error) {
    knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)

```

```

    if err != nil {
        log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
            tableName, err)
    }
    return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
    user actions.User) {
    log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
        table...\n",
        user.UserName, user.UserEmail)
    err := helper.dynamoActor.AddUser(ctx, tableName, user)
    if err != nil {
        panic(err)
    }
}

// ListRecentLogEvents gets the most recent log stream and events for the
// specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
    functionName string) {
    log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
    helper.Pause(10)
    log.Println("Okay, let's check the logs to find what's happened recently with
        your Lambda function.")
    logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
    if err != nil {
        panic(err)
    }
    log.Printf("Getting some recent events from log stream %v\n",
        *logStream.LogStreamName)
    events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
        *logStream.LogStreamName, 10)
    if err != nil {
        panic(err)
    }
    for _, event := range events {
        log.Printf("\t%v", *event.Message)
    }
    log.Println(strings.Repeat("-", 88))
}

```

Crea una struttura che racchiude le azioni di Amazon Cognito.

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
// trigger.
type Trigger int

const (
    PreSignUp Trigger = iota
    UserMigration
    PostAuthentication
)

type TriggerInfo struct {
    Trigger    Trigger
    HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
// is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
    string, triggers ...TriggerInfo) error {
    output, err := actor.CognitoClient.DescribeUserPool(ctx,
        &cognitoidentityprovider.DescribeUserPoolInput{
```

```

    UserPoolId: aws.String(userPoolId),
  })
  if err != nil {
    log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
      userPoolId, err)
    return err
  }
  lambdaConfig := output.UserPool.LambdaConfig
  for _, trigger := range triggers {
    switch trigger.Trigger {
    case PreSignUp:
      lambdaConfig.PreSignUp = trigger.HandlerArn
    case UserMigration:
      lambdaConfig.UserMigration = trigger.HandlerArn
    case PostAuthentication:
      lambdaConfig.PostAuthentication = trigger.HandlerArn
    }
  }
  _, err = actor.CognitoClient.UpdateUserPool(ctx,
    &cognitoidentityprovider.UpdateUserPoolInput{
      UserPoolId:    aws.String(userPoolId),
      LambdaConfig: lambdaConfig,
    })
  if err != nil {
    log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
  }
  return err
}

// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
  string, password string, userEmail string) (bool, error) {
  confirmed := false
  output, err := actor.CognitoClient.SignUp(ctx,
    &cognitoidentityprovider.SignUpInput{
      ClientId: aws.String(clientId),
      Password: aws.String(password),
      Username: aws.String(userName),
      UserAttributes: []types.AttributeType{
        {Name: aws.String("email"), Value: aws.String(userEmail)},
      },
    })
  if err != nil {
    log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
    return false, err
  }
  return confirmed, nil
}

```

```

if err != nil {
    var invalidPassword *types.InvalidPasswordException
    if errors.As(err, &invalidPassword) {
        log.Println(*invalidPassword.Message)
    } else {
        log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
    }
} else {
    confirmed = output.UserConfirmed
}
return confirmed, err
}

// SignIn signs in a user to Amazon Cognito using a username and password
// authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
string, password string) (*types.AuthenticationResultType, error) {
    var authResult *types.AuthenticationResultType
    output, err := actor.CognitoClient.InitiateAuth(ctx,
    &cognitoidentityprovider.InitiateAuthInput{
        AuthFlow:      "USER_PASSWORD_AUTH",
        ClientId:      aws.String(clientId),
        AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
    })
    if err != nil {
        var resetRequired *types.PasswordResetRequiredException
        if errors.As(err, &resetRequired) {
            log.Println(*resetRequired.Message)
        } else {
            log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
        }
    } else {
        authResult = output.AuthenticationResult
    }
    return authResult, err
}

// ForgotPassword starts a password recovery flow for a user. This flow typically
// sends a confirmation code
// to the user's configured notification destination, such as email.

```

```

func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
    userName string) (*types.CodeDeliveryDetailsType, error) {
    output, err := actor.CognitoClient.ForgotPassword(ctx,
        &cognitoidentityprovider.ForgotPasswordInput{
            ClientId: aws.String(clientId),
            Username: aws.String(userName),
        })
    if err != nil {
        log.Printf("Couldn't start password reset for user '%v'. Here's why: %v\n",
            userName, err)
    }
    return output.CodeDeliveryDetails, err
}

// ConfirmForgotPassword confirms a user with a confirmation code and a new
// password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
    string, code string, userName string, password string) error {
    _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
        &cognitoidentityprovider.ConfirmForgotPasswordInput{
            ClientId:      aws.String(clientId),
            ConfirmationCode: aws.String(code),
            Password:      aws.String(password),
            Username:      aws.String(userName),
        })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
        }
    }
    return err
}

// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
    string) error {

```

```

_, err := actor.CognitoClient.DeleteUser(ctx,
&cognitoidentityprovider.DeleteUserInput{
    AccessToken: aws.String(userAccessToken),
})
if err != nil {
    log.Printf("Couldn't delete user. Here's why: %v\n", err)
}
return err
}

// AdminCreateUser uses administrator credentials to add a user to a user pool.
// This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
string, userName string, userEmail string) error {
_, err := actor.CognitoClient.AdminCreateUser(ctx,
&cognitoidentityprovider.AdminCreateUserInput{
    UserPoolId:      aws.String(userPoolId),
    Username:        aws.String(userName),
    MessageAction:    types.MessageActionTypeSuppress,
    UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
aws.String(userEmail)}}},
})
if err != nil {
    var userExists *types.UsernameExistsException
    if errors.As(err, &userExists) {
        log.Printf("User %v already exists in the user pool.", userName)
        err = nil
    } else {
        log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
    }
}
return err
}

// AdminSetUserPassword uses administrator credentials to set a password for a
// user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
string, userName string, password string) error {

```

```
_, err := actor.CognitoClient.AdminSetUserPassword(ctx,
&cognitoidentityprovider.AdminSetUserPasswordInput{
    Password:    aws.String(password),
    UserPoolId:  aws.String(userPoolId),
    Username:    aws.String(userName),
    Permanent:   true,
})
if err != nil {
    var invalidPassword *types.InvalidPasswordException
    if errors.As(err, &invalidPassword) {
        log.Println(*invalidPassword.Message)
    } else {
        log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
err)
    }
}
return err
}
```

Crea una struttura che racchiude le azioni di DynamoDB.

```
import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type DynamoActions struct {
    DynamoClient *dynamodb.Client
}

// User defines structured user data.
```



```

type User struct {
    UserName  string
    UserEmail string
    LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
    UserPoolId string
    ClientId   string
    Time       string
}

// UserList defines a list of users.
type UserList struct {
    Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
strings.
func (users *UserList) UserNameList() []string {
    names := make([]string, len(users.Users))
    for i := 0; i < len(users.Users); i++ {
        names[i] = users.Users[i].UserName
    }
    return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
error {
    var err error
    var item map[string]types.AttributeValue
    var writeReqs []types.WriteRequest
    for i := 1; i < 4; i++ {
        item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
        if err != nil {
            log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
err)
            return err
        }
        writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
&types.PutRequest{Item: item}})
    }
}

```

```

}
_, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
    RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
})
if err != nil {
    log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
        tableName, err)
}
return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
    error) {
    var userList UserList
    output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
        TableName: aws.String(tableName),
    })
    if err != nil {
        log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
            err)
    } else {
        err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
        if err != nil {
            log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
        }
    }
    return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user
    User) error {
    userItem, err := attributevalue.MarshalMap(user)
    if err != nil {
        log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
    }
    _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
        Item:      userItem,
        TableName: aws.String(tableName),
    })
    if err != nil {
        log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
    }
}

```

```
    return err
}
```

Crea una struttura che racchiuda le azioni di CloudWatch Logs.

```
import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
    CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
    functionName string) (types.LogStream, error) {
    var logStream types.LogStream
    logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
    output, err := actor.CwlClient.DescribeLogStreams(ctx,
        &cloudwatchlogs.DescribeLogStreamsInput{
            Descending:    aws.Bool(true),
            Limit:         aws.Int32(1),
            LogGroupName: aws.String(logGroupName),
            OrderBy:      types.OrderByLastEventTime,
        })
    if err != nil {
        log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
            logGroupName, err)
    } else {
        logStream = output.LogStreams[0]
    }
    return logStream, err
}
```

```
// GetLogEvents gets the most recent eventCount events from the specified log
// stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
string, logStreamName string, eventCount int32) (
[]types.OutputLogEvent, error) {
var events []types.OutputLogEvent
logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
output, err := actor.CwlClient.GetLogEvents(ctx,
&cloudwatchlogs.GetLogEventsInput{
    LogStreamName: aws.String(logStreamName),
    Limit:         aws.Int32(eventCount),
    LogGroupName:  aws.String(logGroupName),
})
if err != nil {
    log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
logStreamName, err)
} else {
    events = output.Events
}
return events, err
}
```

Crea una struttura che racchiuda le azioni. CloudFormation

```
import (
    "context"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
    CfnClient *cloudformation.Client
}
```

```
// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
// structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
string) StackOutputs {
    output, err := actor.CfnClient.DescribeStacks(ctx,
        &cloudformation.DescribeStacksInput{
            StackName: aws.String(stackName),
        })
    if err != nil || len(output.Stacks) == 0 {
        log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
            stackName, err)
    }
    stackOutputs := StackOutputs{}
    for _, out := range output.Stacks[0].Outputs {
        stackOutputs[*out.OutputKey] = *out.OutputValue
    }
    return stackOutputs
}
```

Eliminare le risorse.

```
import (
    "context"
    "log"
    "user_pools_and_lambda_triggers/actions"

    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
    userPoolId      string
    userAccessTokens []string
    triggers        []actions.Trigger

    cognitoActor *actions.CognitoActions
    questioner   demotools.IQuestioner
}
```

```

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
demotools.IQuestioner) {
    resources.userAccessTokens = []string{}
    resources.triggers = []actions.Trigger{}
    resources.cognitoActor = cognitoActor
    resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
    defer func() {
        if r := recover(); r != nil {
            log.Printf("Something went wrong during cleanup.\n%v\n", r)
            log.Println("Use the AWS Management Console to remove any remaining resources
\n" +
                "that were created for this scenario.")
        }
    }()

    wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
resources that were created "+
        "during this demo (y/n)?", "y")
    if wantDelete {
        for _, accessToken := range resources.userAccessTokens {
            err := resources.cognitoActor.DeleteUser(ctx, accessToken)
            if err != nil {
                log.Println("Couldn't delete user during cleanup.")
                panic(err)
            }
            log.Println("Deleted user.")
        }
        triggerList := make([]actions.TriggerInfo, len(resources.triggers))
        for i := 0; i < len(resources.triggers); i++ {
            triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
HandlerArn: nil}
        }
        err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
triggerList...)
        if err != nil {
            log.Println("Couldn't update Cognito triggers during cleanup.")
            panic(err)
        }
        log.Println("Removed Cognito triggers from user pool.")
    } else {

```

```
    log.Println("Be sure to remove resources when you're done with them to avoid\n    unexpected charges!")\n  }\n}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - [ConfirmForgotPassword](#)
  - [DeleteUser](#)
  - [ForgotPassword](#)
  - [InitiateAuth](#)
  - [SignUp](#)
  - [UpdateUserPool](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Crea un'API API Gateway REST per tenere traccia COVID-19 dei dati

Il seguente esempio di codice mostra come creare un'API REST che simula un sistema per tenere traccia dei casi giornalieri COVID-19 negli Stati Uniti d'America, utilizzando dati fittizi.

Python

### SDK per Python (Boto3)

Mostra come usare AWS Chalice con per AWS SDK per Python (Boto3) creare un'API REST serverless che utilizzi Amazon API Gateway e Amazon DynamoDB. AWS Lambda L'API REST simula un sistema che tiene traccia dei casi quotidiani COVID-19 negli Stati Uniti d'America, utilizzando dati fittizi. Scopri come:

- Usa AWS Chalice per definire i percorsi nelle funzioni Lambda che vengono chiamate per gestire le richieste REST che arrivano tramite API Gateway.
- Utilizza le funzioni Lambda per recuperare e archiviare i dati in una tabella DynamoDB per soddisfare le richieste REST.

- Definisci la struttura della tabella e le risorse dei ruoli di sicurezza in un AWS CloudFormation modello.
- Usa AWS Chalice e CloudFormation per impacchettare e distribuire tutte le risorse necessarie.
- Usa CloudFormation per ripulire tutte le risorse create.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Gateway API
- CloudFormation
- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Creazione di una REST API per la libreria di prestiti

L'esempio di codice seguente mostra come creare una libreria di prestiti in cui gli utenti possono prendere in prestito e restituire libri tramite una REST API supportata da un database Amazon Aurora.

Python

### SDK per Python (Boto3)

Mostra come utilizzare l' AWS SDK per Python (Boto3) API Amazon Relational Database Service (Amazon RDS) e AWS Chalice per creare un'API REST supportata da un database Amazon Aurora. Il servizio Web è completamente serverless e rappresenta una semplice libreria di prestiti in cui gli utenti possono prendere in prestito e restituire libri. Scopri come:

- Creare e gestire un cluster di database Aurora serverless.
- Utilizzarlo per gestire le credenziali AWS Secrets Manager del database.
- Implementare un livello di archiviazione di dati che utilizza Amazon RDS per spostare i dati dentro e fuori dal database.



- Usa AWS Chalice per distribuire un'API REST serverless su Amazon API Gateway e. AWS Lambda
- Utilizza il pacchetto Richieste per inviare le richieste al servizio Web.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. [GitHub](#)

Servizi utilizzati in questo esempio

- Gateway API
- Aurora
- Lambda
- Secrets Manager

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Creazione di un'applicazione di messaggistica con Step Functions

Il seguente esempio di codice mostra come creare un'applicazione di AWS Step Functions messaggistica che recupera i record dei messaggi da una tabella di database.

Python

SDK per Python (Boto3)

Mostra come usare AWS SDK per Python (Boto3) with per creare un'applicazione di messaggistica che AWS Step Functions recupera i record dei messaggi da una tabella Amazon DynamoDB e li invia con Amazon Simple Queue Service (Amazon SQS). La macchina a stati si integra con una AWS Lambda funzione per la scansione del database alla ricerca di messaggi non inviati.

- Crea una macchina a stati che recuperi e aggiorni i record di messaggi da una tabella Amazon DynamoDB.
- Aggiorna la definizione della macchina a stati per inviare messaggi anche ad Amazon Simple Queue Service (Amazon SQS).
- Avvia e arresta l'esecuzione della macchina a stati.

- Connettiti a Lambda, DynamoDB e Amazon SQS da una macchina a stati utilizzando le integrazioni di servizi.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su. [GitHub](#)

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Crea un'API REST con integrazione di funzioni proxy

L'esempio di codice seguente mostra come:

- Creare un ruolo IAM per l'esecuzione di Lambda
- Creare e distribuire una funzione Lambda
- Crea un'API REST
- Configurare l'integrazione del proxy Lambda
- Distribuzione e test dell'API
- Eseguire la pulizia delle risorse

### Bash

AWS CLI con lo script Bash

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel [repository dei tutorial sugli esempi di codice per gli sviluppatori](#).

```
#!/bin/bash

set -euo pipefail

# Simple API Gateway Lambda Integration Script
# This script creates a REST API with Lambda proxy integration

# Generate random identifiers
FUNCTION_NAME="GetStartedLambdaProxyIntegration-$(openssl rand -hex 4)"
ROLE_NAME="GetStartedLambdaBasicExecutionRole-$(openssl rand -hex 4)"
API_NAME="LambdaProxyAPI-$(openssl rand -hex 4)"

# Get AWS account info
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
REGION=$(aws configure get region || echo "us-east-1")

# Validate inputs
if [[ -z "$ACCOUNT_ID" ]] || [[ -z "$REGION" ]]; then
    echo "Error: Failed to retrieve AWS account information" >&2
    exit 1
fi

echo "Creating Lambda function code..."

# Create Lambda function code with input validation
cat > lambda_function.py << 'EOF'
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    try:
        logger.info("Received event: %s", json.dumps(event))

        greeter = 'World'

        # Safely retrieve greeter from query string parameters
        query_params = event.get('queryStringParameters') or {}
        if isinstance(query_params, dict) and 'greeter' in query_params:
            greeter_value = query_params.get('greeter')
            if isinstance(greeter_value, str) and greeter_value:
```

```

        greeter = greeter_value

    # Safely retrieve greeter from multi-value headers
    multi_headers = event.get('multiValueHeaders') or {}
    if isinstance(multi_headers, dict) and 'greeter' in multi_headers:
        greeter_list = multi_headers.get('greeter', [])
        if isinstance(greeter_list, list) and greeter_list:
            greeter = " and ".join(str(g) for g in greeter_list if g)

    # Safely retrieve greeter from headers
    headers = event.get('headers') or {}
    if isinstance(headers, dict) and 'greeter' in headers:
        greeter_value = headers.get('greeter')
        if isinstance(greeter_value, str) and greeter_value:
            greeter = greeter_value

    # Safely retrieve greeter from body
    body = event.get('body')
    if body and isinstance(body, str):
        try:
            body_dict = json.loads(body)
            if isinstance(body_dict, dict) and 'greeter' in body_dict:
                greeter_value = body_dict.get('greeter')
                if isinstance(greeter_value, str) and greeter_value:
                    greeter = greeter_value
        except (json.JSONDecodeError, ValueError) as e:
            logger.warning("Failed to parse body: %s", str(e))

    # Sanitize greeter to prevent injection
    greeter = greeter.replace("'", '\\\\"').replace('"', "\\\"")

    response = {
        "statusCode": 200,
        "headers": {
            "Content-Type": "application/json"
        },
        "body": json.dumps({"message": f"Hello, {greeter}!"})
    }

    logger.info("Response: %s", json.dumps(response))
    return response

except Exception as e:
    logger.error("Unexpected error: %s", str(e), exc_info=True)

```

```
        return {
            "statusCode": 500,
            "headers": {
                "Content-Type": "application/json"
            },
            "body": json.dumps({"error": "Internal server error"})
        }
EOF

# Create deployment package
zip -q function.zip lambda_function.py || {
    echo "Error: Failed to create function.zip" >&2
    exit 1
}

echo "Creating IAM role..."

# Create IAM trust policy
cat > trust-policy.json << 'EOF'
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
EOF

# Create IAM role with error handling
aws iam create-role \
    --role-name "$ROLE_NAME" \
    --assume-role-policy-document file://trust-policy.json \
    --description "Temporary role for Lambda execution" || {
    echo "Error: Failed to create IAM role" >&2
    exit 1
}

aws iam tag-role --role-name "$ROLE_NAME" --tags Key=project,Value=doc-smith
Key=tutorial,Value=apigateway-lambda-integration
```

```

# Attach execution policy
aws iam attach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" || {
    echo "Error: Failed to attach IAM policy" >&2
    exit 1
}

# Wait for role propagation
sleep 15

echo "Creating Lambda function..."

# Create Lambda function with Python 3.11 (more recent runtime)
aws lambda create-function \
    --function-name "$FUNCTION_NAME" \
    --runtime python3.11 \
    --role "arn:aws:iam::$ACCOUNT_ID:role/$ROLE_NAME" \
    --handler lambda_function.lambda_handler \
    --zip-file fileb://function.zip \
    --timeout 30 \
    --memory-size 128 \
    --environment "Variables={LOG_LEVEL=INFO}" \
    --tags project=doc-smith,tutorial=apigateway-lambda-integration || {
    echo "Error: Failed to create Lambda function" >&2
    exit 1
}

echo "Creating API Gateway..."

# Create REST API with minimum logging
API_RESPONSE=$(aws apigateway create-rest-api \
    --name "$API_NAME" \
    --endpoint-configuration types=REGIONAL \
    --description "API for Lambda proxy integration tutorial" \
    --tags project=doc-smith,tutorial=apigateway-lambda-integration \
    --output json)

API_ID=$(echo "$API_RESPONSE" | grep -o '"id": "[^"]*"' | head -1 | cut -d'"' -f4)

if [[ -z "$API_ID" ]]; then

```

```
    echo "Error: Failed to create API Gateway" >&2
    exit 1
fi

# Get root resource ID
ROOT_RESOURCE_ID=$(aws apigateway get-resources --rest-api-id "$API_ID" --query
'items[?path=='`/`'].id' --output text)

# Create helloworld resource
aws apigateway create-resource \
    --rest-api-id "$API_ID" \
    --parent-id "$ROOT_RESOURCE_ID" \
    --path-part helloworld || {
    echo "Error: Failed to create resource" >&2
    exit 1
}

# Get resource ID
RESOURCE_ID=$(aws apigateway get-resources --rest-api-id "$API_ID" --query
"items[?pathPart=='helloworld'].id" --output text)

# Create ANY method with no authorization (intentional for tutorial)
aws apigateway put-method \
    --rest-api-id "$API_ID" \
    --resource-id "$RESOURCE_ID" \
    --http-method ANY \
    --authorization-type NONE || {
    echo "Error: Failed to create method" >&2
    exit 1
}

# Set up Lambda proxy integration
LAMBDA_URI="arn:aws:apigateway:$REGION:lambda:path/2015-03-31/functions/
arn:aws:lambda:$REGION:$ACCOUNT_ID:function:$FUNCTION_NAME/invocations"

aws apigateway put-integration \
    --rest-api-id "$API_ID" \
    --resource-id "$RESOURCE_ID" \
    --http-method ANY \
    --type AWS_PROXY \
    --integration-http-method POST \
    --uri "$LAMBDA_URI" || {
    echo "Error: Failed to create integration" >&2
    exit 1
}
```

```
}

# Grant API Gateway permission to invoke Lambda
STATEMENT_ID="apigateway-invoke-$(openssl rand -hex 4)"
SOURCE_ARN="arn:aws:execute-api:$REGION:$ACCOUNT_ID:$API_ID/*/*"

aws lambda add-permission \
    --function-name "$FUNCTION_NAME" \
    --statement-id "$STATEMENT_ID" \
    --action lambda:InvokeFunction \
    --principal apigateway.amazonaws.com \
    --source-arn "$SOURCE_ARN" || {
    echo "Error: Failed to add Lambda permission" >&2
    exit 1
}

# Deploy API
aws apigateway create-deployment \
    --rest-api-id "$API_ID" \
    --stage-name test \
    --description "Test deployment" || {
    echo "Error: Failed to deploy API" >&2
    exit 1
}

echo "Testing API..."

# Test the API
INVOKE_URL="https://$API_ID.execute-api.$REGION.amazonaws.com/test/helloworld"

echo "API URL: $INVOKE_URL"

# Test with query parameter (with proper URL encoding)
echo "Testing with query parameter:"
curl -s -X GET "$INVOKE_URL?greeter=John" | jq . 2>/dev/null || curl -s -X GET
"$INVOKE_URL?greeter=John"
echo ""

# Test with header
echo "Testing with header:"
curl -s -X GET "$INVOKE_URL" \
    -H 'content-type: application/json' \
    -H 'greeter: John' | jq . 2>/dev/null || curl -s -X GET "$INVOKE_URL" \
    -H 'content-type: application/json' \
```



```

    -H 'greeter: John'
echo ""

# Test with body
echo "Testing with POST body:"
curl -s -X POST "$INVOKE_URL" \
    -H 'content-type: application/json' \
    -d '{"greeter": "John"}' | jq . 2>/dev/null || curl -s -X POST "$INVOKE_URL" \
    -H 'content-type: application/json' \
    -d '{"greeter": "John"}'
echo ""

echo "Tutorial completed! API is available at: $INVOKE_URL"

# Cleanup
echo "Cleaning up resources..."

# Delete API
aws apigateway delete-rest-api --rest-api-id "$API_ID" || echo "Warning: Failed
to delete API" >&2

# Delete Lambda function
aws lambda delete-function --function-name "$FUNCTION_NAME" || echo "Warning:
Failed to delete Lambda function" >&2

# Detach policy and delete role
aws iam detach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" || echo "Warning: Failed to detach policy" >&2

aws iam delete-role --role-name "$ROLE_NAME" || echo "Warning: Failed to delete
role" >&2

# Clean up local files securely
rm -f lambda_function.py function.zip trust-policy.json

echo "Cleanup completed!"

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dei comandi della AWS CLI .
  - [AddPermission](#)

- [AttachRolePolicy](#)
- [CreateDeployment](#)
- [CreateFunction](#)
- [CreateResource](#)
- [CreateRestApi](#)
- [CreateRole](#)
- [DeleteFunction](#)
- [DeleteRestApi](#)
- [DeleteRole](#)
- [DetachRolePolicy](#)
- [GetCallerIdentity](#)
- [GetResources](#)
- [GetRestApis](#)
- [PutIntegration](#)
- [PutMethod](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette

L'esempio di codice seguente mostra come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

.NET

SDK per .NET

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su [GitHub](#).

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

## C++

### SDK per C++

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

## Java

### SDK per Java 2.x

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

## JavaScript

### SDK per JavaScript (v3)

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. [GitHub](#)

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB

- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

## Kotlin

### SDK per Kotlin

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

### Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

## PHP

### SDK per PHP

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

## Rust

### SDK per Rust

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Per approfondire l'origine di questo esempio, consulta il post su [AWS Community](#).

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Creazione di un'applicazione di chat websocket con API Gateway

L'esempio di codice seguente mostra come creare un'applicazione chat servita da un'API websocket creata su Gateway Amazon API.

### Python

#### SDK per Python (Boto3)

Mostra come utilizzarlo AWS SDK per Python (Boto3) con Amazon API Gateway V2 per creare un'API websocket che si integri con Amazon AWS Lambda DynamoDB.

- Crea un'API WebSocket servita da Gateway API.
- Definisci un gestore Lambda che memorizzi le connessioni in DynamoDB e invii messaggi ad altri partecipanti alla chat.
- Connettiti all'applicazione di chat websocket e invia messaggi con il pacchetto Websockets.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. [GitHub](#)

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Crea un'applicazione che analizza il feedback dei clienti e sintetizza l'audio

Il seguente esempio di codice spiega come creare un'applicazione che analizza schede dei commenti dei clienti, le traduce dalla loro lingua originale, ne determina la valutazione e genera un file audio dal testo tradotto.

## .NET

### SDK per .NET

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.
- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

## Java

### SDK per Java 2.x

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.



- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in [GitHub](#).

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

## JavaScript

### SDK per JavaScript (v3)

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.
- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in [GitHub](#). I seguenti estratti mostrano come AWS SDK per JavaScript viene utilizzato all'interno delle funzioni Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
```

```

* Determine the language and sentiment of the extracted text.
*
* @param {{ source_text: string }} extractTextOutput
*/
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};

```

```

import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
* Fetch the S3 object from the event and analyze it using Amazon Textract.
*
* @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
eventBridgeS3Event
*/

```

```

export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
  // textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};

```

```

import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string }}
 * sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
  });

```

```
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string }}
  textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });
```

```
const { TranslatedText } = await translateClient.send(translateCommand);

return { translated_text: TranslatedText };
};
```

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

## Ruby

### SDK per Ruby

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.
- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in [GitHub](#)

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Creazione di una dashboard di monitoraggio con il nome della funzione come variabile

L'esempio di codice seguente mostra come:

- Crea un pannello di controllo CloudWatch
- Aggiungi widget metrici Lambda con una variabile con nome di funzione
- Verifica la dashboard
- Eseguire la pulizia delle risorse

### Bash

#### AWS CLI con lo script Bash

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel [repository dei tutorial sugli esempi di codice per gli sviluppatori](#).

```
#!/bin/bash

# Script to create a CloudWatch dashboard with Lambda function name as a variable
# This script creates a CloudWatch dashboard that allows you to switch between
# different Lambda functions

# Set up logging with secure permissions
LOG_FILE="${HOME}/.cloudwatch-dashboard-script.log"
touch "$LOG_FILE" && chmod 600 "$LOG_FILE"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "$(date): Starting CloudWatch dashboard creation script"

# Security: Set strict error handling
```

```

set -uo pipefail
trap 'handle_error "Script failed at line $LINENO"' ERR

# Function to handle errors
handle_error() {
    local error_msg="${1:-Unknown error}"
    echo "ERROR: $error_msg" >&2
    echo "Resources created:"
    echo "- CloudWatch Dashboard: LambdaMetricsDashboard"
    echo ""
    echo "======"
    echo "CLEANUP CONFIRMATION"
    echo "======"
    echo "An error occurred. Proceeding with automatic cleanup..."

    echo "Cleaning up resources..."
    aws cloudwatch delete-dashboards --dashboard-names LambdaMetricsDashboard 2>/
    dev/null || true

    # Clean up temporary files
    if [ -n "${TEMP_DIR:-}" ] && [ -d "$TEMP_DIR" ]; then
        rm -rf "$TEMP_DIR"
    fi
    rm -f dashboard-body.json

    echo "Cleanup complete."
    exit 1
}

# Security: Validate AWS CLI is installed
if ! command -v aws &> /dev/null; then
    handle_error "AWS CLI is not installed. Please install it and try again."
fi

# Check if AWS CLI is installed and configured
echo "Checking AWS CLI configuration..."
if ! aws sts get-caller-identity > /dev/null 2>&1; then
    handle_error "AWS CLI is not properly configured. Please configure it with
'aws configure' and try again."
fi

# Get the current region securely
REGION=$(aws configure get region 2>/dev/null || echo "")
if [ -z "$REGION" ]; then

```

```

    REGION="us-east-1"
    echo "No region found in AWS config, defaulting to $REGION"
fi
echo "Using region: $REGION"

# Validate region format
if ! [[ "$REGION" =~ ^[a-z]{2}-[a-z]+-[0-9]{1}$ ]]; then
    handle_error "Invalid AWS region format: $REGION"
fi

# Check if there are any Lambda functions in the account
echo "Checking for Lambda functions..."
LAMBDA_FUNCTIONS=$(aws lambda list-functions --region "$REGION" --query
    "Functions[*].FunctionName" --output text 2>/dev/null || echo "")

if [ -z "$LAMBDA_FUNCTIONS" ]; then
    echo "No Lambda functions found in your account. Creating a simple test
    function..."

    # Create a temporary directory for Lambda function code with secure
    permissions
    TEMP_DIR=$(mktemp -d)
    chmod 700 "$TEMP_DIR"
    trap 'rm -rf "$TEMP_DIR"' EXIT

    # Create a simple Lambda function
    cat > "$TEMP_DIR/index.js" << 'EOF'
exports.handler = async (event) => {
    console.log('Event:', JSON.stringify(event, null, 2));
    return {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
};
EOF

    # Zip the function code
    if ! cd "$TEMP_DIR"; then
        handle_error "Failed to change to temporary directory"
    fi

    if ! zip -q function.zip index.js; then
        handle_error "Failed to create zip file"
    fi

```



```

# Create a role for the Lambda function with restricted trust policy
ROLE_NAME="LambdaDashboardTestRole-$(date +%s)"
TRUST_POLICY='{ "Version": "2012-10-17",          "Statement":
[{"Effect": "Allow", "Principal":
{"Service": "lambda.amazonaws.com"}, "Action": "sts:AssumeRole"}]}'

if ! ROLE_ARN=$(aws iam create-role \
    --role-name "$ROLE_NAME" \
    --assume-role-policy-document "$TRUST_POLICY" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=cloudwatch-
dynamicdash \
    --query "Role.Arn" \
    --output text 2>/dev/null); then
    handle_error "Failed to create IAM role for Lambda function"
fi

echo "Waiting for role to be available..."
sleep 10

# Attach basic Lambda execution policy
if ! aws iam attach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"; then
    aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    handle_error "Failed to attach policy to IAM role"
fi

# Create the Lambda function
FUNCTION_NAME="DashboardTestFunction-$(date +%s)"
if ! aws lambda create-function \
    --function-name "$FUNCTION_NAME" \
    --runtime nodejs18.x \
    --role "$ROLE_ARN" \
    --handler index.handler \
    --zip-file fileb://function.zip \
    --tags project=doc-smith,tutorial=cloudwatch-dynamicdash \
    --region "$REGION" > /dev/null 2>&1; then
    aws iam detach-role-policy \
        --role-name "$ROLE_NAME" \
        --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
    aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true

```

```

        handle_error "Failed to create Lambda function"
    fi

    # Invoke the function to generate some metrics
    echo "Invoking Lambda function to generate metrics..."
    for i in {1..5}; do
        aws lambda invoke --function-name "$FUNCTION_NAME" --payload '{}' /dev/
null --region "$REGION" > /dev/null 2>&1 || true
        sleep 1
    done

    # Go back to original directory
    cd - > /dev/null

    # Set the function name for the dashboard
    DEFAULT_FUNCTION="$FUNCTION_NAME"
else
    # Use the first Lambda function as default
    DEFAULT_FUNCTION=$(echo "$LAMBDA_FUNCTIONS" | awk '{print $1}')
    echo "Found Lambda functions. Using $DEFAULT_FUNCTION as default."
    FUNCTION_NAME=""
    ROLE_NAME=""
fi

# Create a dashboard with Lambda metrics and a function name variable
echo "Creating CloudWatch dashboard with Lambda function name variable..."

# Create a JSON file for the dashboard body with secure permissions
DASHBOARD_JSON="dashboard-body-$.json"
touch "$DASHBOARD_JSON" && chmod 600 "$DASHBOARD_JSON"

# Escape special characters in region and function name for JSON
REGION_ESCAPED=$(printf '%s\n' "$REGION" | sed 's:[\/&]:\\&:g')
FUNCTION_ESCAPED=$(printf '%s\n' "$DEFAULT_FUNCTION" | sed 's:[\/&]:\\&:g')

cat > "$DASHBOARD_JSON" << EOF
{
    "widgets": [
        {
            "type": "metric",
            "x": 0,
            "y": 0,
            "width": 12,
            "height": 6,

```

```

    "properties": {
      "metrics": [
        [ "AWS/Lambda", "Invocations", "FunctionName", "\${FunctionName}" ],
        [ ".", "Errors", ".", "." ],
        [ ".", "Throttles", ".", "." ]
      ],
      "view": "timeSeries",
      "stacked": false,
      "region": "$REGION_ESCAPED",
      "title": "Lambda Function Metrics for \${FunctionName}",
      "period": 300
    }
  },
  {
    "type": "metric",
    "x": 0,
    "y": 6,
    "width": 12,
    "height": 6,
    "properties": {
      "metrics": [
        [ "AWS/Lambda", "Duration", "FunctionName", "\${FunctionName}",
{ "stat": "Average" } ]
      ],
      "view": "timeSeries",
      "stacked": false,
      "region": "$REGION_ESCAPED",
      "title": "Duration for \${FunctionName}",
      "period": 300
    }
  },
  {
    "type": "metric",
    "x": 12,
    "y": 0,
    "width": 12,
    "height": 6,
    "properties": {
      "metrics": [
        [ "AWS/Lambda", "ConcurrentExecutions", "FunctionName", "\${FunctionName}" ]
      ],
      "view": "timeSeries",
      "stacked": false,

```

```

        "region": "$REGION_ESCAPED",
        "title": "Concurrent Executions for \${FunctionName}",
        "period": 300
    }
}
],
"periodOverride": "auto",
"variables": [
    {
        "type": "property",
        "id": "FunctionName",
        "property": "FunctionName",
        "label": "Lambda Function",
        "inputType": "select",
        "values": [
            {
                "value": "$FUNCTION_ESCAPED",
                "label": "$FUNCTION_ESCAPED"
            }
        ]
    }
]
}
]
}
EOF

# Validate JSON before sending
if ! jq empty "$DASHBOARD_JSON" 2>/dev/null; then
    handle_error "Invalid JSON generated for dashboard"
fi

# Create the dashboard using the JSON file
DASHBOARD_NAME="LambdaMetricsDashboard-$(date +%s)"
if ! DASHBOARD_RESULT=$(aws cloudwatch put-dashboard \
    --dashboard-name "$DASHBOARD_NAME" \
    --dashboard-body file://"$DASHBOARD_JSON" \
    --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
"$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
    fi
fi

```

```

        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to create CloudWatch dashboard."
fi

# Display any validation messages but continue
if echo "$DASHBOARD_RESULT" | grep -q "DashboardValidationMessages"; then
    echo "Dashboard created with validation messages:"
    echo "$DASHBOARD_RESULT"
    echo "These validation messages are warnings and the dashboard should still
function."
else
    echo "Dashboard created successfully!"
fi

# Verify the dashboard was created
echo "Verifying dashboard creation..."
if ! DASHBOARD_INFO=$(aws cloudwatch get-dashboard --dashboard-name
"$DASHBOARD_NAME" --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
"$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to verify dashboard creation."
fi

echo "Dashboard verification successful!"
echo "Dashboard details:"
echo "$DASHBOARD_INFO"

# List all dashboards to confirm
echo "Listing all dashboards:"
if ! DASHBOARDS=$(aws cloudwatch list-dashboards --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
"$REGION" 2>/dev/null || true
        aws iam detach-role-policy \

```

```

        --role-name "$ROLE_NAME" \
        --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to list dashboards."
fi
echo "$DASHBOARDS"

# Show instructions for accessing the dashboard
echo ""
echo "Dashboard created successfully! To access it:"
echo "1. Open the CloudWatch console at https://console.aws.amazon.com/
cloudwatch/"
echo "2. In the navigation pane, choose Dashboards"
echo "3. Select $DASHBOARD_NAME"
echo "4. You should see a dropdown menu labeled 'Lambda Function' at the top of
the dashboard"
echo "5. Use this dropdown to select different Lambda functions and see their
metrics"
echo ""

# Create a list of resources for cleanup
RESOURCES=(- CloudWatch Dashboard: $DASHBOARD_NAME)
if [ -n "${FUNCTION_NAME:-}" ]; then
    RESOURCES+=(- Lambda Function: $FUNCTION_NAME)
    RESOURCES+=(- IAM Role: $ROLE_NAME)
fi

# Prompt for cleanup with automatic yes
echo "======"
echo "CLEANUP CONFIRMATION"
echo "======"
echo "Resources created:"
for resource in "${RESOURCES[@]}"; do
    echo "$resource"
done
echo ""
echo "Proceeding with automatic cleanup..."

CLEANUP_CHOICE="y"

if [[ "${CLEANUP_CHOICE,,}" == "y" ]]; then
    echo "Cleaning up resources..."

```

```
# Delete the dashboard
if aws cloudwatch delete-dashboards --dashboard-names "$DASHBOARD_NAME" --
region "$REGION" 2>/dev/null; then
    echo "Dashboard deleted successfully."
else
    echo "WARNING: Failed to delete dashboard. You may need to delete it
manually."
fi

# If we created a Lambda function, delete it and its role
if [ -n "${FUNCTION_NAME:-}" ]; then
    echo "Deleting Lambda function..."
    if aws lambda delete-function --function-name "$FUNCTION_NAME" --region
"$REGION" 2>/dev/null; then
        echo "Lambda function deleted successfully."
    else
        echo "WARNING: Failed to delete Lambda function. You may need to
delete it manually."
    fi

    echo "Detaching role policy..."
    if aws iam detach-role-policy \
        --role-name "$ROLE_NAME" \
        --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null; then
        echo "Role policy detached successfully."
    else
        echo "WARNING: Failed to detach role policy. You may need to detach
it manually."
    fi

    echo "Deleting IAM role..."
    if aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null; then
        echo "IAM role deleted successfully."
    else
        echo "WARNING: Failed to delete IAM role. You may need to delete it
manually."
    fi
fi

# Clean up the JSON file
rm -f "$DASHBOARD_JSON"
```

```
    echo "Cleanup complete."
fi

echo "Script completed successfully!"
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dei comandi della AWS CLI .
  - [AttachRolePolicy](#)
  - [CreateFunction](#)
  - [CreateRole](#)
  - [DeleteDashboards](#)
  - [DeleteFunction](#)
  - [DeleteRole](#)
  - [DetachRolePolicy](#)
  - [GetCallerIdentity](#)
  - [GetDashboard](#)
  - [Invoke](#)
  - [ListDashboards](#)
  - [ListFunctions](#)
  - [PutDashboard](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Creazione della tua prima funzione serverless

L'esempio di codice seguente mostra come:

- Crea un ruolo IAM per Lambda
- Crea codice funzionale
- Creazione di una funzione Lambda
- Testa la tua funzione Lambda
- Eseguire la pulizia delle risorse



## Bash

## AWS CLI con lo script Bash

 Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel [repository dei tutorial sugli esempi di codice per gli sviluppatori](#).

```
#!/bin/bash
# AWS Lambda - Create Your First Function
# This script creates a Lambda function, invokes it with a test event,
# views CloudWatch logs, and cleans up all resources.
#
# Source: https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html
#
# Resources created:
#   - IAM role (Lambda execution role with basic logging permissions)
#   - Lambda function (Python 3.13 or Node.js 22.x runtime)
#   - CloudWatch log group (created automatically by Lambda on invocation)

set -eE -o pipefail

#####
# Setup
#####

UNIQUE_ID=$(head -c 8 /dev/urandom | od -An -tx1 | tr -d ' ')
FUNCTION_NAME="my-lambda-function-${UNIQUE_ID}"
ROLE_NAME="lambda-execution-role-${UNIQUE_ID}"
LOG_GROUP_NAME="/aws/lambda/${FUNCTION_NAME}"

TEMP_DIR=$(mktemp -d)
readonly TEMP_DIR
LOG_FILE="${TEMP_DIR}/lambda-gettingstarted.log"

exec >>(tee -a "$LOG_FILE") 2>&1

declare -a CREATED_RESOURCES
```

```
#####
# Helper functions
#####

cleanup_resources() {
    # Disable error trap to prevent recursion during cleanup
    trap - ERR
    set +eE

    echo ""
    echo "Cleaning up resources..."
    echo ""

    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        local RESOURCE="${CREATED_RESOURCES[$i]}"
        local TYPE="${RESOURCE%%:*}"
        local NAME="${RESOURCE#*:}"

        case "$TYPE" in
            log-group)
                echo "Deleting CloudWatch log group: ${NAME}"
                aws logs delete-log-group \
                    --log-group-name "$NAME" 2>&1 || echo " WARNING: Could not
delete log group ${NAME}."
                ;;
            lambda-function)
                echo "Deleting Lambda function: ${NAME}"
                aws lambda delete-function \
                    --function-name "$NAME" 2>&1 || echo " WARNING: Could not
delete Lambda function ${NAME}."
                echo " Waiting for function deletion to complete..."
                local DELETE_WAIT=0
                while aws lambda get-function --function-name "$NAME" > /dev/null
2>&1; do
                    sleep 2
                    DELETE_WAIT=$((DELETE_WAIT + 2))
                    if [ "$DELETE_WAIT" -ge 60 ]; then
                        echo " WARNING: Timed out waiting for function
deletion."
                        break
                    fi
                done
                ;;
            iam-role-policy)

```

```

        local ROLE_PART="${NAME%%|*}"
        local POLICY_PART="${NAME#*|}"
        echo "Detaching policy from role: ${ROLE_PART}"
        aws iam detach-role-policy \
            --role-name "$ROLE_PART" \
            --policy-arn "$POLICY_PART" 2>&1 || echo " WARNING: Could
not detach policy from role ${ROLE_PART}."
        ;;
    iam-role)
        echo "Deleting IAM role: ${NAME}"
        aws iam delete-role \
            --role-name "$NAME" 2>&1 || echo " WARNING: Could not delete
IAM role ${NAME}."
        ;;
    esac
done

if [ -d "$TEMP_DIR" ]; then
    rm -rf "$TEMP_DIR"
fi

echo ""
echo "Cleanup complete."
}

handle_error() {
    echo ""
    echo "======"
    echo "ERROR: Script failed at $1"
    echo "======"
    echo ""
    if [ ${#CREATED_RESOURCES[@]} -gt 0 ]; then
        echo "Attempting to clean up ${#CREATED_RESOURCES[@]} resource(s)..."
        cleanup_resources
    fi
    exit 1
}

trap 'handle_error "line $LINENO"' ERR

wait_for_resource() {
    local DESCRIPTION="$1"
    local COMMAND="$2"
    local TARGET_VALUE="$3"

```

```

    local TIMEOUT=300
    local ELAPSED=0
    local INTERVAL=5

    echo "Waiting for ${DESCRIPTION}..."
    while true; do
        local RESULT
        RESULT=$(eval "$COMMAND" 2>&1) || true
        if echo "$RESULT" | grep -q "$TARGET_VALUE"; then
            echo "  ${DESCRIPTION} is ready."
            return 0
        fi
        if [ "$ELAPSED" -ge "$TIMEOUT" ]; then
            echo "ERROR: Timed out waiting for ${DESCRIPTION} after ${TIMEOUT}
seconds."
            return 1
        fi
        sleep "$INTERVAL"
        ELAPSED=$((ELAPSED + INTERVAL))
    done
}

validate_input() {
    local input="$1"
    local pattern="$2"
    if ! [[ "$input" =~ $pattern ]]; then
        echo "ERROR: Invalid input: $input"
        return 1
    fi
    return 0
}

#####
# Region pre-check
#####

CONFIGURED_REGION=$(aws configure get region 2>/dev/null || true)
if [ -z "$CONFIGURED_REGION" ] && [ -z "$AWS_DEFAULT_REGION" ] && [ -z
"$AWS_REGION" ]; then
    echo "ERROR: No AWS region configured."
    echo "Run 'aws configure set region <region>' or export AWS_DEFAULT_REGION."
    exit 1
fi

```

```
#####
# Runtime selection
#####

echo ""
echo "=====
echo "AWS Lambda - Create Your First Function"
echo "=====
echo ""
echo "Select a runtime for your Lambda function:"
echo "  1) Python 3.13"
echo "  2) Node.js 22.x"
echo ""
echo "Using default: Python 3.13"
RUNTIME_CHOICE="1"

case "$RUNTIME_CHOICE" in
  1)
    RUNTIME="python3.13"
    HANDLER="lambda_function.lambda_handler"
    CODE_FILE="lambda_function.py"
    cat > "${TEMP_DIR}/${CODE_FILE}" << 'PYTHON_EOF'
import json
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def lambda_handler(event, context):
    if not isinstance(event, dict) or 'length' not in event or 'width' not in event:
        raise ValueError('Event must contain length and width')
    try:
        length = float(event['length'])
        width = float(event['width'])
        if length < 0 or width < 0:
            raise ValueError('Length and width must be non-negative')
        area = calculate_area(length, width)
        print(f'The area is {area}')
        logger.info(f'CloudWatch logs group: {context.log_group_name}')
        return json.dumps({'area': area})
    except (TypeError, ValueError) as e:
        logger.error(f'Error processing input: {str(e)}')
        raise
def calculate_area(length, width):
    return length * width
```

```

PYTHON_EOF
    echo "Selected runtime: Python 3.13"
    ;;
2)
    RUNTIME="nodejs22.x"
    HANDLER="index.handler"
    CODE_FILE="index.mjs"
    cat > "${TEMP_DIR}/${CODE_FILE}" << 'NODEJS_EOF'
export const handler = async (event, context) => {
    if (!event || typeof event.length !== 'number' || typeof event.width !==
'number') {
        throw new Error('Event must contain numeric length and width');
    }
    if (event.length < 0 || event.width < 0) {
        throw new Error('Length and width must be non-negative');
    }
    const area = event.length * event.width;
    console.log(`The area is ${area}`);
    console.log('CloudWatch log group: ', context.logGroupName);
    return JSON.stringify({area});
};
NODEJS_EOF
    echo "Selected runtime: Node.js 22.x"
    ;;
*)
    echo "ERROR: Invalid choice. Please enter 1 or 2."
    exit 1
    ;;
esac

#####
# Step 1: Create IAM execution role
#####

echo ""
echo "=====
echo "Step 1: Create IAM execution role"
echo "=====
echo ""

TRUST_POLICY='{
    "Version":"2012-10-17",
    "Statement": [
        {

```

```

        "Effect": "Allow",
        "Principal": {
            "Service": "lambda.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }
]
}'

echo "Creating IAM role: ${ROLE_NAME}"
ROLE_OUTPUT=$(aws iam create-role \
    --role-name "${ROLE_NAME}" \
    --assume-role-policy-document "$TRUST_POLICY" \
    --query 'Role.Arn' \
    --output text 2>&1)

if ! validate_input "$ROLE_OUTPUT" "^arn:aws:iam::[0-9]+:role/"; then
    echo "ERROR: Failed to create IAM role"
    exit 1
fi

echo "$ROLE_OUTPUT"
ROLE_ARN="$ROLE_OUTPUT"
CREATED_RESOURCES+=("iam-role:${ROLE_NAME}")
echo "Role ARN: ${ROLE_ARN}"

aws iam tag-role \
    --role-name "${ROLE_NAME}" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=lambda-gettingstarted

echo ""
echo "Attaching AWSLambdaBasicExecutionRole policy..."
aws iam attach-role-policy \
    --role-name "${ROLE_NAME}" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>&1
CREATED_RESOURCES+=("iam-role-policy:${ROLE_NAME}|arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole")
echo "Policy attached."

# IAM roles can take a few seconds to propagate
echo "Waiting for IAM role to propagate..."
sleep 10

```

```
#####
# Step 2: Create Lambda function
#####

echo ""
echo "=====
echo "Step 2: Create Lambda function"
echo "=====
echo ""

echo "Creating deployment package..."
ORIGINAL_DIR=$(pwd)
cd "$TEMP_DIR" || exit 1
zip -j function.zip "$CODE_FILE" > /dev/null 2>&1 || {
    echo "ERROR: Failed to create deployment package"
    exit 1
}
cd "$ORIGINAL_DIR" || exit 1

if [ ! -f "${TEMP_DIR}/function.zip" ]; then
    echo "ERROR: Deployment package creation failed"
    exit 1
fi

echo "Creating Lambda function: ${FUNCTION_NAME}"
echo "  Runtime: ${RUNTIME}"
echo "  Handler: ${HANDLER}"
echo ""

CREATE_OUTPUT=$(aws lambda create-function \
    --function-name "$FUNCTION_NAME" \
    --runtime "$RUNTIME" \
    --role "$ROLE_ARN" \
    --handler "$HANDLER" \
    --architectures x86_64 \
    --zip-file "fileb://${TEMP_DIR}/function.zip" \
    --tags project=doc-smith,tutorial=lambda-gettingstarted \
    --query '[FunctionName, FunctionArn, Runtime, State]' \
    --output text 2>&1)

if [ -z "$CREATE_OUTPUT" ]; then
    echo "ERROR: Failed to create Lambda function"
    exit 1
fi
```



```

echo "$CREATE_OUTPUT"
CREATED_RESOURCES+=("lambda-function:${FUNCTION_NAME}")

wait_for_resource "Lambda function to become Active" \
    "aws lambda get-function-configuration --function-name ${FUNCTION_NAME} --
query State --output text" \
    "Active"

#####
# Step 3: Invoke the function
#####

echo ""
echo "=====
echo "Step 3: Invoke the function"
echo "=====
echo ""

TEST_EVENT='{"length": 6, "width": 7}'
echo "Invoking function with test event: ${TEST_EVENT}"
echo ""

echo "$TEST_EVENT" > "${TEMP_DIR}/test-event.json"

if ! validate_input "$TEST_EVENT" '"length": [0-9]+, "width": [0-9]+'; then
    echo "ERROR: Invalid test event format"
    exit 1
fi

INVOKE_OUTPUT=$(aws lambda invoke \
    --function-name "$FUNCTION_NAME" \
    --payload "fileb://${TEMP_DIR}/test-event.json" \
    --cli-read-timeout 30 \
    "${TEMP_DIR}/response.json" 2>&1)
echo "$INVOKE_OUTPUT"

if [ ! -f "${TEMP_DIR}/response.json" ]; then
    echo "ERROR: No response file generated"
    exit 1
fi

RESPONSE=$(cat "${TEMP_DIR}/response.json")
echo ""

```

```

echo "Function response: ${RESPONSE}"
echo ""

if echo "$INVOKE_OUTPUT" | grep -qi "functionerror"; then
    echo "WARNING: Function returned an error."
fi

#####
# Step 4: View CloudWatch logs
#####

echo ""
echo "=====
echo "Step 4: View CloudWatch Logs"
echo "=====
echo ""

echo "Log group: ${LOG_GROUP_NAME}"
echo ""

echo "Waiting for CloudWatch logs to be available..."

LOG_STREAMS=""
for i in $(seq 1 6); do
    LOG_STREAMS=$(aws logs describe-log-streams \
        --log-group-name "$LOG_GROUP_NAME" \
        --order-by LastEventTime \
        --descending \
        --query 'logStreams[0].logStreamName' \
        --output text 2>/dev/null) || true
    if [ -n "$LOG_STREAMS" ] && [ "$LOG_STREAMS" != "None" ]; then
        break
    fi
    LOG_STREAMS=""
    sleep 5
done

if [ -n "$LOG_STREAMS" ] && [ "$LOG_STREAMS" != "None" ]; then
    echo "Latest log stream: ${LOG_STREAMS}"
    echo ""
    echo "--- Log events ---"
    LOG_EVENTS=$(aws logs get-log-events \
        --log-group-name "$LOG_GROUP_NAME" \
        --log-stream-name "$LOG_STREAMS" \

```

```

        --query 'events[].message' \
        --output text 2>&1) || true
    echo "$LOG_EVENTS"
    echo "--- End of log events ---"
else
    echo "No log streams found yet. Logs may take a moment to appear."
    echo "You can view them in the CloudWatch console:"
    echo "  Log group: ${LOG_GROUP_NAME}"
fi

CREATED_RESOURCES+=("log-group:${LOG_GROUP_NAME}")

aws logs tag-log-group \
    --log-group-name "$LOG_GROUP_NAME" \
    --tags project=doc-smith,tutorial=lambda-gettingstarted

#####
# Summary and cleanup
#####

echo ""
echo "=====
echo "SUMMARY"
echo "=====
echo ""
echo "Resources created:"
echo "  IAM role:          ${ROLE_NAME}"
echo "  Lambda function:   ${FUNCTION_NAME}"
echo "  CloudWatch logs:    ${LOG_GROUP_NAME}"
echo ""
echo "=====
echo "CLEANUP"
echo "=====
echo ""
echo "Cleaning up all created resources..."
cleanup_resources

echo ""
echo "Done."

```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dei comandi della AWS CLI .
  - [AttachRolePolicy](#)

- [CreateFunction](#)
- [CreateRole](#)
- [DeleteFunction](#)
- [DeleteLogGroup](#)
- [DeleteRole](#)
- [DescribeLogStreams](#)
- [DetachRolePolicy](#)
- [GetFunction](#)
- [GetFunctionConfiguration](#)
- [GetLogEvents](#)
- [Invoke](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Richiamo a una funzione Lambda da un browser

Il seguente esempio di codice mostra come richiamare una AWS Lambda funzione da un browser.

### JavaScript

#### SDK per JavaScript (v2)

Puoi creare un'applicazione basata su browser che utilizza una AWS Lambda funzione per aggiornare una tabella Amazon DynamoDB con selezioni utente.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. [GitHub](#)

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda

## SDK per JavaScript (v3)

Puoi creare un'applicazione basata su browser che utilizza una AWS Lambda funzione per aggiornare una tabella Amazon DynamoDB con selezioni utente. Questa app utilizza la versione 3. AWS SDK per JavaScript

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Trasformare i dati per l'applicazione con S3 Object Lambda

L'esempio di codice seguente mostra come trasformare i dati per l'applicazione con S3 Object Lambda.

.NET

### SDK per .NET

Mostra come aggiungere codice personalizzato alle richieste S3 GET standard per modificare l'oggetto richiesto recuperato da S3 in modo che questo soddisfi le esigenze del client o dell'applicazione richiedente.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Lambda
- Simple Storage Service (Amazon S3)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzo di Gateway API per invocare una funzione Lambda

I seguenti esempi di codice mostrano come creare una AWS Lambda funzione richiamata da Amazon API Gateway.

### Java

#### SDK per Java 2.x

Mostra come creare una AWS Lambda funzione utilizzando l'API runtime Lambda Java. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. In questo esempio viene illustrato come creare una funzione Lambda invocata da Gateway Amazon API che analizza una tabella Amazon DynamoDB per le ricorrenze di lavoro e utilizza Amazon Simple Notification Service (Amazon SNS) per inviare un messaggio di testo ai dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#)

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon SNS

### JavaScript

#### SDK per JavaScript (v3)

Mostra come creare una AWS Lambda funzione utilizzando l'API di JavaScript runtime Lambda. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. In questo esempio viene illustrato come creare una funzione Lambda invocata da Gateway Amazon API che analizza una tabella Amazon DynamoDB per le ricorrenze di lavoro e utilizza

Amazon Simple Notification Service (Amazon SNS) per inviare un messaggio di testo ai dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. [GitHub](#)

Questo esempio è anche disponibile nella [Guida per lo sviluppatore di AWS SDK per JavaScript v3](#).

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon SNS

## Python

### SDK per Python (Boto3)

L'esempio mostra come creare e utilizzare una REST API di Gateway Amazon API destinata a una funzione AWS Lambda . Il gestore Lambda dimostra come definire route in base ai metodi HTTP, come ottenere dati dalla stringa, dall'intestazione e dal corpo della query e come restituire una risposta JSON.

- Distribuire una funzione Lambda.
- Creare una REST API di Gateway API.
- Creare una risorsa REST destinata alla funzione Lambda.
- Concedere l'autorizzazione affinché Gateway API invochi la funzione Lambda.
- Utilizza il pacchetto Richieste per inviare le richieste alla REST API.
- Eliminare tutte le risorse create durante la demo.

Questo esempio è visualizzato al meglio su GitHub. Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB

- Lambda
- Amazon SNS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzo di Step Functions per richiamare le funzioni Lambda

Il seguente esempio di codice mostra come creare una macchina a AWS Step Functions stati che richiama AWS Lambda funzioni in sequenza.

### Java

#### SDK per Java 2.x

Mostra come creare un flusso di lavoro AWS serverless utilizzando AWS Step Functions and. AWS SDK for Java 2.x Ogni fase del flusso di lavoro viene implementata utilizzando una AWS Lambda funzione.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzo degli eventi pianificati per invocare una funzione Lambda

I seguenti esempi di codice mostrano come creare una AWS Lambda funzione richiamata da un evento EventBridge pianificato di Amazon.



## Java

### SDK per Java 2.x

Mostra come creare un evento EventBridge pianificato da Amazon che richiami una AWS Lambda funzione. Configura EventBridge per utilizzare un'espressione cron per pianificare quando viene richiamata la funzione Lambda. In questo esempio, viene creata una funzione Lambda utilizzando l'API di runtime Lambda Java. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare un'app che invia un messaggio di testo via mobile ai tuoi dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. [GitHub](#)

Servizi utilizzati in questo esempio

- CloudWatch Registri
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

## JavaScript

### SDK per JavaScript (v3)

Mostra come creare un evento EventBridge pianificato da Amazon che richiami una AWS Lambda funzione. Configura EventBridge per utilizzare un'espressione cron per pianificare quando viene richiamata la funzione Lambda. In questo esempio, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare un'app che invia un messaggio di testo via mobile ai tuoi dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. [GitHub](#)

Questo esempio è anche disponibile nella [Guida per lo sviluppatore di AWS SDK per JavaScript v3](#).

## Servizi utilizzati in questo esempio

- CloudWatch Registri
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

## Python

### SDK per Python (Boto3)

Questo esempio mostra come registrare una AWS Lambda funzione come destinazione di un EventBridge evento Amazon pianificato. Il gestore Lambda scrive un messaggio intuitivo e i dati completi dell'evento su Amazon CloudWatch Logs per recuperarli in un secondo momento.

- Distribuzione di una funzione Lambda.
- Crea un evento EventBridge pianificato e rende la funzione Lambda la destinazione.
- Concede il permesso di EventBridge invocare la funzione Lambda.
- Stampa i dati più recenti dai CloudWatch registri per mostrare il risultato delle chiamate pianificate.
- Elimina tutte le risorse create durante la demo.

Questo esempio è visualizzato al meglio su [GitHub](#) Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su [GitHub](#).

## Servizi utilizzati in questo esempio

- CloudWatch Registri
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzare l'API Amazon Neptune per sviluppare una funzione Lambda che esegua query sui dati del grafico

L'esempio di codice seguente mostra come utilizzare l'API Neptune per eseguire query sui dati del grafico.

### Java

#### SDK per Java 2.x

Mostra come utilizzare l'API Java di Amazon Neptune per creare una funzione Lambda che eseguano query sui dati del grafico all'interno del VPC.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Lambda
- Neptune

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Utilizzo delle variabili di proprietà nei dashboard di monitoraggio per monitorare più funzioni serverless

L'esempio di codice seguente mostra come:

- Crea funzioni Lambda per il monitoraggio
- Crea un pannello di controllo CloudWatch
- Aggiungi una variabile di proprietà alla dashboard
- Eseguire la pulizia delle risorse

## Bash

### AWS CLI con lo script Bash

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel [repository dei tutorial sugli esempi di codice per gli sviluppatori](#).

```
#!/bin/bash

# CloudWatch Dashboard with Lambda Function Variable Script
# This script creates a CloudWatch dashboard with a property variable for Lambda
  function names

set -euo pipefail

# Security: Set restrictive umask
umask 0077

# Set up logging with secure permissions
LOG_FILE="cloudwatch-dashboard-script-v4.log"
touch "$LOG_FILE"
chmod 600 "$LOG_FILE"
echo "Starting script execution at $(date)" >> "$LOG_FILE"

# Function to log commands and their output (with sensitive data sanitization)
log_cmd() {
    local cmd="$1"
    local sanitized_cmd="${cmd//--password*/--password [REDACTED]}"
    sanitized_cmd="${sanitized_cmd//--secret*/--secret [REDACTED]}"
    echo "$(date): Running command: $sanitized_cmd" >> "$LOG_FILE"
    eval "$cmd" 2>&1 | tee -a "$LOG_FILE"
    return ${PIPESTATUS[0]}
}

# Function to check for errors in command output
check_error() {
    local cmd_output="$1"
    local cmd_status="$2"
```

```

    local error_msg="$3"

    if [ $cmd_status -ne 0 ] || echo "$cmd_output" | grep -qi "error"; then
        echo "ERROR: $error_msg" | tee -a "$LOG_FILE"
        # Sanitize output before logging
        local sanitized_output="${cmd_output//arn:aws:iam::[0-9]*/
arn:aws:iam::ACCOUNT_ID}"
        echo "Command output: $sanitized_output" | tee -a "$LOG_FILE"
        cleanup_resources
        exit 1
    fi
}

# Trap errors and cleanup
trap 'cleanup_resources' EXIT ERR INT TERM

# Function to clean up resources
cleanup_resources() {
    local exit_code=$?

    echo "" | tee -a "$LOG_FILE"
    echo "===== " | tee -a "$LOG_FILE"
    echo "CLEANUP PROCESS" | tee -a "$LOG_FILE"
    echo "===== " | tee -a "$LOG_FILE"

    if [ -n "${DASHBOARD_NAME:-}" ]; then
        echo "Deleting CloudWatch dashboard: $DASHBOARD_NAME" | tee -a
"$LOG_FILE"
        aws cloudwatch delete-dashboards --dashboard-names "$DASHBOARD_NAME" 2>&1
>> "$LOG_FILE" || true
    fi

    if [ -n "${LAMBDA_FUNCTION1:-}" ]; then
        echo "Deleting Lambda function: $LAMBDA_FUNCTION1" | tee -a "$LOG_FILE"
        aws lambda delete-function --function-name "$LAMBDA_FUNCTION1" 2>&1 >>
"$LOG_FILE" || true
    fi

    if [ -n "${LAMBDA_FUNCTION2:-}" ]; then
        echo "Deleting Lambda function: $LAMBDA_FUNCTION2" | tee -a "$LOG_FILE"
        aws lambda delete-function --function-name "$LAMBDA_FUNCTION2" 2>&1 >>
"$LOG_FILE" || true
    fi
}

```

```

    if [ -n "${ROLE_NAME:-}" ]; then
        echo "Detaching policy from role: $ROLE_NAME" | tee -a "$LOG_FILE"
        aws iam detach-role-policy --role-name "$ROLE_NAME" --policy-arn
arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole 2>&1 >>
"$LOG_FILE" || true

        echo "Deleting IAM role: $ROLE_NAME" | tee -a "$LOG_FILE"
        aws iam delete-role --role-name "$ROLE_NAME" 2>&1 >> "$LOG_FILE" || true
    fi

    # Clean up temporary files securely
    shred -vfz -n 3 trust-policy.json lambda_function.py lambda_function.zip 2>/
dev/null || rm -f trust-policy.json lambda_function.py lambda_function.zip

    echo "Cleanup completed." | tee -a "$LOG_FILE"

    return $exit_code
}

# Validate AWS CLI is installed and authenticated
if ! command -v aws &> /dev/null; then
    echo "ERROR: AWS CLI is not installed" | tee -a "$LOG_FILE"
    exit 1
fi

if ! aws sts get-caller-identity &> /dev/null; then
    echo "ERROR: AWS CLI is not properly authenticated" | tee -a "$LOG_FILE"
    exit 1
fi

# Get AWS region with validation
AWS_REGION=$(aws configure get region 2>/dev/null || echo "")
if [ -z "$AWS_REGION" ]; then
    AWS_REGION="us-east-1"
    echo "No region found in AWS config, defaulting to $AWS_REGION" | tee -a
"$LOG_FILE"
else
    echo "Using AWS region: $AWS_REGION" | tee -a "$LOG_FILE"
fi

# Validate region format
if ! [[ "$AWS_REGION" =~ ^[a-z]{2}-[a-z]+-[0-9]$ ]]; then
    echo "ERROR: Invalid AWS region format: $AWS_REGION" | tee -a "$LOG_FILE"
    exit 1

```

```

fi

# Generate unique identifiers using secure random with validation
RANDOM_ID=$(openssl rand -hex 6)
if [ -z "$RANDOM_ID" ] || [ ${#RANDOM_ID} -ne 12 ]; then
    echo "ERROR: Failed to generate valid random identifier" | tee -a "$LOG_FILE"
    exit 1
fi

DASHBOARD_NAME="LambdaMetricsDashboard-${RANDOM_ID}"
LAMBDA_FUNCTION1="TestFunction1-${RANDOM_ID}"
LAMBDA_FUNCTION2="TestFunction2-${RANDOM_ID}"
ROLE_NAME="LambdaExecutionRole-${RANDOM_ID}"

# Validate resource names don't exceed AWS limits
if [ ${#DASHBOARD_NAME} -gt 128 ] || [ ${#LAMBDA_FUNCTION1} -gt 64 ] ||
[ ${#ROLE_NAME} -gt 64 ]; then
    echo "ERROR: Generated resource names exceed AWS limits" | tee -a "$LOG_FILE"
    exit 1
fi

echo "Using random identifier: $RANDOM_ID" | tee -a "$LOG_FILE"
echo "Dashboard name: $DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "Lambda function names: $LAMBDA_FUNCTION1, $LAMBDA_FUNCTION2" | tee -a
"$LOG_FILE"
echo "IAM role name: $ROLE_NAME" | tee -a "$LOG_FILE"

# Create IAM role for Lambda functions
echo "Creating IAM role for Lambda..." | tee -a "$LOG_FILE"
TRUST_POLICY='{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}'

echo "$TRUST_POLICY" > trust-policy.json
chmod 600 trust-policy.json

```

```
# Validate JSON before use
if ! python3 -m json.tool trust-policy.json > /dev/null 2>&1; then
    echo "ERROR: Invalid trust policy JSON" | tee -a "$LOG_FILE"
    exit 1
fi

ROLE_OUTPUT=$(log_cmd "aws iam create-role --role-name '$ROLE_NAME' --assume-
role-policy-document file://trust-policy.json --tags Key=project,Value=doc-smith
Key=tutorial,Value=cloudwatch-streams --output json")
check_error "$ROLE_OUTPUT" $? "Failed to create IAM role"

ROLE_ARN=$(echo "$ROLE_OUTPUT" | python3 -c "import sys, json;
print(json.load(sys.stdin)['Role']['Arn'])" 2>/dev/null)
if [ -z "$ROLE_ARN" ]; then
    echo "ERROR: Failed to extract Role ARN" | tee -a "$LOG_FILE"
    exit 1
fi
echo "Role ARN: $ROLE_ARN" | tee -a "$LOG_FILE"

# Attach Lambda basic execution policy to the role
echo "Attaching Lambda execution policy to role..." | tee -a "$LOG_FILE"
POLICY_OUTPUT=$(log_cmd "aws iam attach-role-policy --role-name '$ROLE_NAME' --
policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole")
check_error "$POLICY_OUTPUT" $? "Failed to attach policy to role"

# Wait for role to propagate
echo "Waiting for IAM role to propagate..." | tee -a "$LOG_FILE"
sleep 10

# Create simple Python Lambda function code with security validation
echo "Creating Lambda function code..." | tee -a "$LOG_FILE"
cat > lambda_function.py << 'LAMBDA_EOF'
def handler(event, context):
    print("Lambda function executed successfully")
    return {
        'statusCode': 200,
        'body': 'Success'
    }
LAMBDA_EOF

chmod 600 lambda_function.py

# Validate Python syntax
```



```
if ! python3 -m py_compile lambda_function.py 2>/dev/null; then
    echo "ERROR: Invalid Python syntax in Lambda function" | tee -a "$LOG_FILE"
    exit 1
fi

# Zip the Lambda function code
zip -j -q lambda_function.zip lambda_function.py
if [ ! -f lambda_function.zip ]; then
    echo "ERROR: Failed to create lambda_function.zip" | tee -a "$LOG_FILE"
    exit 1
fi
chmod 600 lambda_function.zip

# Validate zip file integrity
if ! unzip -t lambda_function.zip > /dev/null 2>&1; then
    echo "ERROR: Created zip file is corrupted" | tee -a "$LOG_FILE"
    exit 1
fi

# Create first Lambda function
echo "Creating first Lambda function: $LAMBDA_FUNCTION1..." | tee -a "$LOG_FILE"
LAMBDA1_OUTPUT=$(log_cmd "aws lambda create-function --function-name
'$LAMBDA_FUNCTION1' --runtime python3.11 --role '$ROLE_ARN' --handler
lambda_function.handler --zip-file fileb://lambda_function.zip --timeout 30 --
memory-size 128 --tags project=doc-smith,tutorial=cloudwatch-streams")
check_error "$LAMBDA1_OUTPUT" $? "Failed to create first Lambda function"

# Create second Lambda function
echo "Creating second Lambda function: $LAMBDA_FUNCTION2..." | tee -a "$LOG_FILE"
LAMBDA2_OUTPUT=$(log_cmd "aws lambda create-function --function-name
'$LAMBDA_FUNCTION2' --runtime python3.11 --role '$ROLE_ARN' --handler
lambda_function.handler --zip-file fileb://lambda_function.zip --timeout 30 --
memory-size 128 --tags project=doc-smith,tutorial=cloudwatch-streams")
check_error "$LAMBDA2_OUTPUT" $? "Failed to create second Lambda function"

# Invoke Lambda functions to generate some metrics
echo "Invoking Lambda functions to generate metrics..." | tee -a "$LOG_FILE"
log_cmd "aws lambda invoke --function-name '$LAMBDA_FUNCTION1' --payload '{}' /
dev/null" || true
log_cmd "aws lambda invoke --function-name '$LAMBDA_FUNCTION2' --payload '{}' /
dev/null" || true

# Create CloudWatch dashboard with property variable
```

```

echo "Creating CloudWatch dashboard with property variable..." | tee -a
"$LOG_FILE"

# Create dashboard body with proper escaping and validation
DASHBOARD_BODY=$(cat <<EOF
{
  "widgets": [
    {
      "type": "metric",
      "x": 0,
      "y": 0,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "Invocations", "FunctionName", "$LAMBDA_FUNCTION1" ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$AWS_REGION",
        "title": "Lambda Invocations",
        "period": 300,
        "stat": "Sum"
      }
    }
  ]
}
EOF
)

# Validate JSON before sending
if ! echo "$DASHBOARD_BODY" | python3 -m json.tool > /dev/null 2>&1; then
  echo "ERROR: Dashboard body is not valid JSON" | tee -a "$LOG_FILE"
  exit 1
fi

# First create a basic dashboard without variables
echo "Creating initial dashboard without variables..." | tee -a "$LOG_FILE"
DASHBOARD_OUTPUT=$(aws cloudwatch put-dashboard --dashboard-name
"$DASHBOARD_NAME" --dashboard-body "$DASHBOARD_BODY" --output json 2>&1)
check_error "$DASHBOARD_OUTPUT" $? "Failed to create initial CloudWatch
dashboard"

# Now let's try to add a property variable using the console instructions

```

```

echo "To complete the tutorial, please follow these steps in the CloudWatch
  console:" | tee -a "$LOG_FILE"
echo "1. Open the CloudWatch console at https://console.aws.amazon.com/
cloudwatch/" | tee -a "$LOG_FILE"
echo "2. Navigate to Dashboards and select your dashboard: $DASHBOARD_NAME" | tee
  -a "$LOG_FILE"
echo "3. Choose Actions > Variables > Create a variable" | tee -a "$LOG_FILE"
echo "4. Choose Property variable" | tee -a "$LOG_FILE"
echo "5. For Property that the variable changes, choose FunctionName" | tee -a
  "$LOG_FILE"
echo "6. For Input type, choose Select menu (dropdown)" | tee -a "$LOG_FILE"
echo "7. Choose Use the results of a metric search" | tee -a "$LOG_FILE"
echo "8. Choose Pre-built queries > Lambda > Errors" | tee -a "$LOG_FILE"
echo "9. Choose By Function Name and then choose Search" | tee -a "$LOG_FILE"
echo "10. (Optional) Configure any secondary settings as desired" | tee -a
  "$LOG_FILE"
echo "11. Choose Add variable" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"
echo "The dashboard has been created and can be accessed at:" | tee -a
  "$LOG_FILE"
echo "https://console.aws.amazon.com/cloudwatch/home#dashboards:name=
$DASHBOARD_NAME" | tee -a "$LOG_FILE"

# Verify dashboard creation
echo "Verifying dashboard creation..." | tee -a "$LOG_FILE"
VERIFY_OUTPUT=$(aws cloudwatch get-dashboard --dashboard-name "$DASHBOARD_NAME"
  --output json 2>&1)
check_error "$VERIFY_OUTPUT" $? "Failed to verify dashboard creation"

echo "" | tee -a "$LOG_FILE"
echo "===== " | tee -a "$LOG_FILE"
echo "DASHBOARD CREATED SUCCESSFULLY" | tee -a "$LOG_FILE"
echo "===== " | tee -a "$LOG_FILE"
echo "Dashboard Name: $DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "Lambda Functions: $LAMBDA_FUNCTION1, $LAMBDA_FUNCTION2" | tee -a
  "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"
echo "You can view your dashboard in the CloudWatch console:" | tee -a
  "$LOG_FILE"
echo "https://console.aws.amazon.com/cloudwatch/home#dashboards:name=
$DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"

# Auto-confirm cleanup

```

```
echo "" | tee -a "$LOG_FILE"
echo "===== " | tee -a "$LOG_FILE"
echo "CLEANUP CONFIRMATION" | tee -a "$LOG_FILE"
echo "===== " | tee -a "$LOG_FILE"
echo "The following resources were created:" | tee -a "$LOG_FILE"
echo "- CloudWatch Dashboard: $DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "- Lambda Function: $LAMBDA_FUNCTION1" | tee -a "$LOG_FILE"
echo "- Lambda Function: $LAMBDA_FUNCTION2" | tee -a "$LOG_FILE"
echo "- IAM Role: $ROLE_NAME" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"
echo "Auto-confirming cleanup of all created resources..." | tee -a "$LOG_FILE"

echo "Script completed successfully." | tee -a "$LOG_FILE"
exit 0
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dei comandi della AWS CLI .
  - [AttachRolePolicy](#)
  - [CreateFunction](#)
  - [CreateRole](#)
  - [DeleteDashboards](#)
  - [DeleteFunction](#)
  - [DeleteRole](#)
  - [DetachRolePolicy](#)
  - [GetDashboard](#)
  - [Invoke](#)
  - [PutDashboard](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Scrivi dati di attività personalizzati con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito utilizzando un AWS SDK

L'esempio di codice seguente mostra come scrivere dati di attività personalizzate con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito.

- Usa le funzioni di amministratore per aggiungere un utente a un pool di utenti.
- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `PostAuthentication`.
- Accedere con il nuovo utente ad Amazon Cognito.
- La funzione Lambda scrive informazioni personalizzate nei CloudWatch log e in una tabella DynamoDB.
- Acquisisci e visualizza dati personalizzati dalla tabella DynamoDB, quindi elimina le risorse.

Go

## SDK per Go V2

### Note

C'è altro su. [GitHub](#) Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel [Repository di esempi di codice AWS](#).

Esegui uno scenario interattivo al prompt dei comandi.

```
import (  
    "context"  
    "errors"  
    "log"  
    "strings"  
    "user_pools_and_lambda_triggers/actions"  
  
    "github.com/aws/aws-sdk-go-v2/aws"  
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"  
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"  
    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"  
)  
  
// ActivityLog separates the steps of this scenario into individual functions so  
// that  
// they are simpler to read and understand.  
type ActivityLog struct {  
    helper      IScenarioHelper  
    questioner  demotools.IQuestioner
```

```

resources    Resources
cognitoActor *actions.CognitoActions
}

// NewActivityLog constructs a new activity log runner.
func NewActivityLog(sdkConfig aws.Config, questioner demotools.IQuestioner,
    helper IScenarioHelper) ActivityLog {
    scenario := ActivityLog{
        helper:      helper,
        questioner:   questioner,
        resources:    Resources{},
        cognitoActor: &actions.CognitoActions{CognitoClient:
cognitoidentityprovider.NewFromConfig(sdkConfig)},
    }
    scenario.resources.init(scenario.cognitoActor, questioner)
    return scenario
}

// AddUserToPool selects a user from the known users table and uses administrator
    credentials to add the user to the user pool.
func (runner *ActivityLog) AddUserToPool(ctx context.Context, userPoolId string,
    tableName string) (string, string) {
    log.Println("To facilitate this example, let's add a user to the user pool using
    administrator privileges.")
    users, err := runner.helper.GetKnownUsers(ctx, tableName)
    if err != nil {
        panic(err)
    }
    user := users.Users[0]
    log.Printf("Adding known user %v to the user pool.\n", user.UserName)
    err = runner.cognitoActor.AdminCreateUser(ctx, userPoolId, user.UserName,
    user.UserEmail)
    if err != nil {
        panic(err)
    }
    pwSet := false
    password := runner.questioner.AskPassword("\nEnter a password that has at least
    eight characters, uppercase, lowercase, numbers and symbols.\n"+
        "(the password will not display as you type):", 8)
    for !pwSet {
        log.Printf("\nSetting password for user '%v'.\n", user.UserName)
        err = runner.cognitoActor.AdminSetUserPassword(ctx, userPoolId, user.UserName,
        password)
        if err != nil {

```

```

    var invalidPassword *types.InvalidPasswordException
    if errors.As(err, &invalidPassword) {
        password = runner.questioner.AskPassword("\nEnter another password:", 8)
    } else {
        panic(err)
    }
} else {
    pwSet = true
}
}

log.Println(strings.Repeat("-", 88))

return user.UserName, password
}

// AddActivityLogTrigger adds a Lambda handler as an invocation target for the
// PostAuthentication trigger.
func (runner *ActivityLog) AddActivityLogTrigger(ctx context.Context, userPoolId
string, activityLogArn string) {
    log.Println("Let's add a Lambda function to handle the PostAuthentication
trigger from Cognito.\n" +
        "This trigger happens after a user is authenticated, and lets your function
take action, such as logging\n" +
        "the outcome.")
    err := runner.cognitoActor.UpdateTriggers(
        ctx, userPoolId,
        actions.TriggerInfo{Trigger: actions.PostAuthentication, HandlerArn:
aws.String(activityLogArn)})
    if err != nil {
        panic(err)
    }
    runner.resources.triggers = append(runner.resources.triggers,
        actions.PostAuthentication)
    log.Printf("Lambda function %v added to user pool %v to handle
PostAuthentication Cognito trigger.\n",
        activityLogArn, userPoolId)

    log.Println(strings.Repeat("-", 88))
}

// SignInUser signs in as the specified user.
func (runner *ActivityLog) SignInUser(ctx context.Context, clientId string,
    userName string, password string) {

```

```

log.Printf("Now we'll sign in user %v and check the results in the logs and the
DynamoDB table.", userName)
runner.questioner.Ask("Press Enter when you're ready.")
authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
if err != nil {
    panic(err)
}
log.Println("Sign in successful.",
    "The PostAuthentication Lambda handler writes custom information to CloudWatch
Logs.")

runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
*authResult.AccessToken)
}

// GetKnownUserLastLogin gets the login info for a user from the Amazon DynamoDB
table and displays it.
func (runner *ActivityLog) GetKnownUserLastLogin(ctx context.Context, tableName
string, userName string) {
    log.Println("The PostAuthentication handler also writes login data to the
DynamoDB table.")
    runner.questioner.Ask("Press Enter when you're ready to continue.")
    users, err := runner.helper.GetKnownUsers(ctx, tableName)
    if err != nil {
        panic(err)
    }
    for _, user := range users.Users {
        if user.UserName == userName {
            log.Println("The last login info for the user in the known users table is:")
            log.Printf("\t%+v", *user.LastLogin)
        }
    }
    log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *ActivityLog) Run(ctx context.Context, stackName string) {
    defer func() {
        if r := recover(); r != nil {
            log.Println("Something went wrong with the demo.")
            runner.resources.Cleanup(ctx)
        }
    }()
}

```



```

log.Println(strings.Repeat("-", 88))
log.Printf("Welcome\n")

log.Println(strings.Repeat("-", 88))

stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
if err != nil {
    panic(err)
}
runner.resources.userPoolId = stackOutputs["UserPoolId"]
runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])
userName, password := runner.AddUserToPool(ctx, stackOutputs["UserPoolId"],
stackOutputs["TableName"])

runner.AddActivityLogTrigger(ctx, stackOutputs["UserPoolId"],
stackOutputs["ActivityLogFunctionArn"])
runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password)
runner.helper.ListRecentLogEvents(ctx, stackOutputs["ActivityLogFunction"])
runner.GetKnownUserLastLogin(ctx, stackOutputs["TableName"], userName)

runner.resources.Cleanup(ctx)

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}

```

Gestisci il trigger PostAuthentication con una funzione Lambda.

```

import (
    "context"
    "fmt"
    "log"
    "os"
    "time"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"

```

```

    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    dynamodbtypes "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

const TABLE_NAME = "TABLE_NAME"

// LoginInfo defines structured login data that can be marshalled to a DynamoDB
// format.
type LoginInfo struct {
    UserPoolId string `dynamodbav:"UserPoolId"`
    ClientId   string `dynamodbav:"ClientId"`
    Time       string `dynamodbav:"Time"`
}

// UserInfo defines structured user data that can be marshalled to a DynamoDB
// format.
type UserInfo struct {
    Username   string `dynamodbav:"Username"`
    UserEmail  string `dynamodbav:"UserEmail"`
    LastLogin  LoginInfo `dynamodbav:"LastLogin"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
    userEmail, err := attributevalue.Marshal(user.UserEmail)
    if err != nil {
        panic(err)
    }
    return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
    dynamoClient *dynamodb.Client
}

// HandleRequest handles the PostAuthentication event by writing custom data to
// the logs and
// to an Amazon DynamoDB table.
func (h *handler) HandleRequest(ctx context.Context,
    event events.CognitoEventUserPoolsPostAuthentication)
    (events.CognitoEventUserPoolsPostAuthentication, error) {
    log.Printf("Received post authentication trigger from %v for user '%v'",
        event.TriggerSource, event.Username)

```

```
tableName := os.Getenv(TABLE_NAME)
user := UserInfo{
    UserName: event.UserName,
    UserEmail: event.Request.UserAttributes["email"],
    LastLogin: LoginInfo{
        UserPoolId: event.UserPoolID,
        ClientId: event.CallerContext.ClientID,
        Time: time.Now().Format(time.UnixDate),
    },
}
// Write to CloudWatch Logs.
fmt.Printf("%#v", user)

// Also write to an external system. This examples uses DynamoDB to demonstrate.
userMap, err := attributevalue.MarshalMap(user)
if err != nil {
    log.Printf("Couldn't marshal to DynamoDB map. Here's why: %v\n", err)
} else if len(userMap) == 0 {
    log.Printf("User info marshaled to an empty map.")
} else {
    _, err := h.dynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
        Item: userMap,
        TableName: aws.String(tableName),
    })
    if err != nil {
        log.Printf("Couldn't write to DynamoDB. Here's why: %v\n", err)
    } else {
        log.Printf("Wrote user info to DynamoDB table %v.\n", tableName)
    }
}

return event, nil
}

func main() {
    ctx := context.Background()
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        log.Panicln(err)
    }
    h := handler{
        dynamoClient: dynamodb.NewFromConfig(sdkConfig),
    }
    lambda.Start(h.HandleRequest)
```

```
}

```

Crea una struttura che esegue operazioni comuni.

```
import (
    "context"
    "log"
    "strings"
    "time"
    "user_pools_and_lambda_triggers/actions"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudformation"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this
// example.
type IScenarioHelper interface {
    Pause(secs int)
    GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
        error)
    PopulateUserTable(ctx context.Context, tableName string)
    GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
    AddKnownUser(ctx context.Context, tableName string, user actions.User)
    ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
// example.
type ScenarioHelper struct {
    questioner    demotools.IQuestioner
    dynamoActor  *actions.DynamoActions
    cfnActor      *actions.CloudFormationActions
    cwlActor      *actions.CloudWatchLogsActions
    isTestRun     bool
}
```

```
// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
    ScenarioHelper {
    scenario := ScenarioHelper{
        questioner: questioner,
        dynamoActor: &actions.DynamoActions{DynamoClient:
dynamodb.NewFromConfig(sdkConfig)},
        cfnActor: &actions.CloudFormationActions{CfnClient:
cloudformation.NewFromConfig(sdkConfig)},
        cwlActor: &actions.CloudWatchLogsActions{CwlClient:
cloudwatchlogs.NewFromConfig(sdkConfig)},
    }
    return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
    if !helper.isTestRun {
        time.Sleep(time.Duration(secs) * time.Second)
    }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
    structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
    string) (actions.StackOutputs, error) {
    return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
    string) {
    log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
    this example.\n", tableName)
    err := helper.dynamoActor.PopulateTable(ctx, tableName)
    if err != nil {
        panic(err)
    }
}

// GetKnownUsers gets the users from the known users table in a structured
    format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
    (actions.UserList, error) {
```

```

knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
if err != nil {
    log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
        tableName, err)
}
return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
    user actions.User) {
    log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
        table...\n",
        user.UserName, user.UserEmail)
    err := helper.dynamoActor.AddUser(ctx, tableName, user)
    if err != nil {
        panic(err)
    }
}

// ListRecentLogEvents gets the most recent log stream and events for the
    specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
    functionName string) {
    log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
    helper.Pause(10)
    log.Println("Okay, let's check the logs to find what's happened recently with
        your Lambda function.")
    logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
    if err != nil {
        panic(err)
    }
    log.Printf("Getting some recent events from log stream %v\n",
        *logStream.LogStreamName)
    events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
        *logStream.LogStreamName, 10)
    if err != nil {
        panic(err)
    }
    for _, event := range events {
        log.Printf("\t%v", *event.Message)
    }
    log.Println(strings.Repeat("-", 88))
}

```

Crea una struttura che racchiude le azioni di Amazon Cognito.

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
// trigger.
type Trigger int

const (
    PreSignUp Trigger = iota
    UserMigration
    PostAuthentication
)

type TriggerInfo struct {
    Trigger    Trigger
    HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
// is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
string, triggers ...TriggerInfo) error {
```

```

output, err := actor.CognitoClient.DescribeUserPool(ctx,
&cognitoidentityprovider.DescribeUserPoolInput{
    UserPoolId: aws.String(userPoolId),
})
if err != nil {
    log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
userPoolId, err)
    return err
}
lambdaConfig := output.UserPool.LambdaConfig
for _, trigger := range triggers {
    switch trigger.Trigger {
    case PreSignUp:
        lambdaConfig.PreSignUp = trigger.HandlerArn
    case UserMigration:
        lambdaConfig.UserMigration = trigger.HandlerArn
    case PostAuthentication:
        lambdaConfig.PostAuthentication = trigger.HandlerArn
    }
}
_, err = actor.CognitoClient.UpdateUserPool(ctx,
&cognitoidentityprovider.UpdateUserPoolInput{
    UserPoolId:    aws.String(userPoolId),
    LambdaConfig: lambdaConfig,
})
if err != nil {
    log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
}
return err
}

// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
string, password string, userEmail string) (bool, error) {
    confirmed := false
    output, err := actor.CognitoClient.SignUp(ctx,
&cognitoidentityprovider.SignUpInput{
        ClientId: aws.String(clientId),
        Password: aws.String(password),
        Username: aws.String(userName),
        UserAttributes: []types.AttributeType{
            {Name: aws.String("email"), Value: aws.String(userEmail)},

```



```

    },
  })
  if err != nil {
    var invalidPassword *types.InvalidPasswordException
    if errors.As(err, &invalidPassword) {
      log.Println(*invalidPassword.Message)
    } else {
      log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
    }
  } else {
    confirmed = output.UserConfirmed
  }
  return confirmed, err
}

// SignIn signs in a user to Amazon Cognito using a username and password
// authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
string, password string) (*types.AuthenticationResultType, error) {
  var authResult *types.AuthenticationResultType
  output, err := actor.CognitoClient.InitiateAuth(ctx,
    &cognitoidentityprovider.InitiateAuthInput{
      AuthFlow:      "USER_PASSWORD_AUTH",
      ClientId:      aws.String(clientId),
      AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
    })
  if err != nil {
    var resetRequired *types.PasswordResetRequiredException
    if errors.As(err, &resetRequired) {
      log.Println(*resetRequired.Message)
    } else {
      log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
    }
  } else {
    authResult = output.AuthenticationResult
  }
  return authResult, err
}

```

```
// ForgotPassword starts a password recovery flow for a user. This flow typically
// sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
    userName string) (*types.CodeDeliveryDetailsType, error) {
    output, err := actor.CognitoClient.ForgotPassword(ctx,
        &cognitoidentityprovider.ForgotPasswordInput{
            ClientId: aws.String(clientId),
            Username: aws.String(userName),
        })
    if err != nil {
        log.Printf("Couldn't start password reset for user '%v'. Here's why: %v\n",
            userName, err)
    }
    return output.CodeDeliveryDetails, err
}

// ConfirmForgotPassword confirms a user with a confirmation code and a new
// password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
    string, code string, userName string, password string) error {
    _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
        &cognitoidentityprovider.ConfirmForgotPasswordInput{
            ClientId:      aws.String(clientId),
            ConfirmationCode: aws.String(code),
            Password:      aws.String(password),
            Username:      aws.String(userName),
        })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
        }
    }
    return err
}

// DeleteUser removes a user from the user pool.
```

```

func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
string) error {
    _, err := actor.CognitoClient.DeleteUser(ctx,
&cognitoidentityprovider.DeleteUserInput{
    AccessToken: aws.String(userAccessToken),
})
    if err != nil {
        log.Printf("Couldn't delete user. Here's why: %v\n", err)
    }
    return err
}

// AdminCreateUser uses administrator credentials to add a user to a user pool.
// This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
string, userName string, userEmail string) error {
    _, err := actor.CognitoClient.AdminCreateUser(ctx,
&cognitoidentityprovider.AdminCreateUserInput{
    UserPoolId:      aws.String(userPoolId),
    Username:        aws.String(userName),
    MessageAction:   types.MessageActionTypeSuppress,
    UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
aws.String(userEmail)}}},
})
    if err != nil {
        var userExists *types.UsernameExistsException
        if errors.As(err, &userExists) {
            log.Printf("User %v already exists in the user pool.", userName)
            err = nil
        } else {
            log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
        }
    }
    return err
}

// AdminSetUserPassword uses administrator credentials to set a password for a
// user without requiring a
// temporary password.

```

```
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
string, userName string, password string) error {
_, err := actor.CognitoClient.AdminSetUserPassword(ctx,
&cognitoidentityprovider.AdminSetUserPasswordInput{
    Password:    aws.String(password),
    UserPoolId:  aws.String(userPoolId),
    Username:    aws.String(userName),
    Permanent:   true,
})
if err != nil {
    var invalidPassword *types.InvalidPasswordException
    if errors.As(err, &invalidPassword) {
        log.Println(*invalidPassword.Message)
    } else {
        log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
err)
    }
}
return err
}
```

Crea una struttura che racchiude le azioni di DynamoDB.

```
import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
actions
// used in the examples.
type DynamoActions struct {
    DynamoClient *dynamodb.Client
}
```

```

// User defines structured user data.
type User struct {
    UserName  string
    UserEmail string
    LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
    UserPoolId string
    ClientId   string
    Time       string
}

// UserList defines a list of users.
type UserList struct {
    Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
strings.
func (users *UserList) UserNameList() []string {
    names := make([]string, len(users.Users))
    for i := 0; i < len(users.Users); i++ {
        names[i] = users.Users[i].UserName
    }
    return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
error {
    var err error
    var item map[string]types.AttributeValue
    var writeReqs []types.WriteRequest
    for i := 1; i < 4; i++ {
        item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
        if err != nil {
            log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
err)
            return err
        }
    }
}

```

```

    writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
&types.PutRequest{Item: item}})
}
_, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
    RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
})
if err != nil {
    log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
tableName, err)
}
return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
error) {
    var userList UserList
    output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
        TableName: aws.String(tableName),
    })
    if err != nil {
        log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
err)
    } else {
        err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
        if err != nil {
            log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
        }
    }
    return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user
User) error {
    userItem, err := attributevalue.MarshalMap(user)
    if err != nil {
        log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
    }
    _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
        Item:      userItem,
        TableName: aws.String(tableName),
    })
    if err != nil {

```

```

    log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
}
return err
}

```

Crea una struttura che racchiuda le azioni di CloudWatch Logs.

```

import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
    CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
    functionName string) (types.LogStream, error) {
    var logStream types.LogStream
    logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
    output, err := actor.CwlClient.DescribeLogStreams(ctx,
        &cloudwatchlogs.DescribeLogStreamsInput{
            Descending:    aws.Bool(true),
            Limit:         aws.Int32(1),
            LogGroupName: aws.String(logGroupName),
            OrderBy:       types.OrderByLastEventTime,
        })
    if err != nil {
        log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
            logGroupName, err)
    } else {
        logStream = output.LogStreams[0]
    }
    return logStream, err
}

```

```

}

// GetLogEvents gets the most recent eventCount events from the specified log
// stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
    string, logStreamName string, eventCount int32) (
    []types.OutputLogEvent, error) {
    var events []types.OutputLogEvent
    logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
    output, err := actor.CwlClient.GetLogEvents(ctx,
    &cloudwatchlogs.GetLogEventsInput{
        LogStreamName: aws.String(logStreamName),
        Limit:          aws.Int32(eventCount),
        LogGroupName:   aws.String(logGroupName),
    })
    if err != nil {
        log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
        logStreamName, err)
    } else {
        events = output.Events
    }
    return events, err
}

```

Crea una struttura che racchiuda le azioni. CloudFormation

```

import (
    "context"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
    CfnClient *cloudformation.Client
}

```



```
// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
// structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
string) StackOutputs {
    output, err := actor.CfnClient.DescribeStacks(ctx,
&cloudformation.DescribeStacksInput{
    StackName: aws.String(stackName),
})
    if err != nil || len(output.Stacks) == 0 {
        log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
stackName, err)
    }
    stackOutputs := StackOutputs{}
    for _, out := range output.Stacks[0].Outputs {
        stackOutputs[*out.OutputKey] = *out.OutputValue
    }
    return stackOutputs
}
```

Eliminare le risorse.

```
import (
    "context"
    "log"
    "user_pools_and_lambda_triggers/actions"

    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
    userPoolId      string
    userAccessTokens []string
    triggers        []actions.Trigger

    cognitoActor *actions.CognitoActions
    questioner   demotools.IQuestioner
}
```

```

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
demotools.IQuestioner) {
    resources.userAccessTokens = []string{}
    resources.triggers = []actions.Trigger{}
    resources.cognitoActor = cognitoActor
    resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
    defer func() {
        if r := recover(); r != nil {
            log.Printf("Something went wrong during cleanup.\n%v\n", r)
            log.Println("Use the AWS Management Console to remove any remaining resources
\n" +
                "that were created for this scenario.")
        }
    }()

    wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
resources that were created "+
        "during this demo (y/n)?", "y")
    if wantDelete {
        for _, accessToken := range resources.userAccessTokens {
            err := resources.cognitoActor.DeleteUser(ctx, accessToken)
            if err != nil {
                log.Println("Couldn't delete user during cleanup.")
                panic(err)
            }
            log.Println("Deleted user.")
        }
        triggerList := make([]actions.TriggerInfo, len(resources.triggers))
        for i := 0; i < len(resources.triggers); i++ {
            triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
HandlerArn: nil}
        }
        err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
triggerList...)
        if err != nil {
            log.Println("Couldn't update Cognito triggers during cleanup.")
            panic(err)
        }
        log.Println("Removed Cognito triggers from user pool.")
    }
}

```

```
} else {  
    log.Println("Be sure to remove resources when you're done with them to avoid  
unexpected charges!")  
}  
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - [AdminCreateUser](#)
  - [AdminSetUserPassword](#)
  - [DeleteUser](#)
  - [InitiateAuth](#)
  - [UpdateUserPool](#)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. [Usare Lambda con un AWS SDK](#) Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Esempi serverless per Lambda

I seguenti esempi di codice mostrano come usare Lambda con AWS gli SDK.

### Esempi

- [Connessione a un database Amazon RDS in una funzione Lambda](#)
- [Invocare una funzione Lambda da un trigger Kinesis](#)
- [Richiamare una funzione Lambda da un trigger DynamoDB](#)
- [Richiamare una funzione Lambda da un trigger Amazon DocumentDB](#)
- [Invocare una funzione Lambda da un trigger Amazon MSK](#)
- [Invocazione di una funzione Lambda da un trigger Amazon S3](#)
- [Invocazione di una funzione Lambda da un trigger Amazon SNS](#)
- [Richiamo di una funzione Lambda da un trigger Amazon SQS](#)
- [Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis](#)
- [Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB](#)

- [Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS](#)

## Connessione a un database Amazon RDS in una funzione Lambda

Gli esempi di codice seguenti mostrano come implementare una funzione Lambda che si connette a un database RDS. La funzione effettua una semplice richiesta al database e restituisce il risultato.

.NET

SDK per .NET

### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda con .NET.

```
using System.Data;
using System.Text.Json;
using Amazon.Lambda.APIGatewayEvents;
using Amazon.Lambda.Core;
using MySql.Data.MySqlClient;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace aws_rds;

public class InputModel
{
    public string key1 { get; set; }
    public string key2 { get; set; }
}

public class Function
{
    /// <summary>
```

```

    // Handles the Lambda function execution for connecting to RDS using IAM
    authentication.
    /// </summary>
    /// <param name="input">The input event data passed to the Lambda function</
param>
    /// <param name="context">The Lambda execution context that provides runtime
information</param>
    /// <returns>A response object containing the execution result</returns>

    public async Task<APIGatewayProxyResponse>
    FunctionHandler(APIGatewayProxyRequest request, ILambdaContext context)
    {
        // Sample Input: {"body": "{\"key1\":\"20\", \"key2\":\"25\"}"}
        var input = JsonSerializer.Deserialize<InputModel>(request.Body);

        /// Obtain authentication token
        var authToken = RDSAuthTokenGenerator.GenerateAuthToken(
            Environment.GetEnvironmentVariable("RDS_ENDPOINT"),
            Convert.ToInt32(Environment.GetEnvironmentVariable("RDS_PORT")),
            Environment.GetEnvironmentVariable("RDS_USERNAME")
        );

        /// Build the Connection String with the Token
        string connectionString =
        $"Server={Environment.GetEnvironmentVariable("RDS_ENDPOINT"));";

        $"Port={Environment.GetEnvironmentVariable("RDS_PORT"));";

        $"Uid={Environment.GetEnvironmentVariable("RDS_USERNAME"));"; +
        $"Pwd={authToken});";

        try
        {
            await using var connection = new MySqlConnection(connectionString);
            await connection.OpenAsync();

            const string sql = "SELECT @param1 + @param2 AS Sum";

            await using var command = new MySqlCommand(sql, connection);
            command.Parameters.AddWithValue("@param1", int.Parse(input.key1 ??
"0"));
            command.Parameters.AddWithValue("@param2", int.Parse(input.key2 ??
"0"));

```

```

        await using var reader = await command.ExecuteReaderAsync();
        if (await reader.ReadAsync())
        {
            int result = reader.GetInt32("Sum");

            //Sample Response: {"statusCode":200,"body":{"\"message\": \"The
sum is: 45\\\"}","isBase64Encoded":false}
            return new APIGatewayProxyResponse
            {
                StatusCode = 200,
                Body = JsonSerializer.Serialize(new { message = $"The sum is:
{result}" })
            };
        }

        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }

        return new APIGatewayProxyResponse
        {
            StatusCode = 500,
            Body = JsonSerializer.Serialize(new { error = "Internal server
error" })
        };
    }
}

```

Go

SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Connessione a un database Amazon RDS in una funzione Lambda tramite Go.

```
/*
Golang v2 code here.
*/

package main

import (
    "context"
    "database/sql"
    "encoding/json"
    "fmt"
    "os"

    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
    _ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
    Name string `json:"name"`
}

func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

    var dbName string = os.Getenv("DatabaseName")
    var dbUser string = os.Getenv("DatabaseUser")
    var dbHost string = os.Getenv("DBHost") // Add hostname without https
    var dbPort int = os.Getenv("Port")      // Add port number
    var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
    var region string = os.Getenv("AWS_REGION")

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
        context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
        panic("failed to create authentication token: " + err.Error())
    }
}
```

```
dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
    dbUser, authenticationToken, dbEndpoint, dbName,
)

db, err := sql.Open("mysql", dsn)
if err != nil {
    panic(err)
}

defer db.Close()

var sum int
err = db.QueryRow("SELECT ?+? AS sum", 3, 2).Scan(&sum)
if err != nil {
    panic(err)
}
s := fmt.Sprint(sum)
message := fmt.Sprintf("The selected sum is: %s", s)

messageBytes, err := json.Marshal(message)
if err != nil {
    return nil, err
}

messageString := string(messageBytes)
return map[string]interface{}{
    "statusCode": 200,
    "headers":    map[string]string{"Content-Type": "application/json"},
    "body":       messageString,
}, nil
}

func main() {
    lambda.Start(HandleRequest)
}
```



## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rdsdata.RdsDataClient;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementRequest;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementResponse;
import software.amazon.awssdk.services.rdsdata.model.Field;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class RdsLambdaHandler implements
    RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    @Override
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent
        event, Context context) {
        APIGatewayProxyResponseEvent response = new
            APIGatewayProxyResponseEvent();

        try {
            // Obtain auth token
            String token = createAuthToken();

            // Define connection configuration
```

```

        String connectionString = String.format("jdbc:mysql://%s:%s/%s?
useSSL=true&requireSSL=true",
            System.getenv("ProxyHostName"),
            System.getenv("Port"),
            System.getenv("DBName"));

        // Establish a connection to the database
        try (Connection connection =
DriverManager.getConnection(connectionString, System.getenv("DBUserName"),
token);

            PreparedStatement statement =
connection.prepareStatement("SELECT ? + ? AS sum")) {

            statement.setInt(1, 3);
            statement.setInt(2, 2);

            try (ResultSet resultSet = statement.executeQuery()) {
                if (resultSet.next()) {
                    int sum = resultSet.getInt("sum");
                    response.setStatusCode(200);
                    response.setBody("The selected sum is: " + sum);
                }
            }
        }

        } catch (Exception e) {
            response.setStatusCode(500);
            response.setBody("Error: " + e.getMessage());
        }

        return response;
    }

    private String createAuthToken() {
        // Create RDS Data Service client
        RdsDataClient rdsDataClient = RdsDataClient.builder()
            .region(Region.of(System.getenv("AWS_REGION")))
            .credentialsProvider(DefaultCredentialsProvider.create())
            .build();

        // Define authentication request
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
            .resourceArn(System.getenv("ProxyHostName"))
            .secretArn(System.getenv("DBUserName"))

```

```

        .database(System.getenv("DBName"))
        .sql("SELECT 'RDS IAM Authentication'")
        .build();

    // Execute request and obtain authentication token
    ExecuteStatementResponse response =
rdsDataClient.executeStatement(request);
    Field tokenField = response.records().get(0).get(0);

    return tokenField.stringValue();
}
}

```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Connessione a un database Amazon RDS in una funzione Lambda utilizzando JavaScript

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
    // Define connection authentication parameters
    const dbinfo = {

        hostname: process.env.ProxyHostName,
        port: process.env.Port,
        username: process.env.DBUserName,

```

```
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;
}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

## Connessione a un database Amazon RDS in una funzione Lambda utilizzando TypeScript

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that
// the DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!

async function createAuthToken(): Promise<string> {

    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });

    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
        });
        const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
        console.log('result:', rows);
    }
}
```

```
        return rows;
    }
    catch (err) {
        console.log(err);
    }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number;
body: string }> => {
    // Execute database flow
    const result = await dbOps();

    // Return error if result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

## PHP

### SDK per PHP

#### Note

C'è di più su. [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite PHP.

```
<?php
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
```

```
# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    private function getAuthToken(): string {
        // Define connection authentication parameters
        $dbConnection = [
            'hostname' => getenv('DB_HOSTNAME'),
            'port' => getenv('DB_PORT'),
            'username' => getenv('DB_USERNAME'),
            'region' => getenv('AWS_REGION'),
        ];

        // Create RDS AuthTokenGenerator object
        $generator = new
        AuthTokenGenerator(CredentialProvider::defaultProvider());

        // Request authorization token from RDS, specifying the username
        return $generator->createToken(
            $dbConnection['hostname'] . ':' . $dbConnection['port'],
            $dbConnection['region'],
            $dbConnection['username']
        );
    }

    private function getQueryResults() {
        // Obtain auth token
        $token = $this->getAuthToken();
    }
}
```

```

        // Define connection configuration
        $connectionConfig = [
            'host' => getenv('DB_HOSTNAME'),
            'user' => getenv('DB_USERNAME'),
            'password' => $token,
            'database' => getenv('DB_NAME'),
        ];

        // Create the connection to the DB
        $conn = new PDO(
            "mysql:host={$connectionConfig['host']};dbname={$connectionConfig['database']}",
            $connectionConfig['user'],
            $connectionConfig['password'],
            [
                PDO::MYSQL_ATTR_SSL_CA => '/path/to/rds-ca-2019-root.pem',
                PDO::MYSQL_ATTR_SSL_VERIFY_SERVER_CERT => true,
            ]
        );

        // Obtain the result of the query
        $stmt = $conn->prepare('SELECT ?+? AS sum');
        $stmt->execute([3, 2]);

        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    /**
     * @param mixed $event
     * @param Context $context
     * @return array
     */
    public function handle(mixed $event, Context $context): array
    {
        $this->logger->info("Processing query");

        // Execute database flow
        $result = $this->getQueryResults();

        return [
            'sum' => $result['sum']
        ];
    }
}

```



```
$logger = new StderrLogger();  
return new Handler($logger);
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite Python.

```
import json  
import os  
import boto3  
import pymysql  
  
# RDS settings  
proxy_host_name = os.environ['PROXY_HOST_NAME']  
port = int(os.environ['PORT'])  
db_name = os.environ['DB_NAME']  
db_user_name = os.environ['DB_USER_NAME']  
aws_region = os.environ['AWS_REGION']  
  
# Fetch RDS Auth Token  
def get_auth_token():  
    client = boto3.client('rds')  
    token = client.generate_db_auth_token(  
        DBHostname=proxy_host_name,  
        Port=port  
        DBUsername=db_user_name  
        Region=aws_region  
    )  
    return token  
  
def lambda_handler(event, context):  
    token = get_auth_token()
```

```
try:
    connection = pymysql.connect(
        host=proxy_host_name,
        user=db_user_name,
        password=token,
        db=db_name,
        port=port,
        ssl={'ca': 'Amazon RDS'} # Ensure you have the CA bundle for SSL
connection
    )

    with connection.cursor() as cursor:
        cursor.execute('SELECT %s + %s AS sum', (3, 2))
        result = cursor.fetchone()

    return result

except Exception as e:
    return (f"Error: {str(e)}") # Return an error message if an exception
occurs
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite Ruby.

```
# Ruby code here.

require 'aws-sdk-rds'
require 'json'
require 'mysql2'

def lambda_handler(event:, context:)
    endpoint = ENV['DBEndpoint'] # Add the endpoint without https"
```

```
port = ENV['Port']          # 3306
user = ENV['DBUser']
region = ENV['DBRegion']    # 'us-east-1'
db_name = ENV['DBName']

credentials = Aws::Credentials.new(
  ENV['AWS_ACCESS_KEY_ID'],
  ENV['AWS_SECRET_ACCESS_KEY'],
  ENV['AWS_SESSION_TOKEN']
)
rds_client = Aws::RDS::AuthTokenGenerator.new(
  region: region,
  credentials: credentials
)

token = rds_client.auth_token(
  endpoint: endpoint+ ':' + port,
  user_name: user,
  region: region
)

begin
  conn = Mysql2::Client.new(
    host: endpoint,
    username: user,
    password: token,
    port: port,
    database: db_name,
    sslca: '/var/task/global-bundle.pem',
    sslverify: true,
    enable_clear_text_plugin: true
  )
  a = 3
  b = 2
  result = conn.query("SELECT #{a} + #{b} AS sum").first['sum']
  puts result
  conn.close
  {
    statusCode: 200,
    body: result.to_json
  }
rescue => e
  puts "Database connection failed due to #{e}"
end
```

```
end
```

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Connessione a un database Amazon RDS in una funzione Lambda tramite Rust.

```
use aws_config::BehaviorVersion;
use aws_credential_types::provider::ProvideCredentials;
use aws_sigv4::{
    http_request::{sign, SignableBody, SignableRequest, SigningSettings},
    sign::v4,
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
use sqlx::postgres::PgConnectOptions;
use std::env;
use std::time::{Duration, SystemTime};

const RDS_CERTS: &[u8] = include_bytes!("global-bundle.pem");

async fn generate_rds_iam_token(
    db_hostname: &str,
    port: u16,
    db_username: &str,
) -> Result<String, Error> {
    let config = aws_config::load_defaults(BehaviorVersion::v2024_03_28()).await;

    let credentials = config
        .credentials_provider()
        .expect("no credentials provider found")
        .provide_credentials()
        .await
        .expect("unable to load credentials");
    let identity = credentials.into();
```

```

    let region = config.region().unwrap().to_string();

    let mut signing_settings = SigningSettings::default();
    signing_settings.expires_in = Some(Duration::from_secs(900));
    signing_settings.signature_location =
aws_sigv4::http_request::SignatureLocation::QueryParams;

    let signing_params = v4::SigningParams::builder()
        .identity(&identity)
        .region(&region)
        .name("rds-db")
        .time(SystemTime::now())
        .settings(signing_settings)
        .build()?;

    let url = format!(
        "https://{db_hostname}:{port}/?Action=connect&DBUser={db_user}",
        db_hostname = db_hostname,
        port = port,
        db_user = db_username
    );

    let signable_request =
        SignableRequest::new("GET", &url, std::iter::empty(),
SignableBody::Bytes(&[]))
        .expect("signable request");

    let (signing_instructions, _signature) =
        sign(signable_request, &signing_params.into())?.into_parts();

    let mut url = url::Url::parse(&url).unwrap();
    for (name, value) in signing_instructions.params() {
        url.query_pairs_mut().append_pair(name, &value);
    }

    let response = url.to_string().split_off("https://".len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(handler)).await
}

```

```
async fn handler(_event: LambdaEvent<Value>) -> Result<Value, Error> {
    let db_host = env::var("DB_HOSTNAME").expect("DB_HOSTNAME must be set");
    let db_port = env::var("DB_PORT")
        .expect("DB_PORT must be set")
        .parse::<u16>()
        .expect("PORT must be a valid number");
    let db_name = env::var("DB_NAME").expect("DB_NAME must be set");
    let db_user_name = env::var("DB_USERNAME").expect("DB_USERNAME must be set");

    let token = generate_rds_iam_token(&db_host, db_port, &db_user_name).await?;

    let opts = PgConnectOptions::new()
        .host(&db_host)
        .port(db_port)
        .username(&db_user_name)
        .password(&token)
        .database(&db_name)
        .ssl_root_cert_from_pem(RDS_CERTS.to_vec())
        .ssl_mode(sqlx::postgres::PgSslMode::Require);

    let pool = sqlx::postgres::PgPoolOptions::new()
        .connect_with(opts)
        .await?;

    let result: i32 = sqlx::query_scalar("SELECT $1 + $2")
        .bind(3)
        .bind(2)
        .fetch_one(&pool)
        .await?;

    println!("Result: {:?}", result);

    Ok(json!({
        "statusCode": 200,
        "content-type": "text/plain",
        "body": format!("The selected sum is: {result}")
    })))
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Invocare una funzione Lambda da un trigger Kinesis

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso Kinesis. La funzione recupera il payload Kinesis, lo decodifica da Base64 e registra il contenuto del record.

.NET

SDK per .NET

### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
```

```
{
    if (evnt.Records.Count == 0)
    {
        Logger.LogInformation("Empty Kinesis Event received");
        return;
    }

    foreach (var record in evnt.Records)
    {
        try
        {
            Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
            string data = await GetRecordDataAsync(record.Kinesis, context);
            Logger.LogInformation($"Data: {data}");
            // TODO: Do interesting work based on the new data
        }
        catch (Exception ex)
        {
            Logger.LogError($"An error occurred {ex.Message}");
            throw;
        }
    }
    Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
}

private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
{
    byte[] bytes = record.Data.ToArray();
    string data = Encoding.UTF8.GetString(bytes);
    await Task.CompletedTask; //Placeholder for actual async work
    return data;
}
}
```



## Go

## SDK per Go V2

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Utilizzo di un evento Kinesis con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "log"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
    if len(kinesisEvent.Records) == 0 {
        log.Printf("empty Kinesis event received")
        return nil
    }

    for _, record := range kinesisEvent.Records {
        log.Printf("processed Kinesis event with EventId: %v", record.EventID)
        recordDataBytes := record.Kinesis.Data
        recordDataText := string(recordDataBytes)
        log.Printf("record data: %v", recordDataText)
        // TODO: Do interesting work based on the new data
    }
    log.Printf("successfully processed %v records", len(kinesisEvent.Records))
    return nil
}

func main() {
    lambda.Start(handler)
}
```

```
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

public class Handler implements RequestHandler<KinesisEvent, Void> {
    @Override
    public Void handleRequest(final KinesisEvent event, final Context context) {
        LambdaLogger logger = context.getLogger();
        if (event.getRecords().isEmpty()) {
            logger.log("Empty Kinesis Event received");
            return null;
        }
        for (KinesisEvent.KinesisEventRecord record : event.getRecords()) {
            try {
                logger.log("Processed Event with EventId: "+record.getEventID());
                String data = new String(record.getKinesis().getData().array());
                logger.log("Data:"+ data);
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex) {
                logger.log("An error occurred:"+ex.getMessage());
                throw ex;
            }
        }
    }
}
```

```

        }
    }
    logger.log("Successfully processed:"+event.getRecords().size()+"
records");
    return null;
}
}

```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Consumo di un evento Kinesis con Lambda utilizzando. JavaScript

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    for (const record of event.Records) {
        try {
            console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
            const recordData = await getRecordDataAsync(record.kinesis);
            console.log(`Record Data: ${recordData}`);
            // TODO: Do interesting work based on the new data
        } catch (err) {
            console.error(`An error occurred ${err}`);
            throw err;
        }
    }
    console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
    var data = Buffer.from(payload.data, "base64").toString("utf-8");
    await Promise.resolve(1); //Placeholder for actual async work
}

```

```
    return data;
  }
```

## Consumo di un evento Kinesis con Lambda utilizzando TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
```

```
var data = Buffer.from(payload.data, "base64").toString("utf-8");
await Promise.resolve(1); //Placeholder for actual async work
return data;
}
```

## PHP

### SDK per PHP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
}
```

```

    */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
            marked as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);

```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite Python.

```

# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import base64
def lambda_handler(event, context):

    for record in event['Records']:
        try:
            print(f"Processed Kinesis Event - EventID: {record['eventID']}")

```

```

        record_data = base64.b64decode(record['kinesis']
['data']).decode('utf-8')
        print(f"Record Data: {record_data}")
        # TODO: Do interesting work based on the new data
    except Exception as e:
        print(f"An error occurred {e}")
        raise e
    print(f"Successfully processed {len(event['Records'])} records.")

```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite Ruby.

```

# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue => err
      $stderr.puts "An error occurred #{err}"
      raise err
    end
  end
  puts "Successfully processed #{event['Records'].length} records."
end

def get_record_data_async(payload)

```

```

data = Base64.decode64(payload['data']).force_encoding('UTF-8')
# Placeholder for actual async work
# You can use Ruby's asynchronous programming tools like async/await or fibers
here.
return data
end

```

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Kinesis con Lambda tramite Rust.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error>
{
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
{}", record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
        }
    });
}

```



```

        Err(e) => {
            tracing::error!("Error: {}", e);
        }
    });

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
        time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Richiamare una funzione Lambda da un trigger DynamoDB

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso DynamoDB. La funzione recupera il payload DynamoDB e registra il contenuto del record.

## .NET

### SDK per .NET

#### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento DynamoDB con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext
context)
    {
        context.Logger.LogInformation($"Beginning to process
{dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

```
}  
}
```

Go

SDK per Go V2

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento DynamoDB con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
package main  
  
import (  
    "context"  
    "github.com/aws/aws-lambda-go/lambda"  
    "github.com/aws/aws-lambda-go/events"  
    "fmt"  
)  
  
func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,  
    error) {  
    if len(event.Records) == 0 {  
        return nil, fmt.Errorf("received empty event")  
    }  
  
    for _, record := range event.Records {  
        LogDynamoDBRecord(record)  
    }  
  
    message := fmt.Sprintf("Records processed: %d", len(event.Records))  
    return &message, nil  
}  
  
func main() {
```

```
lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
    fmt.Println(record.EventID)
    fmt.Println(record.EventName)
    fmt.Printf("%+v\n", record.Change)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento DynamoDB con Lambda tramite Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import
    com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
        GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
```

```
        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
GSON.toJson(record.getDynamodb()));
    }
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Consumo di un evento DynamoDB con Lambda utilizzando. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

### Consumo di un evento DynamoDB con Lambda utilizzando. TypeScript

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};
```

```
});  
}  
const logDynamoDBRecord = (record) => {  
  console.log(record.eventID);  
  console.log(record.eventName);  
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);  
};
```

## PHP

### SDK per PHP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento DynamoDB con Lambda tramite PHP.

```
<?php  
  
# using bref/bref and bref/logger for simplicity  
  
use Bref\Context\Context;  
use Bref\Event\DynamoDb\DynamoDbEvent;  
use Bref\Event\DynamoDb\DynamoDbHandler;  
use Bref\Logger\StderrLogger;  
  
require __DIR__ . '/vendor/autoload.php';  
  
class Handler extends DynamoDbHandler  
{  
  private StderrLogger $logger;  
  
  public function __construct(StderrLogger $logger)  
  {  
    $this->logger = $logger;  
  }  
  
  /**  
   * @throws JsonException
```

```

    * @throws \Bref\Event\InvalidLambdaEvent
    */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:". $eventName. "\n");
            $this->logger->info("Keys:". json_encode($keys). "\n");
            $this->logger->info("Old Image:". json_encode($old). "\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
            marked as failed
        }

        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords items");
    }
}

$logger = new StderrLogger();
return new Handler($logger);

```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Utilizzo di un evento DynamoDB con Lambda tramite Python.

```
import json

def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Utilizzo di un evento DynamoDB con Lambda tramite Ruby.

```
def lambda_handler(event:, context:)
  return 'received empty event' if event['Records'].empty?

  event['Records'].each do |record|
    log_dynamodb_record(record)
  end

  "Records processed: #{event['Records'].length}"
end

def log_dynamodb_record(record)
```



```
puts record['eventID']
puts record['eventName']
puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
end
```

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento DynamoDB con Lambda tramite Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
};

// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features = ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) -> Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}", records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }
}
```

```

    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");

    // Prepare the response
    Ok(())

}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())
}

```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Richiamare una funzione Lambda da un trigger Amazon DocumentDB

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso di modifiche di DocumentDB. La funzione recupera il payload DocumentDB e registra il contenuto del record.

.NET

SDK per .NET

### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite .NET.

```
using Amazon.Lambda.Core;
using System.Text.Json;
using System;
using System.Collections.Generic;
using System.Text.Json.Serialization;
//Assembly attribute to enable the Lambda function's JSON input to be converted
//into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace LambdaDocDb;

public class Function
{
    /// <summary>
    /// Lambda function entry point to process Amazon DocumentDB events.
    /// </summary>
    /// <param name="event">The Amazon DocumentDB event.</param>
    /// <param name="context">The Lambda context object.</param>
    /// <returns>A string to indicate successful processing.</returns>
    public string FunctionHandler(Event evnt, ILambdaContext context)
    {
```

```

        foreach (var record in evnt.Events)
        {
            ProcessDocumentDBEvent(record, context);
        }

        return "OK";
    }

    private void ProcessDocumentDBEvent(DocumentDBEventRecord record,
    ILambdaContext context)
    {

        var eventData = record.Event;
        var operationType = eventData.OperationType;
        var databaseName = eventData.Ns.Db;
        var collectionName = eventData.Ns.Coll;
        var fullDocument = JsonSerializer.Serialize(eventData.FullDocument, new
    JsonSerializerOptions { WriteIndented = true });

        context.Logger.LogLine($"Operation type: {operationType}");
        context.Logger.LogLine($"Database: {databaseName}");
        context.Logger.LogLine($"Collection: {collectionName}");
        context.Logger.LogLine($"Full document:\n{fullDocument}");
    }

    public class Event
    {
        [JsonPropertyName("eventSourceArn")]
        public string EventSourceArn { get; set; }

        [JsonPropertyName("events")]
        public List<DocumentDBEventRecord> Events { get; set; }

        [JsonPropertyName("eventSource")]
        public string EventSource { get; set; }
    }

    public class DocumentDBEventRecord
    {
        [JsonPropertyName("event")]
        public EventData Event { get; set; }
    }

```

```
public class EventData
{
    [JsonPropertyName("_id")]
    public IdData Id { get; set; }

    [JsonPropertyName("clusterTime")]
    public ClusterTime ClusterTime { get; set; }

    [JsonPropertyName("documentKey")]
    public DocumentKey DocumentKey { get; set; }

    [JsonPropertyName("fullDocument")]
    public Dictionary<string, object> FullDocument { get; set; }

    [JsonPropertyName("ns")]
    public Namespace Namespace { get; set; }

    [JsonPropertyName("operationType")]
    public string OperationType { get; set; }
}

public class IdData
{
    [JsonPropertyName("_data")]
    public string Data { get; set; }
}

public class ClusterTime
{
    [JsonPropertyName("$timestamp")]
    public Timestamp Timestamp { get; set; }
}

public class Timestamp
{
    [JsonPropertyName("t")]
    public long T { get; set; }

    [JsonPropertyName("i")]
    public int I { get; set; }
}

public class DocumentKey
```

```
{
    [JsonPropertyName("_id")]
    public Id Id { get; set; }
}

public class Id
{
    [JsonPropertyName("$oid")]
    public string Oid { get; set; }
}

public class Namespace
{
    [JsonPropertyName("db")]
    public string Db { get; set; }

    [JsonPropertyName("coll")]
    public string Coll { get; set; }
}
}
```

## Go

### SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite Go.

```
package main

import (
    "context"
    "encoding/json"
    "fmt"
```

```
"github.com/aws/aws-lambda-go/lambda"
)

type Event struct {
    Events []Record `json:"events"`
}

type Record struct {
    Event struct {
        OperationType string `json:"operationType"`
        NS             struct {
            DB    string `json:"db"`
            Coll string `json:"coll"`
        } `json:"ns"`
        FullDocument interface{} `json:"fullDocument"`
    } `json:"event"`
}

func main() {
    lambda.Start(handler)
}

func handler(ctx context.Context, event Event) (string, error) {
    fmt.Println("Loading function")
    for _, record := range event.Events {
        logDocumentDBEvent(record)
    }

    return "OK", nil
}

func logDocumentDBEvent(record Record) {
    fmt.Printf("Operation type: %s\n", record.Event.OperationType)
    fmt.Printf("db: %s\n", record.Event.NS.DB)
    fmt.Printf("collection: %s\n", record.Event.NS.Coll)
    docBytes, _ := json.MarshalIndent(record.Event.FullDocument, "", " ")
    fmt.Printf("Full document: %s\n", string(docBytes))
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite Java.

```
import java.util.List;
import java.util.Map;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class Example implements RequestHandler<Map<String, Object>, String> {

    @SuppressWarnings("unchecked")
    @Override
    public String handleRequest(Map<String, Object> event, Context context) {
        List<Map<String, Object>> events = (List<Map<String, Object>>)
event.get("events");
        for (Map<String, Object> record : events) {
            Map<String, Object> eventData = (Map<String, Object>)
record.get("event");
            processEventData(eventData);
        }

        return "OK";
    }

    @SuppressWarnings("unchecked")
    private void processEventData(Map<String, Object> eventData) {
        String operationType = (String) eventData.get("operationType");
        System.out.println("operationType: %s".formatted(operationType));

        Map<String, Object> ns = (Map<String, Object>) eventData.get("ns");

        String db = (String) ns.get("db");
        System.out.println("db: %s".formatted(db));
    }
}
```



```
String coll = (String) ns.get("coll");
System.out.println("coll: %s".formatted(coll));

Map<String, Object> fullDocument = (Map<String, Object>)
eventData.get("fullDocument");
System.out.println("fullDocument: %s".formatted(fullDocument));
}
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Consumo di un evento Amazon DocumentDB con Lambda utilizzando. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
  event.events.forEach(record => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record) => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
2));
};
```

## Consumo di un evento Amazon DocumentDB con Lambda utilizzando TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-lambda';

console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null, 2));
};
```

## PHP

### SDK per PHP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite PHP.

```
<?php

require __DIR__.'./vendor/autoload.php';
```

```
use Bref\Context\Context;
use Bref\Event\Handler;

class DocumentDBEventHandler implements Handler
{
    public function handle($event, Context $context): string
    {
        $events = $event['events'] ?? [];
        foreach ($events as $record) {
            $this->logDocumentDBEvent($record['event']);
        }
        return 'OK';
    }

    private function logDocumentDBEvent($event): void
    {
        // Extract information from the event record

        $operationType = $event['operationType'] ?? 'Unknown';
        $db = $event['ns']['db'] ?? 'Unknown';
        $collection = $event['ns']['coll'] ?? 'Unknown';
        $fullDocument = $event['fullDocument'] ?? [];

        // Log the event details

        echo "Operation type: $operationType\n";
        echo "Database: $db\n";
        echo "Collection: $collection\n";
        echo "Full document: " . json_encode($fullDocument, JSON_PRETTY_PRINT) .
        "\n";
    }
}

return new DocumentDBEventHandler();
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon DocumentDB con Lambda tramite Python.

```
import json

def lambda_handler(event, context):
    for record in event.get('events', []):
        log_document_db_event(record)
    return 'OK'

def log_document_db_event(record):
    event_data = record.get('event', {})
    operation_type = event_data.get('operationType', 'Unknown')
    db = event_data.get('ns', {}).get('db', 'Unknown')
    collection = event_data.get('ns', {}).get('coll', 'Unknown')
    full_document = event_data.get('fullDocument', {})

    print(f"Operation type: {operation_type}")
    print(f"db: {db}")
    print(f"collection: {collection}")
    print("Full document:", json.dumps(full_document, indent=2))
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Utilizzo di un evento Amazon DocumentDB con Lambda tramite Ruby.

```
require 'json'

def lambda_handler(event:, context:)
  event['events'].each do |record|
    log_document_db_event(record)
  end
  'OK'
end

def log_document_db_event(record)
  event_data = record['event'] || {}
  operation_type = event_data['operationType'] || 'Unknown'
  db = event_data.dig('ns', 'db') || 'Unknown'
  collection = event_data.dig('ns', 'coll') || 'Unknown'
  full_document = event_data['fullDocument'] || {}

  puts "Operation type: #{operation_type}"
  puts "db: #{db}"
  puts "collection: #{collection}"
  puts "Full document: #{JSON.pretty_generate(full_document)}"
end
```

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Utilizzo di un evento Amazon DocumentDB con Lambda tramite Ruby.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
  event::documentdb::{DocumentDbEvent, DocumentDbInnerEvent},
};
```

```
// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
  ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<DocumentDbEvent>) -> Result<(),
  Error> {

    tracing::info!("Event Source ARN: {:?}", event.payload.event_source_arn);
    tracing::info!("Event Source: {:?}", event.payload.event_source);

    let records = &event.payload.events;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_document_db_event(record);
    }

    tracing::info!("Document db records processed");

    // Prepare the response
    Ok(())
}

fn log_document_db_event(record: &DocumentDbInnerEvent)-> Result<(), Error>{
    tracing::info!("Change Event: {:?}", record.event);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
```

```
tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

let func = service_fn(function_handler);
lambda_runtime::run(func).await?;
Ok(())
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Invocare una funzione Lambda da un trigger Amazon MSK

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un cluster Amazon MSK. La funzione recupera il payload MSK e registra il contenuto del record.

### .NET

#### SDK per .NET

##### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite .NET.

```
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KafkaEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeria

namespace MSKLambda;

public class Function
{

    /// <param name="input">The event for the Lambda function handler to
    process.</param>
    /// <param name="context">The ILambdaContext that provides methods for
    logging and describing the Lambda environment.</param>
    /// <returns></returns>
    public void FunctionHandler(KafkaEvent evnt, ILambdaContext context)
    {

        foreach (var record in evnt.Records)
        {
            Console.WriteLine("Key:" + record.Key);
            foreach (var eventRecord in record.Value)
            {
                var valueBytes = eventRecord.Value.ToArray();
                var valueText = Encoding.UTF8.GetString(valueBytes);

                Console.WriteLine("Message:" + valueText);
            }
        }
    }
}
```



## Go

### SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite Go.

```
package main

import (
    "encoding/base64"
    "fmt"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.KafkaEvent) {
    for key, records := range event.Records {
        fmt.Println("Key:", key)

        for _, record := range records {
            fmt.Println("Record:", record)

            decodedValue, _ := base64.StdEncoding.DecodeString(record.Value)
            message := string(decodedValue)
            fmt.Println("Message:", message)
        }
    }
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent.KafkaEventRecord;

import java.util.Base64;
import java.util.Map;

public class Example implements RequestHandler<KafkaEvent, Void> {

    @Override
    public Void handleRequest(KafkaEvent event, Context context) {
        for (Map.Entry<String, java.util.List<KafkaEventRecord>> entry :
event.getRecords().entrySet()) {
            String key = entry.getKey();
            System.out.println("Key: " + key);

            for (KafkaEventRecord record : entry.getValue()) {
                System.out.println("Record: " + record);

                byte[] value = Base64.getDecoder().decode(record.getValue());
                String message = new String(value);
                System.out.println("Message: " + message);
            }
        }

        return null;
    }
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Consumo di un evento Amazon MSK con JavaScript Lambda utilizzando.

```
exports.handler = async (event) => {
  // Iterate through keys
  for (let key in event.records) {
    console.log('Key: ', key)
    // Iterate through records
    event.records[key].map((record) => {
      console.log('Record: ', record)
      // Decode base64
      const msg = Buffer.from(record.value, 'base64').toString()
      console.log('Message:', msg)
    })
  }
}
```

Consumo di un evento Amazon MSK con TypeScript Lambda utilizzando.

```
import { MSKEvent, Context } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "msk-handler-sample",
});

export const handler = async (
```

```
event: MSKEvent,
context: Context
): Promise<void> => {
  for (const [topic, topicRecords] of Object.entries(event.records)) {
    logger.info(`Processing key: ${topic}`);

    // Process each record in the partition
    for (const record of topicRecords) {
      try {
        // Decode the message value from base64
        const decodedMessage = Buffer.from(record.value, 'base64').toString();

        logger.info({
          message: decodedMessage
        });
      }
      catch (error) {
        logger.error('Error processing event', { error });
        throw error;
      }
    }
  }
}
```

## PHP

### SDK per PHP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite PHP.

```
<?php
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

// using bref/bref and bref/logger for simplicity
```

```
use Bref\Context\Context;
use Bref\Event\Kafka\KafkaEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): void
    {
        $kafkaEvent = new KafkaEvent($event);
        $this->logger->info("Processing records");
        $records = $kafkaEvent->getRecords();

        foreach ($records as $record) {
            try {
                $key = $record->getKey();
                $this->logger->info("Key: $key");

                $values = $record->getValue();
                $this->logger->info(json_encode($values));

                foreach ($values as $value) {
                    $this->logger->info("Value: $value");
                }
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}
```

```
    }  
}  
  
$logger = new StderrLogger();  
return new Handler($logger);
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite Python.

```
import base64  
  
def lambda_handler(event, context):  
    # Iterate through keys  
    for key in event['records']:  
        print('Key:', key)  
        # Iterate through records  
        for record in event['records'][key]:  
            print('Record:', record)  
            # Decode base64  
            msg = base64.b64decode(record['value']).decode('utf-8')  
            print('Message:', msg)
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento Amazon MSK con Lambda tramite Ruby.

```
require 'base64'

def lambda_handler(event:, context:)
  # Iterate through keys
  event['records'].each do |key, records|
    puts "Key: #{key}"

    # Iterate through records
    records.each do |record|
      puts "Record: #{record}"

      # Decode base64
      msg = Base64.decode64(record['value'])
      puts "Message: #{msg}"
    end
  end
end
```

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Utilizzo di un evento Amazon MSK con Lambda tramite Rust.

```

use aws_lambda_events::event::kafka::KafkaEvent;
use lambda_runtime::{run, service_fn, tracing, Error, LambdaEvent};
use base64::prelude::*;
use serde_json::{Value};
use tracing::{info};

/// Pre-Requisites:
/// 1. Install Cargo Lambda - see https://www.cargo-lambda.info/guide/getting-started.html
/// 2. Add packages tracing, tracing-subscriber, serde_json, base64
///
/// This is the main body for the function.
/// Write your code inside it.
/// There are some code example in the following URLs:
/// - https://github.com/aws-labs/aws-lambda-rust-runtime/tree/main/examples
/// - https://github.com/aws-samples/serverless-rust-demo/

async fn function_handler(event: LambdaEvent<KafkaEvent>) -> Result<Value, Error>
{
    let payload = event.payload.records;

    for (_name, records) in payload.iter() {

        for record in records {

            let record_text = record.value.as_ref().ok_or("Value is None")?;
            info!("Record: {}", &record_text);

            // perform Base64 decoding
            let record_bytes = BASE64_STANDARD.decode(record_text)?;
            let message = std::str::from_utf8(&record_bytes)?;

            info!("Message: {}", message);
        }

    }

    Ok(()).into()
}

#[tokio::main]
```



```
async fn main() -> Result<(), Error> {  
  
    // required to enable CloudWatch error logging by the runtime  
    tracing::init_default_subscriber();  
    info!("Setup CW subscriber!");  
  
    run(service_fn(function_handler)).await  
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Invocazione di una funzione Lambda da un trigger Amazon S3

I seguenti esempi di codice mostrano come implementare una funzione Lambda che riceve un evento attivato dal caricamento di un oggetto in un bucket S3. La funzione recupera il nome del bucket S3 e la chiave dell'oggetto dal parametro evento e chiama l'API Amazon S3 per recuperare e registrare il tipo di contenuto dell'oggetto.

.NET

SDK per .NET

### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
using System.Threading.Tasks;  
using Amazon.Lambda.Core;  
using Amazon.S3;  
using System;  
using Amazon.Lambda.S3Events;  
using System.Web;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
                var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

                context.Logger.LogLine($"Request is for {bucket} and {key}");

                var objectResult = await _s3Client.GetObjectAsync(bucket, key);

                context.Logger.LogLine($"Returning {objectResult.Key}");

                return objectResult.Key;
            }
            catch (Exception e)
            {
            }
        }
    }
}
```

```

        context.Logger.LogLine($"Error processing request -
        {e.Message}");

        return string.Empty;
    }
}
}
}
}

```

## Go

### SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite Go.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "log"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

func handler(ctx context.Context, s3Event events.S3Event) error {
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        log.Printf("failed to load default config: %s", err)
        return err
    }
    s3Client := s3.NewFromConfig(sdkConfig)

```

```
for _, record := range s3Event.Records {
    bucket := record.S3.Bucket.Name
    key := record.S3.Object.URLDecodedKey
    headOutput, err := s3Client.HeadObject(ctx, &s3.HeadObjectInput{
        Bucket: &bucket,
        Key:     &key,
    })
    if err != nil {
        log.Printf("error getting head of object %s/%s: %s", bucket, key, err)
        return err
    }
    log.Printf("successfully retrieved %s/%s of type %s", bucket, key,
        *headOutput.ContentType)
}

return nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import software.amazon.awssdk.services.s3.model.HeadObjectRequest;
import software.amazon.awssdk.services.s3.model.HeadObjectResponse;
```

```

import software.amazon.awssdk.services.s3.S3Client;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;
import
    com.amazonaws.services.lambda.runtime.events.models.s3.S3EventNotification.S3EventNotifi

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Handler implements RequestHandler<S3Event, String> {
    private static final Logger logger = LoggerFactory.getLogger(Handler.class);
    @Override
    public String handleRequest(S3Event s3event, Context context) {
        try {
            S3EventNotificationRecord record = s3event.getRecords().get(0);
            String srcBucket = record.getS3().getBucket().getName();
            String srcKey = record.getS3().getObject().getUrlDecodedKey();

            S3Client s3Client = S3Client.builder().build();
            HeadObjectResponse headObject = getHeadObject(s3Client, srcBucket,
srcKey);

            logger.info("Successfully retrieved " + srcBucket + "/" + srcKey + " of
type " + headObject.contentType());

            return "Ok";
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    private HeadObjectResponse getHeadObject(S3Client s3Client, String bucket,
String key) {
        HeadObjectRequest headObjectRequest = HeadObjectRequest.builder()
            .bucket(bucket)
            .key(key)
            .build();
        return s3Client.headObject(headObjectRequest);
    }
}

```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Consumo di un evento S3 con JavaScript Lambda utilizzando.

```
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

export const handler = async (event, context) => {

    // Get the object from the event and show its content type
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));

    try {
        const { ContentType } = await client.send(new HeadObjectCommand({
            Bucket: bucket,
            Key: key,
        }));

        console.log('CONTENT TYPE:', ContentType);
        return ContentType;

    } catch (err) {
        console.log(err);
        const message = `Error getting object ${key} from bucket ${bucket}. Make sure they exist and your bucket is in the same region as this function.`;
        console.log(message);
        throw new Error(message);
    }
};
```

## Consumo di un evento S3 con TypeScript Lambda utilizzando.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

## PHP

### SDK per PHP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Utilizzo di un evento S3 con Lambda tramite PHP.

```
<?php

use Bref\Context\Context;
use Bref\Event\S3\S3Event;
use Bref\Event\S3\S3Handler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends S3Handler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    public function handleS3(S3Event $event, Context $context) : void
    {
        $this->logger->info("Processing S3 records");

        // Get the object from the event and show its content type
        $records = $event->getRecords();

        foreach ($records as $record)
        {
            $bucket = $record->getBucket()->getName();
            $key = urldecode($record->getObject()->getKey());

            try {
                $fileSize = urldecode($record->getObject()->getSize());
                echo "File Size: " . $fileSize . "\n";
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                echo $e->getMessage() . "\n";
                echo 'Error getting object ' . $key . ' from bucket ' .
                $bucket . '. Make sure they exist and your bucket is in the same region as this
                function.' . "\n";
                throw $e;
            }
        }
    }
}
```



```
    }  
}  
  
$logger = new StderrLogger();  
return new Handler($logger);
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
import json  
import urllib.parse  
import boto3  
  
print('Loading function')  
  
s3 = boto3.client('s3')  
  
def lambda_handler(event, context):  
    #print("Received event: " + json.dumps(event, indent=2))  
  
    # Get the object from the event and show its content type  
    bucket = event['Records'][0]['s3']['bucket']['name']  
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'],  
    encoding='utf-8')  
    try:  
        response = s3.get_object(Bucket=bucket, Key=key)  
        print("CONTENT TYPE: " + response['ContentType'])  
        return response['ContentType']  
    except Exception as e:
```

```
print(e)
print('Error getting object {} from bucket {}. Make sure they exist and
your bucket is in the same region as this function.'.format(key, bucket))
raise e
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite Ruby.

```
require 'json'
require 'uri'
require 'aws-sdk'

puts 'Loading function'

def lambda_handler(event:, context:)
  s3 = Aws::S3::Client.new(region: 'region') # Your AWS region
  # puts "Received event: #{JSON.dump(event)}"

  # Get the object from the event and show its content type
  bucket = event['Records'][0]['s3']['bucket']['name']
  key = URI.decode_www_form_component(event['Records'][0]['s3']['object']['key'],
  Encoding::UTF_8)
  begin
    response = s3.get_object(bucket: bucket, key: key)
    puts "CONTENT TYPE: #{response.content_type}"
    return response.content_type
  rescue StandardError => e
    puts e.message
    puts "Error getting object #{key} from bucket #{bucket}. Make sure they exist
and your bucket is in the same region as this function."
    raise e
  end
end
```

```
end
```

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento S3 con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::s3::S3Event;
use aws_sdk_s3::{Client};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Main function
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    // Initialize the AWS SDK for Rust
    let config = aws_config::load_from_env().await;
    let s3_client = Client::new(&config);

    let res = run(service_fn(|request: LambdaEvent<S3Event>| {
        function_handler(&s3_client, request)
    })).await;

    res
}
```

```
async fn function_handler(
    s3_client: &Client,
    evt: LambdaEvent<S3Event>
) -> Result<(), Error> {
    tracing::info!(records = ?evt.payload.records.len(), "Received request from
    SQS");

    if evt.payload.records.len() == 0 {
        tracing::info!("Empty S3 event received");
    }

    let bucket = evt.payload.records[0].s3.bucket.name.as_ref().expect("Bucket
    name to exist");
    let key = evt.payload.records[0].s3.object.key.as_ref().expect("Object key to
    exist");

    tracing::info!("Request is for {} and object {}", bucket, key);

    let s3_get_object_result = s3_client
        .get_object()
        .bucket(bucket)
        .key(key)
        .send()
        .await;

    match s3_get_object_result {
        Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
        contains a 'body' property of type ByteStream"),
        Err(_) => tracing::info!("Failure with S3 Get Object request")
    }

    Ok(())
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Invocazione di una funzione Lambda da un trigger Amazon SNS

I seguenti esempi di codice mostrano come implementare una funzione Lambda che riceve un evento attivato dal ricevimento di messaggi da un argomento SNS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

.NET

SDK per .NET

### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }
}
```

```
private async Task ProcessRecordAsync(SNSEvent.SNSRecord record,
ILambdaContext context)
{
    try
    {
        context.Logger.LogInformation($"Processed record
{record.Sns.Message}");

        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
    catch (Exception e)
    {
        //You can use Dead Letter Queue to handle failures. By configuring a
        Lambda DLQ.
        context.Logger.LogError($"An error occurred");
        throw;
    }
}
}
```

Go

SDK per Go V2

 Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-lambda-go/events"
```

```
"github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
    for _, record := range snsEvent.Records {
        processMessage(record)
    }
    fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
    message := record.SNS.Message
    fmt.Printf("Processed message: %s\n", message)
    // TODO: Process your record here
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import com.amazonaws.services.lambda.runtime.events.SNSEvent.SNSRecord;
```

```
import java.util.Iterator;
import java.util.List;

public class SNSEventHandler implements RequestHandler<SNSEvent, Boolean> {
    LambdaLogger logger;

    @Override
    public Boolean handleRequest(SNSEvent event, Context context) {
        logger = context.getLogger();
        List<SNSRecord> records = event.getRecords();
        if (!records.isEmpty()) {
            Iterator<SNSRecord> recordsIter = records.iterator();
            while (recordsIter.hasNext()) {
                processRecord(recordsIter.next());
            }
        }
        return Boolean.TRUE;
    }

    public void processRecord(SNSRecord record) {
        try {
            String message = record.getSNS().getMessage();
            logger.log("message: " + message);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```



## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Consumo di un evento SNS con JavaScript Lambda utilizzando.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consumo di un evento SNS con TypeScript Lambda utilizzando.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
```

```
    ): Promise<void> => {
      for (const record of event.Records) {
        await processMessageAsync(record);
      }
      console.info("done");
    };

    async function processMessageAsync(record: SNSEventRecord): Promise<any> {
      try {
        const message: string = JSON.stringify(record.Sns.Message);
        console.log(`Processed message ${message}`);
        await Promise.resolve(1); //Placeholder for actual async work
      } catch (err) {
        console.error("An error occurred");
        throw err;
      }
    }
  }
}
```

## PHP

### SDK per PHP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

/*
Since native PHP support for AWS Lambda is not available, we are utilizing Bref's
PHP functions runtime for AWS Lambda.
For more information on Bref's PHP runtime for Lambda, refer to: https://bref.sh/
docs/runtimes/function

Another approach would be to create a custom runtime.
```

```
A practical example can be found here: https://aws.amazon.com/blogs/apn/aws-lambda-custom-runtime-for-php-a-practical-example/  
*/  
  
// Additional composer packages may be required when using Bref or any other PHP  
// functions runtime.  
// require __DIR__ . '/vendor/autoload.php';  
  
use Bref\Context\Context;  
use Bref\Event\Sns\SnsEvent;  
use Bref\Event\Sns\SnsHandler;  
  
class Handler extends SnsHandler  
{  
    public function handleSns(SnsEvent $event, Context $context): void  
    {  
        foreach ($event->getRecords() as $record) {  
            $message = $record->getMessage();  
  
            // TODO: Implement your custom processing logic here  
            // Any exception thrown will be logged and the invocation will be  
            marked as failed  
  
            echo "Processed Message: $message" . PHP_EOL;  
        }  
    }  
}  
  
return new Handler();
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for record in event['Records']:
        process_message(record)
    print("done")

def process_message(record):
    try:
        message = record['Sns']['Message']
        print(f"Processed message {message}")
        # TODO; Process your record here

    except Exception as e:
        print("An error occurred")
        raise e
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
    event['Records'].map { |record| process_message(record) }
end

def process_message(record)
    message = record['Sns']['Message']
    puts("Processing message: #{message}")
rescue StandardError => e
    puts("Error processing message: #{e}")
```

```
    raise
end
```

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SNS con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
// aws_lambda_events = { version = "0.10.0", default-features = false, features
//   = ["sns"] }
// lambda_runtime = "0.8.1"
// tokio = { version = "1", features = ["macros"] }
// tracing = { version = "0.1", features = ["log"] }
// tracing-subscriber = { version = "0.3", default-features = false, features =
//   ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for record in event.payload.records {
        process_record(&record)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);
}
```

```
// Implement your record handling code here.

Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Richiamo di una funzione Lambda da un trigger Amazon SQS

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da una coda SQS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

### .NET

#### SDK per .NET

##### Note

C'è altro su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
            //Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

## Go

### SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
    for _, record := range event.Records {
        err := processMessage(record)
        if err != nil {
            return err
        }
    }
    fmt.Println("done")
    return nil
}

func processMessage(record events.SQSMessage) error {
    fmt.Printf("Processed message %s\n", record.Body)
    // TODO: Do interesting work based on the new message
    return nil
}

func main() {
    lambda.Start(handler)
}
```



## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class Function implements RequestHandler<SQSEvent, Void> {
    @Override
    public Void handleRequest(SQSEvent sqsEvent, Context context) {
        for (SQSMessage msg : sqsEvent.getRecords()) {
            processMessage(msg, context);
        }
        context.getLogger().log("done");
        return null;
    }

    private void processMessage(SQSMessage msg, Context context) {
        try {
            context.getLogger().log("Processed message " + msg.getBody());

            // TODO: Do interesting work based on the new message

        } catch (Exception e) {
            context.getLogger().log("An error occurred");
            throw e;
        }
    }
}
```

```
}  
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Consumo di un evento SQS con JavaScript Lambda utilizzando.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
exports.handler = async (event, context) => {  
  for (const message of event.Records) {  
    await processMessageAsync(message);  
  }  
  console.info("done");  
};  
  
async function processMessageAsync(message) {  
  try {  
    console.log(`Processed message ${message.body}`);  
    // TODO: Do interesting work based on the new message  
    await Promise.resolve(1); //Placeholder for actual async work  
  } catch (err) {  
    console.error("An error occurred");  
    throw err;  
  }  
}
```

Consumo di un evento SQS con TypeScript Lambda utilizzando.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";
```

```
export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

## PHP

### SDK per PHP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
```

```
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for message in event['Records']:
        process_message(message)
    print("done")

def process_message(message):
    try:
        print(f"Processed message {message['body']}")
        # TODO: Do interesting work based on the new message
    except Exception as err:
        print("An error occurred")
        raise err
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
    event['Records'].each do |message|
        process_message(message)
    end
    puts "done"
end

def process_message(message)
    begin
        puts "Processed message #{message['body']}"
        # TODO: Do interesting work based on the new message
    end
```

```
rescue StandardError => err
  puts "An error occurred"
  raise err
end
end
```

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Utilizzo di un evento SQS con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
        tracing::info!("Message body: {}",
            record.body.as_deref().unwrap_or_default())
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
        time.
        .without_time()
}
```

```
        .init();

        run(service_fn(function_handler)).await
    }
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

I seguenti esempi di codice spiegano come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da un flusso Kinesis. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

.NET

SDK per .NET

### Note

C'è altro su [GitHub](#) Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))
]
```

```

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
        ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed
item immediately.
                Lambda will immediately begin to retry processing from this
failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>
                {
                    new StreamsEventResponse.BatchItemFailure
{ ItemIdentifier = record.Kinesis.SequenceNumber }
                }
                };
            }
        }
    }
}

```



```

    }
    Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
    return new StreamsEventResponse();
}

private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
{
    byte[] bytes = record.Data.ToArray();
    string data = Encoding.UTF8.GetString(bytes);
    await Task.CompletedTask; //Placeholder for actual async work
    return data;
}
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
}

```

## Go

### SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Go.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

```

```
import (
    "context"
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
    (map[string]interface{}, error) {
    batchItemFailures := []map[string]interface{}{}

    for _, record := range kinesisEvent.Records {
        curRecordSequenceNumber := ""

        // Process your record
        if /* Your record processing condition here */ {
            curRecordSequenceNumber = record.Kinesis.SequenceNumber
        }

        // Add a condition to check if the record processing failed
        if curRecordSequenceNumber != "" {
            batchItemFailures = append(batchItemFailures, map[string]interface{}{
                "itemIdentifier": curRecordSequenceNumber})
        }
    }

    kinesisBatchResponse := map[string]interface{}{
        "batchItemFailures": batchItemFailures,
    }
    return kinesisBatchResponse, nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent input, Context
context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
ArrayList<>();
        String curRecordSequenceNumber = "";

        for (KinesisEvent.KinesisEventRecord kinesisEventRecord :
input.getRecords()) {
            try {
                //Process your record
                KinesisEvent.Record kinesisRecord =
kinesisEventRecord.getKinesis();
                curRecordSequenceNumber = kinesisRecord.getSequenceNumber();

            } catch (Exception e) {
```

```

        /* Since we are working with streams, we can return the failed
        item immediately.
           Lambda will immediately begin to retry processing from this
        failed item onwards. */
        batchItemFailures.add(new
StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
        return new StreamsEventResponse(batchItemFailures);
    }
}

return new StreamsEventResponse(batchItemFailures);
}
}

```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Javascript.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    for (const record of event.Records) {
        try {
            console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
            const recordData = await getRecordDataAsync(record.kinesis);
            console.log(`Record Data: ${recordData}`);
            // TODO: Do interesting work based on the new data
        } catch (err) {
            console.error(`An error occurred ${err}`);
            /* Since we are working with streams, we can return the failed item
            immediately.
               Lambda will immediately begin to retry processing from this failed
            item onwards. */

```

```

        return {
            batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
        };
    }
}
console.log(`Successfully processed ${event.Records.length} records.`);
return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
    var data = Buffer.from(payload.data, "base64").toString("utf-8");
    await Promise.resolve(1); //Placeholder for actual async work
    return data;
}

```

## Segnalazione degli errori degli elementi batch di Kinesis utilizzando Lambda. TypeScript

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
    KinesisStreamEvent,
    Context,
    KinesisStreamHandler,
    KinesisStreamRecordPayload,
    KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
    logLevel: "INFO",
    serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
    event: KinesisStreamEvent,
    context: Context
): Promise<KinesisStreamBatchResponse> => {
    for (const record of event.Records) {
        try {
            logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
            const recordData = await getRecordDataAsync(record.kinesis);

```

```

        logger.info(`Record Data: ${recordData}`);
        // TODO: Do interesting work based on the new data
    } catch (err) {
        logger.error(`An error occurred ${err}`);
        /* Since we are working with streams, we can return the failed item
        immediately.
           Lambda will immediately begin to retry processing from this failed
        item onwards. */
        return {
            batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
        };
    }
}
logger.info(`Successfully processed ${event.Records.length} records.`);
return { batchItemFailures: [] };
};

async function getRecordDataAsync(
    payload: KinesisStreamRecordPayload
): Promise<string> {
    var data = Buffer.from(payload.data, "base64").toString("utf-8");
    await Promise.resolve(1); //Placeholder for actual async work
    return data;
}

```

## PHP

### SDK per PHP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite PHP.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

```

```

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();

        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(

```

```

        fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
        $failedRecords
    );

    return [
        'batchItemFailures' => $failures
    ];
}

$logger = new StderrLogger();
return new Handler($logger);

```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Python.

```

# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["kinesis"]["sequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}

```



## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  batch_item_failures = []

  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue StandardError => err
      puts "An error occurred #{err}"
      # Since we are working with streams, we can return the failed item
      # immediately.
      # Lambda will immediately begin to retry processing from this failed item
      # onwards.
      return { batchItemFailures: [{ itemIdentifier: record['kinesis']
['sequenceNumber'] }] }
    end
  end

  puts "Successfully processed #{event['Records'].length} records."
  { batchItemFailures: batch_item_failures }
end
```

```
def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('utf-8')
  # Placeholder for actual async work
  sleep(1)
  data
end
```

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
    Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in &event.payload.records {
        tracing::info!(
            "EventId: {}",
```

```

        record.event_id.as_deref().unwrap_or_default()
    );

    let record_processing_result = process_record(record);

    if record_processing_result.is_err() {
        response.batch_item_failures.push(KinesisBatchItemFailure {
            item_identifier: record.kinesis.sequence_number.clone(),
        });
        /* Since we are working with streams, we can return the failed item
        immediately.
        Lambda will immediately begin to retry processing from this failed
        item onwards. */
        return Ok(response);
    }

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()

```

```
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB

I seguenti esempi di codice spiegano come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da un flusso DynamoDB. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

### .NET

#### SDK per .NET

##### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
        ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
        {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
        List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
                { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

```
}
```

## Go

### SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
    ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
    BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
(*BatchResult, error) {
    var batchItemFailures []BatchItemFailure
    curRecordSequenceNumber := ""

    for _, record := range event.Records {
        // Process your record
        curRecordSequenceNumber = record.Change.SequenceNumber
    }
}
```

```

if curRecordSequenceNumber != "" {
    batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
curRecordSequenceNumber})
}

batchResult := BatchResult{
    BatchItemFailures: batchItemFailures,
}

return &batchResult, nil
}

func main() {
    lambda.Start(HandleRequest)
}

```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite Java.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;

import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
StreamsEventResponse> {

```

```
@Override
public StreamsEventResponse handleRequest(DynamodbEvent input, Context
context) {

    List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
    ArrayList<>();
    String curRecordSequenceNumber = "";

    for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
    input.getRecords()) {
        try {
            //Process your record
            StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
            curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

        } catch (Exception e) {
            /* Since we are working with streams, we can return the failed
            item immediately.
            Lambda will immediately begin to retry processing from this
            failed item onwards. */
            batchItemFailures.add(new
            StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
            return new StreamsEventResponse(batchItemFailures);
        }
    }

    return new StreamsEventResponse();
}
```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).



## Segnalazione degli errori degli elementi batch di DynamoDB utilizzando Lambda. JavaScript

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

## Segnalazione degli errori degli elementi batch di DynamoDB utilizzando Lambda. TypeScript

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }
}
```

```
}

    return { batchItemFailures: batchItemFailures };
};
```

## PHP

### SDK per PHP

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
```

```
{
    $dynamoDbEvent = new DynamoDbEvent($event);
    $this->logger->info("Processing records");

    $records = $dynamoDbEvent->getRecords();
    $failedRecords = [];
    foreach ($records as $record) {
        try {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data
        } catch (Exception $e) {
            $this->logger->error($e->getMessage());
            // failed processing the record
            $failedRecords[] = $record->getSequenceNumber();
        }
    }
    $totalRecords = count($records);
    $this->logger->info("Successfully processed $totalRecords records");

    // change format for the response
    $failures = array_map(
        fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
        $failedRecords
    );

    return [
        'batchItemFailures' => $failures
    ];
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite Ruby.

```
def lambda_handler(event:, context:)
  records = event["Records"]
  cur_record_sequence_number = ""

  records.each do |record|
    begin
      # Process your record
      cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
      # Return failed record's sequence number
      return {"batchItemFailures" => [{"itemIdentifier" =>
cur_record_sequence_number}]}
    end
  end

  {"batchItemFailures" => []}
end
```

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite Rust.

```
use aws_lambda_events::{
  event::dynamodb::{Event, EventRecord, StreamRecord},
  streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
  let stream_record: &StreamRecord = &record.change;
```

```

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("").to_string(),
            });
            return Ok(response);
        }

        // Process your record here...
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
            immediately.
            Lambda will immediately begin to retry processing from this failed
            item onwards. */
            return Ok(response);
        }
    }
}

```

```
        tracing::info!("Successfully processed {} record(s)", records.len());

        Ok(response)
    }

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
        time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS

I seguenti esempi di codice spiegano come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da una coda SQS. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

### .NET

#### SDK per .NET

##### Note

C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

## Segnalazione di errori di elementi batch di SQS con Lambda tramite .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
        ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
        List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
                SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
    }
}
```



```

        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}

```

## Go

### SDK per Go V2

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite Go.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent)
(map[string]interface{}, error) {
    batchItemFailures := []map[string]interface{}{}

    for _, message := range sqsEvent.Records {
        if len(message.Body) > 0 {
            // Your message processing condition here
            fmt.Printf("Successfully processed message: %s\n", message.Body)
        } else {
            // Message processing failed
            fmt.Printf("Failed to process message %s\n", message.MessageId)
            batchItemFailures = append(batchItemFailures, map[string]interface{}{
                "itemIdentifier": message.MessageId})
        }
    }
}

```

```

    }
}

sqsBatchResponse := map[string]interface{}{
    "batchItemFailures": batchItemFailures,
}
return sqsBatchResponse, nil
}

func main() {
    lambda.Start(handler)
}

```

## Java

### SDK per Java 2.x

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Segnalazione di errori di elementi batch di SQS con Lambda tramite Java.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSBatchResponse;

import java.util.ArrayList;
import java.util.List;

public class ProcessSQSMessageBatch implements RequestHandler<SQSEvent,
    SQSBatchResponse> {
    @Override
    public SQSBatchResponse handleRequest(SQSEvent sqsEvent, Context context) {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
        ArrayList<SQSBatchResponse.BatchItemFailure>();
    }
}

```

```

    for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
        try {
            //process your message
        } catch (Exception e) {
            //Add failed message identifier to the batchItemFailures list
            batchItemFailures.add(new
SQSBatchResponse.BatchItemFailure(message.getMessageId()));
        }
    }
    return new SQSBatchResponse(batchItemFailures);
}
}

```

## JavaScript

### SDK per JavaScript (v3)

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Segnalazione degli errori degli elementi batch SQS con Lambda using. JavaScript

```

// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
    }
}

```

```
    }  
    console.log(`Processed message: ${record.body}`);  
  }  
}
```

## Segnalazione degli errori degli elementi batch SQS con Lambda using. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord }  
  from 'aws-lambda';  
  
export const handler = async (event: SQSEvent, context: Context):  
  Promise<SQSBatchResponse> => {  
  const batchItemFailures: SQSBatchItemFailure[] = [];  
  
  for (const record of event.Records) {  
    try {  
      await processMessageAsync(record);  
    } catch (error) {  
      batchItemFailures.push({ itemIdentifier: record.messageId });  
    }  
  }  
  
  return {batchItemFailures: batchItemFailures};  
};  
  
async function processMessageAsync(record: SQSRecord): Promise<void> {  
  if (record.body && record.body.includes("error")) {  
    throw new Error('There is an error in the SQS Message.');  }  
  console.log(`Processed message ${record.body}`);  
}
```

## PHP

## SDK per PHP

 Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();

        foreach ($records as $record) {
            try {
                // Assuming the SQS message is in JSON format
```

```

        $message = json_decode($record->getBody(), true);
        $this->logger->info(json_encode($message));
        // TODO: Implement your custom processing logic here
    } catch (Exception $e) {
        $this->logger->error($e->getMessage());
        // failed processing the record
        $this->markAsFailed($record);
    }
}
$totalRecords = count($records);
$this->logger->info("Successfully processed $totalRecords SQS records");
}
}

$logger = new StderrLogger();
return new Handler($logger);

```

## Python

### SDK per Python (Boto3)

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

### Segnalazione di errori di elementi batch di SQS con Lambda tramite Python.

```

# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event, context):
    if event:
        batch_item_failures = []
        sqs_batch_response = {}

        for record in event["Records"]:
            try:
                print(f"Processed message: {record['body']}")
            except Exception as e:

```

```
        batch_item_failures.append({"itemIdentifier":  
record['messageId']})  
  
    sqs_batch_response["batchItemFailures"] = batch_item_failures  
    return sqs_batch_response
```

## Ruby

### SDK per Ruby

#### Note

C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
require 'json'  
  
def lambda_handler(event:, context:)  
  if event  
    batch_item_failures = []  
    sqs_batch_response = {}  
  
    event["Records"].each do |record|  
      begin  
        # process message  
        rescue StandardError => e  
          batch_item_failures << {"itemIdentifier" => record['messageId']}  
        end  
      end  
  
      sqs_batch_response["batchItemFailures"] = batch_item_failures  
      return sqs_batch_response  
    end  
  end  
end
```

## Rust

### SDK per Rust

#### Note

C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di [Esempi serverless](#).

Segnalazione di errori di elementi batch di SQS con Lambda tramite Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) ->
    Result<SqsBatchResponse, Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
    })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
```



```
run(service_fn(function_handler)).await  
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

## AWS contributi della community per Lambda

AWS i contributi della community sono esempi che sono stati creati e gestiti da più team AWS. Per fornire un feedback, utilizza il meccanismo disponibile nei repository collegati.

### Esempi

- [Creare e testare un'applicazione serverless](#)

## Creare e testare un'applicazione serverless

Gli esempi di codice seguenti mostrano come creare e testare un'applicazione serverless utilizzando Gateway API con Lambda e DynamoDB

### .NET

#### SDK per .NET

Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per .NET.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su [GitHub](#).

#### Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

## Go

### SDK per Go V2

Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per Go.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

## Java

### SDK per Java 2.x

Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per Java.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

## Rust

### SDK per Rust

Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per Rust.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su [GitHub](#).

### Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta [Usare Lambda con un AWS SDK](#). Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

# Quote di Lambda

## Important

Account AWS Le nuove hanno ridotto la concorrenza e le quote di memoria per Lambda Functions e Lambda MicroVMS. AWS aumenta queste quote automaticamente in base all'utilizzo.

AWS Lambda è progettato per scalare rapidamente per soddisfare la domanda, consentendo alle funzioni di aumentare la scalabilità in base al traffico dell'applicazione. Lambda è progettato per attività di elaborazione di breve durata che non conservano o si basano sullo stato tra una chiamata e l'altra. Il codice può essere eseguito per un massimo di 15 minuti in una singola chiamata e una singola funzione può utilizzare fino a 10.240 MB di memoria.

È importante conoscere le barriere messe in atto per proteggere il tuo account e i carichi di lavoro degli altri clienti. Le quote di servizio sono previste in tutti i AWS servizi e sono costituite da limiti rigidi, che non è possibile modificare, e limiti flessibili, per i quali è possibile richiedere aumenti. Per impostazione predefinita, a tutti i nuovi account viene assegnato un profilo di quota che consente l'esplorazione dei AWS servizi.

Per visualizzare le quote applicabili al tuo account, vai al [pannello di controllo di Service Quotas](#). Qui puoi visualizzare le quote dei servizi, richiedere un aumento delle quote e visualizzare l'utilizzo corrente. Da qui, puoi visualizzare in dettaglio un AWS servizio specifico, come Lambda:

 **AWS Lambda**

AWS Lambda is a compute service that runs your code in response to events and automatically manages the compute resources for you.

Service quotas

View your applied quota values, default quota values, and request quota increases for quotas.

Search by quota name

Request increase at account level

< 1 >

| Quota name                                                                                                                                     | Applied account-level quota value | AWS default quota value | Utilization   | Adjustability  |
|------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|-------------------------|---------------|----------------|
| <input checked="" type="radio"/> <a href="#">Asynchronous payload</a>                                                                          | 256 kilobytes                     | 256 kilobytes           | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">Concurrency scaling rate</a>                                                                      | 1,000                             | 1,000                   | Not available | Not adjustable |
| <input type="radio"/> <a href="#">Concurrent executions</a>                                                                                    | 1,000                             | 1,000                   | 0             | Account level  |
| <input checked="" type="radio"/> <a href="#">Deployment package size (console editor)</a>                                                      | 3 megabytes                       | 3 megabytes             | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">Deployment package size (direct upload)</a>                                                       | 50 megabytes                      | 50 megabytes            | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">Deployment package size (unzipped)</a>                                                            | 250 megabytes                     | 250 megabytes           | Not available | Not adjustable |
| <input type="radio"/> <a href="#">Elastic network interfaces per VPC</a>                                                                       | Not available                     | 500                     | Not available | Account level  |
| <input checked="" type="radio"/> <a href="#">Environment variable size</a>                                                                     | 4 kilobytes                       | 4 kilobytes             | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">File descriptors</a>                                                                              | 1,024                             | 1,024                   | Not available | Not adjustable |
| <input type="radio"/> <a href="#">Function and layer storage</a>                                                                               | 75 gigabytes                      | 75 gigabytes            | Not available | Account level  |
| <input checked="" type="radio"/> <a href="#">Function layers</a>                                                                               | 5                                 | 5                       | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">Function resource-based policy</a>                                                                | 20 kilobytes                      | 20 kilobytes            | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">Function timeout</a>                                                                              | 900                               | 900                     | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">Processes and threads</a>                                                                         | 1,024                             | 1,024                   | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">Rate of control plane API requests (excludes invocation, GetFunction, and GetPolicy requests)</a> | 15                                | 15                      | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">Rate of GetFunction API requests</a>                                                              | 100                               | 100                     | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">Rate of GetPolicy API requests</a>                                                                | 15                                | 15                      | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">Synchronous payload</a>                                                                           | 6 megabytes                       | 6 megabytes             | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">Temporary storage</a>                                                                             | 512 megabytes                     | 512 megabytes           | Not available | Not adjustable |
| <input checked="" type="radio"/> <a href="#">Test events (console editor)</a>                                                                  | 10                                | 10                      | Not available | Not adjustable |

Le sezioni seguenti elencano le quote e i limiti predefiniti in Lambda per categoria.

Argomenti

- [Elaborazione e archiviazione](#)
- [Configurazione, implementazione ed esecuzione della funzione](#)
- [Richieste API Lambda](#)
- [Funzioni durevoli](#)
- [MicroVM Lambda](#)
- [Altri servizi](#)

## Elaborazione e archiviazione

Lambda imposta le quote per la quantità di risorse di calcolo e storage che è possibile utilizzare per eseguire e archiviare le funzioni. Le quote per le esecuzioni e l'archiviazione simultanee sono applicate in base alla Regione AWS. Le quote dell'interfaccia di rete elastica (ENI) sono applicate in base al cloud privato virtuale (VPC), indipendentemente dalla regione. Le seguenti quote possono essere aumentate rispetto ai relativi valori predefiniti. Per ulteriori informazioni, consulta [Richiesta di un aumento di quota](#) nella Guida per l'utente delle Service Quotas.

| Risorsa                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Quota predefinita                                       | Può essere aumentato fino a                                                                                       |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| Esecuzioni simultanee                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 1.000                                                   | Decine di migliaia                                                                                                |
| <p>Archiviazione per le funzioni caricate (archivi di file.zip) e i livelli utilizzando l' Lambda-managed archiviazione. Ogni versione di funzione e di livello consuma spazio di storage.</p> <p>Per evitare limiti di archiviazione, puoi configurare le funzioni e i livelli in modo da utilizzare invece l'archiviazione di codice S3 <a href="#">autogestita</a>.</p> <p>Per le best practice da seguire per la gestione dell'archiviazione del codice, consulta <a href="#">Monitoraggio dell'archiviazione del codice Lambda</a> in Serverless Land.</p> | 300 GB (decompresso)                                    | Non aumentabile. Usa lo storage di codice S3 <a href="#">autogestito</a> per l'archiviazione oltre questo limite. |
| Storage per le funzioni definite come immagini di container. Queste immagini sono memorizzate in Amazon ECR.                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Consulta <a href="#">Service Quotas di Amazon ECR</a> . |                                                                                                                   |
| <a href="#">Interfacce di rete elastiche per Virtual Private Cloud (VPC)</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 500                                                     | Migliaia                                                                                                          |

| Risorsa                                                                                                                                                                                                                                                                       | Quota predefinita | Può essere aumentato fino a |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-----------------------------|
| <div>  <b>Note</b><br/>           Questa quota è condivisa con altri servizi, ad esempio, Amazon Elastic File System (Amazon EFS). Consulta <a href="#">Quote Amazon VPC</a>.         </div> |                   |                             |

Per ulteriori informazioni sulla simultaneità e su come Lambda ridimensiona la simultaneità della funzione in risposta al traffico, consulta [Informazioni sulla scalabilità della funzione Lambda](#).

## Configurazione, implementazione ed esecuzione della funzione

Le seguenti quote si applicano alla configurazione, all'implementazione e all'esecuzione della funzione. Fatto salvo per quanto indicato, non possono essere modificate.

|                                                                                                                                                                                                                                                                           |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <div>  <b>Note</b><br/>           La documentazione Lambda, i messaggi di log e la console utilizzano l'abbreviazione MB (anziché MiB) per fare riferimento a 1.024 KB.         </div> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

| Risorsa                                               | Quota                                                                                                                                                                                                                                                                |
|-------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">Allocazione di memoria</a> della funzione | Da 128 MB a 10.240 MB, in incrementi di 1 MB.<br><br>Nota: Lambda alloca la potenza della CPU in proporzione alla quantità di memoria configurata. È possibile aumentare o diminuire la memoria e la potenza della CPU assegnate alla funzione utilizzando l'imposta |

| Risorsa                                                                     | Quota                                                                                                                                                                                                                                                          |
|-----------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                             | zione Memory (MB). A 1.769 MB, una funzione ha l'equivalente di una vCPU.                                                                                                                                                                                      |
| Timeout della funzione.                                                     | 900 secondi (15 minuti)                                                                                                                                                                                                                                        |
| <a href="#">Variabili di ambiente</a> della funzione                        | 4 KB, per tutte le variabili di ambiente associate alla funzione, in forma aggregata                                                                                                                                                                           |
| <a href="#">Policy basata sulle risorse</a> della funzione                  | 20 KB                                                                                                                                                                                                                                                          |
| <a href="#">Livelli</a> della funzione                                      | 5 livelli                                                                                                                                                                                                                                                      |
| <a href="#">Limite di dimensionamento della simultaneità</a> delle funzioni | Per ogni funzione, 1.000 ambienti di esecuzione ogni 10 secondi                                                                                                                                                                                                |
| <a href="#">Payload dell'invocazione</a> (richiesta e risposta)             | <p>6 MB ciascuno per richiesta e risposta (sincrono)</p> <p>200 MB per ogni risposta <a href="#">in streaming</a> (sincrona)</p> <p>1 MB (asincrono)</p> <p>1 MB per la dimensione totale combinata dei valori della riga di richiesta e dell'intestazione</p> |
| Larghezza di banda per le <a href="#">risposte in streaming</a>             | <p>Senza limite per i primi 6 MB di risposta della funzione</p> <p>Per risposte superiori a 6 MB, 2 MB/s per il resto della risposta</p>                                                                                                                       |



| Risorsa                                                                             | Quota                                                                                                                                                                                                                                                                                                                      |
|-------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Larghezza di banda di rete per ambiente di esecuzione                               | 625 Mbps. Per le funzioni non collegate a un VPC, è possibile richiedere un aumento tramite la console <a href="#">Service Quotas</a> . Dopo l'approvazione, la larghezza di banda della rete aumenta proporzionalmente alla memoria, partendo da 2.048 MB di memoria e raggiungendo un massimo di 3.000 Mbps a 10.240 MB. |
| Dimensioni del <a href="#">pacchetto di implementazione (archivio di file .zip)</a> | <p>50 MB (compressi, se caricato tramite l'API Lambda o gli SDK). Carica file di dimensioni maggiori con Amazon S3.</p> <p>50 MB (se caricati tramite la console Lambda)</p> <p>250 MB La dimensione massima del contenuto di un pacchetto di implementazione, inclusi livelli e runtime personalizzati (decompresso).</p> |
| Impostazioni dell'immagine di container                                             | 16 KB                                                                                                                                                                                                                                                                                                                      |
| Dimensione del pacchetto del codice dell' <a href="#">immagine di container</a>     | 10 GB (dimensione massima dell'immagine non compressa, inclusi tutti i livelli)                                                                                                                                                                                                                                            |
| Eventi di test (editor della console)                                               | 10                                                                                                                                                                                                                                                                                                                         |
| Storage della directory /tmp                                                        | Compreso tra 512 MB e 10.240 MB in incrementi di 1 MB                                                                                                                                                                                                                                                                      |

| Risorsa                                                                                                                                                                                                                                                                                                                                                                                                                                                | Quota |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| Descrittori di file                                                                                                                                                                                                                                                                                                                                                                                                                                    | 1,024 |
| <div>  <b>Note</b><br/>           Le istanze gestite Lambda utilizzano un limite di descrittori di file più elevato di 4.096. Per ulteriori informazioni, consulta <a href="#">Comprendere l'ambiente di esecuzione delle istanze gestite Lambda</a>.         </div>                                                                                                |       |
| Esecuzione processes/threads                                                                                                                                                                                                                                                                                                                                                                                                                           | 1,024 |
| <div>  <b>Note</b><br/>           Le istanze gestite Lambda utilizzano il processo e i limiti di thread predefiniti di Bottlerocket. <a href="https://aws.amazon.com/bottlerocket/">https://aws.amazon.com/bottlerocket/</a> Per ulteriori informazioni, consulta <a href="#">Comprendere l'ambiente di esecuzione delle istanze gestite Lambda</a>.         </div> |       |

## Richieste API Lambda

Le seguenti quote sono associate alle richieste API Lambda.

| Risorsa                                                   | Quota                                                             |
|-----------------------------------------------------------|-------------------------------------------------------------------|
| Richieste di chiamata per funzione per regione (sincrona) | Ogni istanza del tuo ambiente di esecuzione può gestire fino a 10 |

| Risorsa                                                                                                  | Quota                                                                                                                                                                                                                                                                                    |
|----------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                          | richieste al secondo. In altre parole, il limite totale di chiamate è 10 volte il limite di simultaneità. Consulta <a href="#">Informazioni sulla scalabilità della funzione Lambda</a> .                                                                                                |
| Richieste di chiamata per funzione per regione (asincrona)                                               | Ogni istanza del tuo ambiente di esecuzione può soddisfare un numero illimitato di richieste. In altre parole, il limite totale di chiamate si basa solo sulla simultaneità disponibile per la funzione. Consulta <a href="#">Informazioni sulla scalabilità della funzione Lambda</a> . |
| Richieste di invocazione per versione di funzione o alias (richieste al secondo)                         | 10 x <a href="#">simultaneità fornita</a> allocata <div> <p> <b>Note</b></p> <p>Questa quota si applica solo alle funzioni che utilizzano la simultaneità fornita.</p> </div>                       |
| Richieste API <a href="#">GetFunction</a>                                                                | 100 richieste al secondo. Non può essere aumentato.                                                                                                                                                                                                                                      |
| Richieste API <a href="#">GetPolicy</a>                                                                  | 15 richieste al secondo. Non può essere aumentato.                                                                                                                                                                                                                                       |
| Resto delle richieste API del piano di controllo (escluse invocazioni e richieste) GetFunction GetPolicy | 15 richieste al secondo per tutte le API (non 15 richieste al secondo per API). Non può essere aumentato.                                                                                                                                                                                |

## Funzioni durevoli

Le seguenti quote si applicano alle funzioni [durevoli](#). Le quote si applicano per. Regione AWS Valori predefiniti più elevati si applicano negli Stati Uniti orientali (Virginia settentrionale), negli Stati Uniti occidentali (Oregon) e in Europa (Irlanda), come indicato nella tabella. Salvo quanto indicato, queste quote possono essere aumentate. Per ulteriori informazioni, consulta [Richiesta di un aumento di quota](#) nella Guida per l'utente delle Service Quotas.

| Risorsa                                                              | Quota predefinita                                                                                                                                                                                                                          |
|----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Numero massimo di esecuzioni durevoli                                | <p>5.000.000 per regione.</p> <p>10.000.000 negli Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon) ed Europa (Irlanda).</p> <p>Si tratta di un limite flessibile che può essere aumentato.</p>            |
| Numero massimo di operazioni durevoli per esecuzione duratura        | <p>3.000</p> <p>Non può essere aumentato. Per ulteriori informazioni, vedere Operazioni durevoli <a href="#">disponibili</a>.</p>                                                                                                          |
| Storage di esecuzione durevole scritto in megabyte                   | <p>100 MB</p> <p>Non può essere aumentato. La dimensione cumulativa del payload è stata mantenuta dalle funzioni durevoli per esecuzione. Per ulteriori informazioni, vedere Dati <a href="#">persistenti per operazione duratura</a>.</p> |
| Frequenza di richiamo delle funzioni per avviare esecuzioni durature | 300 richieste al secondo.                                                                                                                                                                                                                  |
| Richieste API <a href="#">CheckpointDurableExecution</a>             | 10.000 richieste al secondo.                                                                                                                                                                                                               |

| Risorsa                                                             | Quota predefinita                                                                                                                                                            |
|---------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                     | 20.000 richieste al secondo negli Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon) ed Europa (Irlanda).                                     |
| Richieste API <a href="#">GetDurableExecution</a>                   | 30 richieste al secondo.                                                                                                                                                     |
| Richieste API <a href="#">GetDurableExecutionHistory</a>            | 15 richieste al secondo.                                                                                                                                                     |
| Richieste API <a href="#">GetDurableExecutionState</a>              | 10.000 richieste al secondo.<br><br>20.000 richieste al secondo negli Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon) ed Europa (Irlanda). |
| Richieste API <a href="#">ListDurableExecutionsByFunction</a>       | 15 richieste al secondo.                                                                                                                                                     |
| Richieste API <a href="#">SendDurableExecutionCallbackFailure</a>   | 3.000 richieste al secondo.<br><br>6.000 richieste al secondo negli Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon) ed Europa (Irlanda).   |
| Richieste API <a href="#">SendDurableExecutionCallbackHeartbeat</a> | 3.000 richieste al secondo.<br><br>6.000 richieste al secondo negli Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon) ed Europa (Irlanda).   |
| Richieste API <a href="#">SendDurableExecutionCallbackSuccess</a>   | 3.000 richieste al secondo.<br><br>6.000 richieste al secondo negli Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon) ed Europa (Irlanda).   |

| Risorsa                                            | Quota predefinita                                                                                                                                                      |
|----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Richieste API <a href="#">StopDurableExecution</a> | 300 richieste al secondo.<br><br>600 richieste al secondo negli Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon) ed Europa (Irlanda). |

## MicroVM Lambda

Lambda MicroVMS imposta le quote per le richieste di elaborazione, archiviazione e API. Le quote contrassegnate come regolabili possono essere aumentate tramite la console Service Quotas.

### Note

Le microVMS Lambda supportano l'architettura ARM64 (Graviton).AWS

## Elaborazione e archiviazione

| Risorsa                                                         | Quota predefinita                                                                                                                                                                                                                                                  | Regolabile |
|-----------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| Memoria allocata su tutte le MicroVM (per account, per regione) | 400 GB (ovvero 200 microVM da 2 GB di memoria configurata ciascuna o 400 microVM da 1 GB di memoria configurata ciascuna).<br><br>1.024 GB negli Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon), Stati Uniti orientali (Ohio) e | Sì         |

| Risorsa                                  | Quota predefinita                                                                                                                                                                                    | Regolabile |
|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
|                                          | Asia Pacifico (Tokyo)<br>(ovvero, 512 microVM da 2 GB di memoria configurata ciascuna o 1.024 microVM da 1 GB di memoria configurata ciascuna).<br><br>Riducibile fino a quattro volte questa quota. |            |
| Durata massima di esecuzione per microVM | 8 ore (28.800 secondi)                                                                                                                                                                               | No         |

## Immagini e versioni

| Risorsa                                                     | Quota predefinita                                                                                                                                         | Regolabile |
|-------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| Immagini MicroVM per account per regione                    | 100                                                                                                                                                       | Sì         |
| Versioni per immagine MicroVM                               | 50                                                                                                                                                        | Sì         |
| Creazioni simultanee di immagini (per account, per regione) | 5<br><br>10 negli Stati Uniti orientali (Virginia settentrionale), Stati Uniti occidentali (Oregon), Stati Uniti orientali (Ohio) e Asia Pacifico (Tokyo) | Sì         |

## Per-MicroVM produttività

| Risorsa                            | Limite                                                           | Regolabile |
|------------------------------------|------------------------------------------------------------------|------------|
| Connessioni simultanee per microVM | 8 (1 vCPU), 16 (2 vCPU), 32 (4 vCPU), 64 (8 vCPU), 128 (16 vCPU) | No         |
| Richieste al secondo per microVM   | 40 (4 vCPU/8 GB), 160 (16 vCPU/32 GB)                            | No         |

## Limiti frequenza API

| Operazione API              | Velocità (TPS) | Burst | Regolabile |
|-----------------------------|----------------|-------|------------|
| RunMicrovm                  | 5              | 5     | Sì         |
| ResumeMicrovm               | 5              | 5     | Sì         |
| SuspendMicrovm              | 2              | 2     | Sì         |
| TerminateMicrovm            | 10             | 10    | Sì         |
| GetMicrovm                  | 100            | 100   | Sì         |
| CreateMicrovmAuthToken      | 50             | 50    | Sì         |
| CreateMicrovmShellAuthToken | 5              | 5     | Sì         |

### Note

TPS = transazioni al secondo. Questi limiti tariffari sono per account, per regione. Riprova le richieste limitate utilizzando il backoff esponenziale con jitter.



## Altri servizi

Le quote per altri servizi, come AWS Identity and Access Management (IAM), Amazon (Lambda @Edge) e Amazon Virtual Private Cloud CloudFront (Amazon VPC), possono influire sulle tue funzioni Lambda. Per ulteriori informazioni, consulta la pagina [Servizio AWS quotas](#) nella Riferimenti generali di Amazon Web Services e la pagina [Richiamare Lambda con eventi provenienti da altri AWS services](#).

Molte applicazioni che coinvolgono Lambda utilizzano più servizi. AWS Poiché servizi diversi hanno quote diverse per varie funzionalità, può essere difficile gestire queste quote nell'intera applicazione. Ad esempio, API Gateway ha un limite di accelerazione predefinito di 10.000 richieste al secondo, mentre Lambda ha un limite di concorrenza predefinito di 1.000. A causa di questa discrepanza, è possibile avere più richieste in arrivo da API Gateway che Lambda è in grado di gestire. Puoi risolvere questo problema richiedendo un aumento del limite di concorrenza di Lambda in modo che corrisponda al livello di traffico previsto.

Il test di carico dell'applicazione consente di monitorarne le prestazioni end-to-end prima della distribuzione in produzione. Durante un test di carico, è possibile identificare eventuali quote che possono fungere da fattore limitante per i livelli di traffico previsti e agire di conseguenza.

## Cronologia dei documenti

Nella tabella seguente sono descritte le modifiche importanti apportate alla Guida per sviluppatori di AWS Lambda a partire da maggio 2018. Per ricevere notifiche sugli aggiornamenti della documentazione, è possibile iscriversi al [feed RSS](#).

| Modifica                                                  | Descrizione                                                                                                                                                                                                                                                                                                                               | Data           |
|-----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| <a href="#">AWS aggiornamento gestito delle politiche</a> | Lambda ha aggiunto una nuova policy AWS gestita (AWSLambdaNetworkConnectorOperatorPolicy) per concedere le autorizzazioni per creare e amministrare risorse di interfaccia di rete elastica (ENI) gestite da Lambda Network Connector. Per ulteriori informazioni, consulta Aggiornamenti <a href="#">Lambda alle policy gestite. AWS</a> | 19 giugno 2026 |
| <a href="#">AWS aggiornamento della politica gestita</a>  | Lambda ha aggiornato la policy AWSLambdaManagedEC2ResourceOperator gestita per aggiungere CloudWatch le autorizzazioni Amazon (logs:CreateLogGroup, logs:CreateLogStream, logs:PutLogEvents). Per i dettagli, consulta Aggiornamenti <a href="#">Lambda alle AWS policy gestite.</a>                                                      | 10 giugno 2026 |
| <a href="#">Self-managed Archiviazione del codice S3</a>  | Lambda ora supporta bucket S3 autogestiti per l'archivi                                                                                                                                                                                                                                                                                   | 30 aprile 2026 |

azione di codici di funzioni e livelli. Puoi configurare Lambda in modo che faccia riferimento al codice direttamente dal bucket S3, eliminando i limiti di archiviazione. Lambda-managed Per i dettagli, consulta Archiviazione del codice [Self-managed S3](#).

### [AWS aggiornamento della politica gestita](#)

Lambda ha aggiornato la policy AWSLambdaManagedEC2ResourceOperator gestita per aggiungere l'ec2:DescribeVpcEncryptionControls autorizzazione. Per i dettagli, consulta Aggiornamenti [Lambda alle politiche AWS gestite](#).

23 aprile 2026

### [AWS aggiornamenti gestiti delle politiche](#)

Lambda ha rilasciato una nuova policy gestita AWSLambdaBasicDurableExecutionRolePolicy che fornisce autorizzazioni di scrittura ai CloudWatch log e read/write autorizzazioni alle API di esecuzione durevole utilizzate da Lambda durable functions.

1 dicembre 2025

|                                                           |                                                                                                                                                                                                                                |                  |
|-----------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">Service-linked ruolo per Lambda</a>           | Lambda ora supporta un ruolo collegato ai servizi (AWSServiceRoleForLambda ) che consente a Lambda di terminare le istanze gestite come parte dei fornitori di capacità Lambda.                                                | 30 novembre 2025 |
| <a href="#">Istanze gestite Lambda</a>                    | Lambda ora supporta le istanze gestite, che forniscono capacità di calcolo dedicata per le tue funzioni con prestazioni, sicurezza e controllo migliorati. Per i dettagli, consulta <a href="#">Lambda Managed Instances</a> . | 30 novembre 2025 |
| <a href="#">AWS aggiornamenti gestiti delle politiche</a> | Lambda ha aggiunto una nuova policy AWS gestita (AWSLambdaBasicDurableExecutionRolePolicy ).                                                                                                                                   | 30 novembre 2025 |
| <a href="#">AWS aggiornamenti gestiti delle politiche</a> | Lambda ha aggiornato la policy AWS gestita esistente AWSLambda_FullAccess ) consente kms:DescribeKey le iam:CreateServiceLinkedRole azioni.                                                                                    | 30 novembre 2025 |

[AWS aggiornamento della politica gestita](#)

Lambda ha aggiunto una nuova policy AWS gestita (AWSLambdaManagedEC2ResourceOperator ) per abilitare la gestione automatizzata delle istanze Amazon EC2 per i fornitori di capacità Lambda. Per i dettagli, consulta gli aggiornamenti [Lambda alle policy gestite. AWS](#)

30 novembre 2025

[AWS aggiornamento della politica gestita](#)

Lambda ha aggiunto una nuova policy AWS gestita (AWSLambdaServiceRolePolicy ). Per i dettagli, consulta Aggiornamenti [Lambda alle policy AWS gestite.](#)

30 novembre 2025

[AWS aggiornamenti gestiti delle politiche](#)

Lambda ha aggiornato due policy AWS gestite esistenti (AWSLambda\_ReadOnlyAccess eAWSLambda\_FullAccess ).

20 febbraio 2025

[Node.js Runtime 22.x](#)

Lambda ora supporta Node.js 22 come runtime gestito e immagine di base del contenitore (nodejs22.x ).

22 novembre 2024

[SnapStart supporto per Python e .NET](#)

Lambda SnapStart è ora disponibile per i runtime gestiti da Python e .NET, a partire da e. python3.12 dotnet8

18 novembre 2024

|                                                                                        |                                                                                                                                                                                                                        |                  |
|----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">Runtime di Python 3.13</a>                                                 | Ora Lambda supporta Python 3.13 come runtime gestito e immagine di base del container.                                                                                                                                 | 13 novembre 2024 |
| <a href="#">Crittografia gestita dal cliente per pacchetti di implementazione .zip</a> | Lambda ora supporta la crittografia delle chiavi gestita AWS KMS dal cliente per i pacchetti di distribuzione .zip.                                                                                                    | 8 novembre 2024  |
| <a href="#">Supporto per nuove regioni SnapStart</a>                                   | Lambda <a href="#">SnapStart</a> è ora disponibile nelle seguenti regioni: Europa (Spagna), Europa (Zurigo), Asia Pacifico (Melbourne), Asia Pacifico (Hyderabad) e Medio Oriente (Emirati Arabi Uniti).               | 12 gennaio 2024  |
| <a href="#">AWS aggiornamenti gestiti delle politiche</a>                              | Lambda ha aggiornato una policy AWS gestita esistente (AWSLambdaVPCAccessExecutionRole ).                                                                                                                              | 5 gennaio 2024   |
| <a href="#">Runtime di Python 3.12</a>                                                 | Ora Lambda supporta Python 3.12 come runtime gestito e immagine di base del container. Per ulteriori informazioni, consulta il runtime di <a href="#">Python 3.12 ora disponibile nel Compute AWS Lambda</a> Blog. AWS | 14 dicembre 2023 |
| <a href="#">Runtime Java 21</a>                                                        | Lambda ora supporta Java 21 come runtime gestito e immagine di base del container (java21).                                                                                                                            | 16 novembre 2023 |

|                                                            |                                                                                                                                                                                                                                                                                                                                                     |                  |
|------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">Node.js Runtime 20.x</a>                       | Lambda ora supporta Node.js 20 come runtime gestito e immagine di base del contenitore (nodejs20.x). Per ulteriori informazioni, consulta il runtime <a href="#">Node.js 20.x ora disponibile nel AWS Lambda</a> AWS Compute Blog.                                                                                                                  | 14 novembre 2023 |
| <a href="#">Runtime provided.al2023</a>                    | Lambda ora supporta Amazon Linux 2023 come runtime gestito e immagine di base del container. Per ulteriori informazioni, consulta <a href="#">Introducing the Amazon Linux 2023 runtime for nel AWS Lambda</a> AWS Compute Blog.                                                                                                                    | 9 novembre 2023  |
| <a href="#">Supporto IPv6 per sottoreti dual-stack</a>     | Lambda ora supporta il traffico IPv6 in uscita verso le sottoreti dual-stack. Per ulteriori informazioni, consultare <a href="#">Supporto IPv6</a> .                                                                                                                                                                                                | 12 ottobre 2023  |
| <a href="#">Test di funzioni e applicazioni serverless</a> | Scopri le tecniche per eseguire il debug e automatizzare i test delle funzioni serverless nel cloud. Ora è presente un capitolo sui test e nelle sezioni dei linguaggi Python e Typescript sono state incluse delle risorse in merito. Per ulteriori informazioni, consulta la sezione <a href="#">Test di funzioni e applicazioni serverless</a> . | 16 giugno 2023   |

|                                                     |                                                                                                                                                                                                                                                                                                                    |                 |
|-----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">Runtime Ruby 3.2</a>                    | Lambda ora supporta un nuovo runtime per Ruby 3.2. Per ulteriori informazioni, consulta la sezione <a href="#">Compilazione di funzioni Lambda con Ruby</a> .                                                                                                                                                      | 7 giugno 2023   |
| <a href="#">Streaming delle risposte</a>            | Lambda ora supporta lo streaming delle risposte dalle funzioni. Per ulteriori informazioni, consulta la pagina <a href="#">Configurazione di una funzione Lambda per lo streaming delle risposte</a> .                                                                                                             | 6 aprile 2023   |
| <a href="#">Parametri di chiamata asincrona</a>     | Lambda rilascia parametri di chiamata asincrona. Per ulteriori informazioni, consulta <a href="#">Parametri di chiamata asincrona</a> .                                                                                                                                                                            | 9 febbraio 2023 |
| <a href="#">Controlli della versione di runtime</a> | Lambda rilascia nuove versioni di runtime che includono aggiornamenti di sicurezza , correzioni di bug e nuove funzionalità. Ora puoi controllare quando le tue funzioni vengono aggiornate alle nuove versioni di runtime. Per ulteriori informazioni, consulta <a href="#">Aggiornamenti di runtime Lambda</a> . | 23 gennaio 2023 |



|                                                               |                                                                                                                                                                                                                                                                                 |                  |
|---------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">Lambda SnapStart</a>                              | Usa Lambda SnapStart per ridurre i tempi di avvio delle funzioni Java senza fornire risorse aggiuntive o implementare ottimizzazioni complesse delle prestazioni. Per ulteriori informazioni, consulta <a href="#">Migliorare le prestazioni di avvio con Lambda. SnapStart</a> | 28 novembre 2022 |
| <a href="#">Node.js 18 runtime</a>                            | Lambda ora supporta un nuovo runtime per Node.js 18. Node.js 18 utilizza Amazon Linux 2. Per i dettagli, consulta <a href="#">Creazione di funzioni Lambda con Node.js</a> .                                                                                                    | 18 novembre 2022 |
| <a href="#">lambda: chiave condizionale SourceFunctionArn</a> | Per una AWS risorsa, la chiave <code>lambda:SourceFunctionArn</code> condizionale filtra l'accesso alla risorsa tramite l'ARN di una funzione Lambda. Per maggiori dettagli, consulta <a href="#">Utilizzo delle credenziali dell'ambiente di esecuzione Lambda</a> .           | 1 luglio 2022    |
| <a href="#">Node.js 16 runtime</a>                            | Lambda ora supporta un nuovo runtime per Node.js 16. Node.js 16 utilizza Amazon Linux 2. Per i dettagli, consulta <a href="#">Creazione di funzioni Lambda con Node.js</a> .                                                                                                    | 11 maggio 2022   |

|                                                                                              |                                                                                                                                                                                                                   |                  |
|----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">URL della funzione Lambda</a>                                                    | Lambda ora supporta gli URL della funzione, che sono endpoint HTTP(S) dedicati per le funzioni Lambda. Per informazioni dettagliate, consulta <a href="#">URL della funzione Lambda</a> .                         | 6 aprile 2022    |
| <a href="#">Eventi di test condivisi nella console AWS Lambda</a>                            | Lambda ora supporta la condivisione di eventi di test con altri utenti nello stesso Account AWS. Per informazioni dettagliate, consulta <a href="#">Test delle funzioni Lambda nella console</a> .                | 16 marzo 2022    |
| <a href="#">PrincipalOrgId nelle politiche basate sulle risorse</a>                          | Lambda ora supporta la concessione delle autorizzazioni a un'organizzazione in AWS Organizations. Per informazioni dettagliate, consulta <a href="#">Utilizzo delle policy basate su risorse per AWS Lambda</a> . | 11 marzo 2022    |
| <a href="#">Runtime .NET 6</a>                                                               | Lambda ora supporta un nuovo runtime per .NET 6. Per informazioni dettagliate, consultare <a href="#">Runtime di Lambda</a> .                                                                                     | 23 febbraio 2022 |
| <a href="#">Filtraggio di eventi per le origini di eventi Kinesis, DynamoDB e Amazon SQS</a> | Lambda ora supporta il filtraggio di eventi per le origini di eventi Kinesis, DynamoDB e Amazon SQS. Per informazioni dettagliate, consulta <a href="#">Filtro eventi Lambda</a> .                                | 24 novembre 2021 |

|                                                                                              |                                                                                                                                                                                                |                   |
|----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| <a href="#">Autenticazione mTLS per Amazon MSK e origine eventi Apache Kafka autogestito</a> | Lambda ora supporta l'autenticazione mTLS per Amazon MSK e le origini eventi Apache Kafka autogestito. Per informazioni dettagliate, consultare <a href="#">Uso di Lambda con Amazon MSK</a> . | 19 novembre 2021  |
| <a href="#">Lambda su Graviton2</a>                                                          | Lambda ora supporta Graviton2 per le funzioni che utilizzano l'architettura arm64. Per informazioni dettagliate, consulta <a href="#">Architetture del set di istruzioni Lambda</a> .          | 29 settembre 2021 |
| <a href="#">Runtime di Python 3.9</a>                                                        | Lambda ora supporta un nuovo runtime per Python 3.9. Per informazioni dettagliate, consultare <a href="#">Runtime di Lambda</a> .                                                              | 16 agosto 2021    |
| <a href="#">Nuove versioni di runtime per Python Node.js e Java</a>                          | Sono disponibili nuove versioni di runtime per Node.js Python e Java. Per informazioni dettagliate, consultare <a href="#">Runtime di Lambda</a> .                                             | 21 luglio 2021    |

### [Supporto per RabbitMQ come origine evento su Lambda](#)

Lambda ora supporta Amazon MQ per RabbitMQ come origine evento. Amazon MQ è un servizio gestito di broker di messaggistica per Apache ActiveMQ e RabbitMQ che facilita la configurazione e la gestione di broker di messaggistica nel cloud. Per informazioni dettagliate, consultare [Uso di Lambda con Amazon MQ](#).

7 luglio 2021

### [SASL/PLAIN autenticazione per Kafka autogestito su Lambda](#)

SASL/PLAIN è ora un meccanismo di autenticazione supportato per le sorgenti di eventi Kafka autogestite su Lambda. I clienti che già utilizzano SASL/PLAIN il proprio cluster Kafka autogestito possono ora utilizzare facilmente Lambda per creare applicazioni consumer senza dover modificare il modo in cui si autenticano. Per informazioni dettagliate, consultare [Uso di Lambda con Apache Kafka autogestito](#).

29 giugno 2021

|                                                       |                                                                                                                                                                                                                                                                                                                     |                 |
|-------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">API estensioni Lambda</a>                 | Disponibilità generale per le estensioni Lambda. Utilizza le estensioni per potenziare le funzioni Lambda. È possibile utilizzare le estensioni fornite dai Lambda Partners oppure creare le tue estensioni Lambda. Per informazioni dettagliate, consultare <a href="#">API Extensions di Lambda</a> .             | 24 maggio 2021  |
| <a href="#">Nuova esperienza della console Lambda</a> | La console Lambda è stata riprogettata per migliorare le prestazioni e la coerenza.                                                                                                                                                                                                                                 | 2 marzo 2021    |
| <a href="#">Node.js 14 runtime</a>                    | Lambda ora supporta un nuovo runtime per Node.js 14. Node.js 14 utilizza Amazon Linux 2. Per i dettagli, consulta <a href="#">Creazione di funzioni Lambda con Node.js</a> .                                                                                                                                        | 27 gennaio 2021 |
| <a href="#">Immagini di container Lambda</a>          | Lambda ora supporta le funzioni definite come immagini di container. È possibile combinare la flessibilità degli strumenti per container con l'agilità e la semplicità operativa di Lambda per costruire applicazioni. Per ulteriori dettagli, consultare <a href="#">Uso di immagini di container con Lambda</a> . | 1 dicembre 2020 |

### [Firma del codice per le funzioni Lambda](#)

Lambda ora supporta la firma del codice. Gli amministratori possono configurare le funzioni Lambda in modo da accettare al momento della distribuzione solo codice firmato. Lambda controlla le firme per assicurarsi che il codice non venga alterato o manomesso. Inoltre, Lambda assicura che il codice sia firmato da sviluppatori affidabili prima di accettarne la distribuzione. Per informazioni dettagliate, consultare l'argomento [Configurazione della firma del codice per Lambda](#).

23 novembre 2020

### [Anteprima: API Runtime Logs di Lambda](#)

Lambda ora supporta l'API Runtime Logs. Le estensioni di Lambda possono utilizzare l'API Logs per eseguire la sottoscrizione ai flussi di log nell'ambiente di esecuzione. Per informazioni dettagliate, consultare [API Runtime Logs di Lambda](#).

12 novembre 2020

### [Nuova origine evento per Amazon MQ](#)

Lambda ora supporta Amazon MQ come origine evento. Utilizza una funzione Lambda per elaborare i record dal broker di messaggi Amazon MQ. Per informazioni dettagliate, consultare [Uso di Lambda con Amazon MQ](#).

5 novembre 2020

### [Anteprima: API estensioni Lambda](#)

Utilizza le estensioni Lambda per potenziare le funzioni Lambda. È possibile utilizzare le estensioni fornite dai Lambda Partners oppure creare le tue estensioni Lambda. Per informazioni dettagliate, consultare [API Extensions di Lambda](#).

8 ottobre 2020

### [Supporto per Java 8 e runtime personalizzati su AL2](#)

Lambda supporta ora Java 8 e runtime personalizzati su Amazon Linux 2. Per informazioni dettagliate, consultare [Runtime di Lambda](#).

12 agosto 2020

### [Nuova origine evento per Amazon Managed Streaming for Apache Kafka](#)

Lambda ora supporta Amazon MSK come origine evento. Utilizza una funzione Lambda con Amazon MSK per elaborare i record in un argomento Kafka. Per informazioni dettagliate, consultare [Uso di Lambda con Amazon MSK](#).

11 agosto 2020

### [Chiavi di condizione IAM per le impostazioni di Amazon VPC](#)

Ora puoi utilizzare i tasti di Lambda-specific condition e per le impostazioni del VPC. Ad esempio, è possibile richiedere che tutte le funzioni dell'organizzazione siano connesse a un VPC. È inoltre possibile specificare le sottoreti e i gruppi di sicurezza che gli utenti della funzione possono e non possono utilizzare. Per informazioni dettagliate, consultare [Configurazione di VPC per le funzioni IAM](#).

10 agosto 2020

### [Impostazioni di concorrenza per i consumatori di streaming Kinesis HTTP/2](#)

È ora possibile utilizzare e le seguenti impostazioni di concorrenza per i consumatori Kinesis con fan-out (HTTP/2 stream) migliorato: ParallelizationFactor MaximumRetryAttempts MaximumRecordAgeInSeconds DestinationConfig BisectBatchOnFunctionError Per informazioni dettagliate, consulta [Utilizzo con Amazon Kinesis. AWS Lambda](#)

7 luglio 2020



### [Finestra batch per gli utenti di streaming Kinesis HTTP/2](#)

È ora possibile configurare una finestra batch (MaximumBatchingWindowInSeconds) per gli HTTP/2 stream. Lambda legge i record dal flusso fino a quando non ha raccolto un batch completo o fino alla scadenza del periodo di batch. Per informazioni dettagliate, consulta [Utilizzo AWS Lambda con Amazon Kinesis](#).

18 giugno 2020

### [Supporto per i file system Amazon EFS](#)

È ora possibile connettere un file system Amazon EFS alle proprie funzioni Lambda per l'accesso condiviso ai file di rete. Per informazioni dettagliate, consultare [Configurazione dell'accesso al file system per le funzioni Lambda](#).

16 giugno 2020

### [AWS CDK applicazioni di esempio nella console Lambda](#)

La console Lambda ora include applicazioni di esempio che utilizzano il Kit di sviluppo per il cloud AWS (AWS CDK) form. TypeScript AWS CDK È un framework che consente di definire le risorse dell'applicazione in Python TypeScript, Java o .NET.

1 giugno 2020

### [Supporto per il runtime .NET Core 3.1.0 in AWS Lambda](#)

AWS Lambda ora supporta il runtime .NET Core 3.1.0. Per i dettagli, consultare [Interfaccia a riga di comando \(CLI\) di .NET Core](#).

31 marzo 2020

## [Supporto per le API HTTP di API Gateway](#)

Documentazione aggiornata e ampliata per l'utilizzo di Lambda con API Gateway, incluso il supporto per le API HTTP. È stata aggiunta un'applicazione di esempio che crea un'API e funziona con CloudFormation. Per informazioni dettagliate, consultare [Uso di Lambda con Gateway Amazon API](#).

23 marzo 2020

## [Ruby 2.7](#)

Un nuovo runtime è disponibile per Ruby 2.7, ruby2.7, che è il primo runtime Ruby a utilizzare Amazon Linux 2. Per informazioni dettagliate, consultare [Compilazione di funzioni Lambda con Ruby](#).

19 febbraio 2020

## [Parametri di concorrenza](#)

Lambda ora riporta i parametri `ConcurrentExecutions` per tutte le funzioni, gli alias e le versioni. È possibile visualizzare un grafico per questo parametro nella pagina di monitoraggio della funzione. In precedenza, `ConcurrentExecutions` è stato segnalato solo a livello di account e per le funzioni che utilizzano la concorrenza riservata. Per ulteriori dettagli, consultare [Parametri delle funzioni di AWS Lambda](#).

18 febbraio 2020

## [Aggiornamento agli stati delle funzioni](#)

Gli stati delle funzioni vengono ora applicati per impostazione predefinita a tutte le funzioni. Quando si connette una funzione a un VPC, Lambda crea interfacce di rete elastiche condivise. Ciò consente alla funzione di dimensionarsi senza creare interfacce di rete aggiuntive. Durante questo periodo, non è possibile eseguire ulteriori operazioni sulla funzione, incluso l'aggiornamento della configurazione e la pubblicazione delle versioni. In alcuni casi, viene influenzata anche la chiamata. I dettagli sullo stato corrente di una funzione sono disponibili dall'API Lambda.

Questo aggiornamento viene rilasciato in fasi. Per i dettagli, consulta [Ciclo di vita aggiornato degli stati Lambda per le reti VPC nel blog di Compute. AWS](#) Per ulteriori informazioni sugli stati delle funzioni, consultare [Stati delle funzioni AWS Lambda](#).

24 gennaio 2020

### [Aggiornamenti all'output dell'API di configurazione delle funzioni](#)

Sono stati aggiunti i codici di motivo a [StateReasonCode](#) (InvalidSubnet, InvalidSecurityGroup) e LastUpdateReasonCode (, SubnetOutOfIPAddresses InvalidSubnet, InvalidSecurityGroup) per le funzioni che si connettono a un VPC. Per ulteriori informazioni sugli stati delle funzioni, consultare [Stati delle funzioni AWS Lambda](#).

20 gennaio 2020

### [Concorrenza fornita](#)

È ora possibile allocare la concorrenza fornita per una versione o un alias di funzione. La concorrenza fornita consente a una funzione di dimensionarsi senza fluttuazioni nella latenza. Per ulteriori dettagli, consultare l'argomento relativo alla [gestione della concorrenza di una funzione Lambda](#).

3 dicembre 2019

### [Creare un proxy di database](#)

È ora possibile utilizzare la console Lambda per creare un proxy di database per una funzione Lambda. Un proxy di database consente a una funzione di raggiungere livelli di concorrenza elevati senza esaurire le connessioni al database. Per informazioni dettagliate, consultare [Configurazione dell'accesso al database per una funzione Lambda](#).

3 dicembre 2019

### [Supporto dei percentili per il parametro della durata](#)

È ora possibile filtrare il parametro della durata in base ai percentili. Per ulteriori dettagli, consultare [Parametri di AWS Lambda](#).

26 novembre 2019

### [Maggiore concorrenza per le origini eventi di flusso](#)

Una nuova opzione per le mappature dell'origine evento del [flusso DynamoDB](#) e del [flusso Kinesis](#) consente di elaborare più batch alla volta da ogni shard. Quando si aumenta il numero di batch simultanei per shard, la concorrenza della funzione può essere fino a 10 volte il numero degli shard nel flusso. Per informazioni dettagliate, consultare [Mappatura dell'origine evento di Lambda](#).

25 novembre 2019

## [Stati delle funzioni](#)

Quando crei o aggiorni una funzione, questa entra in uno stato di attesa mentre Lambda fornisce le risorse per supportarla. Se connetti la funzione a un VPC, Lambda può creare immediatamente un'interfaccia di rete elastica condivisa, invece di creare interfacce di rete quando viene richiamata la funzione. Ciò si traduce in prestazioni migliori per VPC-connected le funzioni, ma potrebbe richiedere un aggiornamento dell'automazione. Per ulteriori dettagli, consultare [Stati delle funzioni di AWS Lambda](#).

25 novembre 2019

## [Opzioni di gestione degli errori per la chiamata asincrona](#)

Sono disponibili nuove opzioni di configurazione per la chiamata asincrona. È possibile configurare Lambda in modo da limitare i tentativi e impostare un'età massima dell'evento. Per ulteriori dettagli, consultare l'argomento relativo alla [configurazione della gestione degli errori per l'invocazione asincrona](#).

25 novembre 2019

## [Gestione degli errori per le origini eventi di flusso](#)

25 novembre 2019

Sono disponibili nuove opzioni di configurazione per le mappature delle origini eventi che leggono dai flussi. È possibile configurare le mappature dell'origine evento del [flusso DynamoDB](#) e del [flusso Kinesis](#) per limitare i tentativi e impostare un'età massima dei record. Quando si verificano errori, è possibile configurare la mappatura dell'origine evento per dividere i batch prima di ripetere il tentativo e inviare record di invocazione per i batch in errore a una coda o un argomento. Per informazioni dettagliate, consultare [Mappatura dell'origine evento di Lambda](#).

### [Destinazioni per la chiamata asincrona](#)

È ora possibile configurare Lambda in modo da inviare record di chiamate asincrone a un altro servizio. I record di invocazione contengono i dettagli sull'evento, il contesto e la risposta della funzione. È possibile inviare i record di chiamata a una coda SQS, a un argomento SNS, a una funzione Lambda o a un bus di eventi. EventBridge Per ulteriori dettagli, consultare [l'argomento relativo alla configurazione delle destinazioni per l'invocazione asincrona](#).

25 novembre 2019

### [Nuovi runtime per Node.js Python e Java](#)

Sono disponibili nuovi runtime per Node.js 12, Python 3.8 e Java 11. Per informazioni dettagliate, consultare [Runtime di Lambda](#).

18 novembre 2019

### [Supporto per la coda FIFO per le origini eventi Amazon SQS](#)

È ora possibile creare una mappatura di origine eventi che legge da una coda first-in, first-out (FIFO). In precedenza, erano supportate solo le code standard. Per informazioni dettagliate, consultare [Uso di Lambda con Amazon SQS](#).

18 novembre 2019



### [Creazione di applicazioni nella console Lambda](#)

La creazione di applicazioni nella console Lambda è ora disponibile a livello generale. Per le istruzioni, consulta [Gestione delle applicazioni nella console Lambda](#).

31 ottobre 2019

### [Creazione di applicazioni nella console Lambda \(beta\)](#)

Ora è possibile creare un'applicazione Lambda con una pipeline di distribuzione continua integrata nella console Lambda. La console fornisce applicazioni di esempio che è possibile utilizzare come punto di partenza per il proprio progetto. Scegli tra AWS CodeCommit e GitHub per il controllo del codice sorgente. Ogni volta che si inseriscono modifiche al repository, la pipeline inclusa le compila e le distribuisce automaticamente. Per le istruzioni, consulta [Gestione delle applicazioni nella console Lambda](#).

3 ottobre 2019

## [Miglioramenti delle prestazioni delle VPC-connected funzioni](#)

Lambda ora utilizza un nuovo tipo di interfaccia di rete elastica che è condivisa da tutte le funzioni in una sottorete VPC (Virtual Private Cloud). Quando si connette una funzione a un VPC, Lambda crea un'interfaccia di rete per ogni combinazione di gruppo di sicurezza e sottorete scelta. Quando le interfacce di rete condivise sono disponibili, la funzione non deve più creare interfacce di rete aggiuntive man mano che aumentano le dimensioni. Questo migliora notevolmente i tempi di avvio. Per ulteriori informazioni, consultare [Configurazione di una funzione Lambda per accedere alle risorse in un VPC](#).

3 settembre 2019

[Impostazioni dei batch di flussi](#)

Ora è possibile configurare un periodo di batch per le mappature dell'origine evento [Amazon DynamoDB](#) e [Amazon Kinesis](#). Configura un periodo di batch di un massimo di cinque minuti per eseguire il buffer dei record in entrata fino a quando non diventi disponibile il batch completo. In tal modo si riduce il numero di volte in cui la funzione viene invocata quando il flusso è meno attivo.

29 agosto 2019

[CloudWatch Integrazione con Logs Insights](#)

La pagina di monitoraggio nella console Lambda ora include i report di Amazon CloudWatch Logs Insights.

18 giugno 2019

[Amazon Linux 2018.03](#)

L'ambiente di esecuzione Lambda è in fase di aggiornamento per consentire l'uso di Amazon Linux 2018.03. Per ulteriori dettagli, consultare [Ambiente di esecuzione](#).

21 maggio 2019

## [Node.js 10](#)

È disponibile un nuovo runtime per Node.js 10, nodejs10. x. Questo runtime utilizza la versione Node.js 10.15 e verrà aggiornato periodicamente con l'ultima release di Point 10. Node.js Node.js 10 è anche il primo runtime a utilizzare Amazon Linux 2. Per i dettagli, consulta [Creazione di funzioni Lambda con Node.js](#). 13 maggio 2019

## [GetLayerVersionByArn API](#)

Usa l'[GetLayerVersionByArn](#) API per scaricare le informazioni sulla versione del livello con la versione ARN come input. Rispetto a GetLayerVersion, GetLayerVersionByArn consente di utilizzare direttamente l'ARN invece di analizzarlo per ottenere il nome del livello e il numero di versione. 25 aprile 2019

## [Ruby](#)

AWS Lambda ora supporta Ruby 2.5 con un nuovo runtime. Per informazioni dettagliate, consultare [e Compilazione di funzioni Lambda con Ruby](#). 29 novembre 2018

## [Livelli](#)

Con i livelli Lambda, è possibile impacchettare e distribuire librerie, runtime personalizzati e altre dipendenze separatamente dal codice della funzione. Condividere i propri livelli con gli altri account o in tutto il mondo. Per informazioni dettagliate, consultare [Livelli di Lambda](#).

29 novembre 2018

## [Runtime personalizzati](#)

Costruire un runtime personalizzato per eseguire le funzioni Lambda nel proprio linguaggio o di programmazione preferito. Per informazioni dettagliate, consultare [Runtime Lambda personalizzati](#).

29 novembre 2018

## [Trigger di Application Load Balancer](#)

Elastic Load Balancing ora supporta le funzioni Lambda come target per gli Application Load Balancer. Per informazioni dettagliate, consultare [Uso di Lambda con i sistemi di bilanciamento del carico delle applicazioni](#).

29 novembre 2018

### [Usa i consumatori di HTTP/2 streaming Kinesis come trigger](#)

È possibile utilizzare i consumatori di flussi di HTTP/2 dati Kinesis per inviare eventi a AWS Lambda. I consumatori di stream hanno un throughput di lettura dedicato da ogni frammento del flusso di dati e lo utilizzano HTTP/2 per ridurre al minimo la latenza. Per informazioni dettagliate, consultare [Uso di Lambda con Kinesis](#).

19 novembre 2018

### [Python 3.7](#)

AWS Lambda ora supporta Python 3.7 con un nuovo runtime. Per ulteriori informazioni, consultare [Compilazioni di funzioni Lambda con Python](#).

19 novembre 2018

### [Aumento del limite di payload per il richiamo della funzione asincrona](#)

La dimensione massima del payload per le invocazioni asincrone è aumentata da 128 KB a 256 KB, che corrisponde alla dimensione massima del messaggio da un trigger Amazon SNS. Per informazioni dettagliate, consultare [Quote di Lambda](#).

16 novembre 2018

### [AWS GovCloud \(US-East\) Regione](#)

AWS Lambda è ora disponibile nella regione AWS GovCloud (US-East).

12 novembre 2018

[AWS SAM Gli argomenti sono stati spostati in una Guida per gli sviluppatori separata](#)

Alcuni argomenti erano incentrati sulla creazione di applicazioni serverless utilizzando AWS Serverless Application Model (AWS SAM). Questi argomenti sono stati spostati nella [Guida per gli sviluppatori di AWS Serverless Application Model](#).

25 ottobre 2018

[Visualizzazione delle applicazioni Lambda nella console](#)

È possibile visualizzare lo stato delle applicazioni Lambda nella pagina [Applicazioni \(Applicazioni\)](#) della console Lambda. Questa pagina mostra lo stato dello CloudFormation stack. Include collegamenti a pagine con ulteriori informazioni sulle risorse dello stack. È possibile anche visualizzare i parametri di aggregazione per l'applicazione e creare pannelli di controllo di monitoraggio personalizzati.

11 ottobre 2018

[Limite di timeout dell'esecuzione della funzione](#)

Per consentire funzioni di lunga durata, il timeout di esecuzione massimo configurabile è aumentato da 5 minuti a 15 minuti. Per informazioni dettagliate, consultare [Limiti di Lambda](#).

10 ottobre 2018

|                                                                          |                                                                                                                                                                                                                        |                   |
|--------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| <a href="#">Supporto per il linguaggio PowerShell Core in AWS Lambda</a> | AWS Lambda ora supporta il linguaggio PowerShell Core. Per ulteriori informazioni, consulta <a href="#">Modello di programmazione per la creazione di funzioni Lambda in PowerShell</a>                                | 11 settembre 2018 |
| <a href="#">Supporto per il runtime .NET Core 2.1.0 in AWS Lambda</a>    | AWS Lambda ora supporta il runtime .NET Core 2.1.0. Per ulteriori informazioni, consultare <a href="#">Interfaccia a riga di comando (CLI) di .NET Core</a> .                                                          | 9 luglio 2018     |
| <a href="#">Aggiornamenti ora disponibili tramite RSS</a>                | Ora è possibile iscriversi a un feed RSS per seguire la pubblicazione di nuove versioni di questa guida.                                                                                                               | 5 luglio 2018     |
| <a href="#">Supporto per Amazon SQS come origine evento</a>              | AWS Lambda ora supporta Amazon Simple Queue Service (Amazon SQS) come fonte di eventi. Per ulteriori informazioni, consultare <a href="#">Richiamo delle funzioni Lambda</a> .                                         | 28 giugno 2018    |
| <a href="#">Regione Cina (Ningxia)</a>                                   | AWS Lambda è ora disponibile nella regione Cina (Ningxia). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina <a href="#">Regioni ed endpoint</a> nella Riferimenti generali di AWS. | 28 giugno 2018    |



## Aggiornamenti precedenti

La tabella seguente descrive le modifiche importanti apportate in ogni versione della Guida per gli sviluppatori AWS Lambda prima di giugno 2018.

| Modifica                                                    | Descrizione                                                                                                                                                                                                                                                   | Data             |
|-------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| Supporto runtime per Node.js il runtime 8.10                | AWS Lambda ora supporta la versione Node.js runtime 8.10. Per ulteriori informazioni, consulta <a href="#">Creazione di funzioni Lambda con Node.js</a> .                                                                                                     | 2 aprile 2018    |
| ID di revisione di alias e funzioni                         | AWS Lambda ora supporta gli ID di revisione sulle versioni e sugli alias delle funzioni. È possibile utilizzare questi ID per monitorare e applicare gli aggiornamenti condizionali quando si aggiornano la versione di una funzione o le risorse dell'alias. | 25 gennaio 2018  |
| Supporto del runtime per Go e .NET 2.0                      | AWS Lambda ha aggiunto il supporto di runtime per Go e .NET 2.0. Per ulteriori informazioni, consultare <a href="#">Compilazione di funzioni Lambda con Go</a> e <a href="#">Compilazione di funzioni Lambda con C#</a> .                                     | 15 gennaio 2018  |
| Nuovo progetto della console                                | AWS Lambda ha introdotto una nuova console Lambda per semplificare l'esperienza e ha aggiunto un Cloud9 Code Editor per migliorare le capacità di debug e revisione del codice funzionale.                                                                    | 30 novembre 2017 |
| Impostazione dei limiti di concorrenza per singole funzioni | AWS Lambda ora supporta l'impostazione di limiti di concorrenza per le singole funzioni. Per ulteriori informazioni, consulta <a href="#">Configurazione della simultaneità riservata per una funzione</a> .                                                  | 30 novembre 2017 |
| Trasferimento del traffico con gli alias                    | AWS Lambda ora supporta lo spostamento del traffico con alias. Per ulteriori informazioni, consulta <a href="#">Creare una implementazione continua con alias ponderati</a> .                                                                                 | 28 novembre 2017 |

| Modifica                           | Descrizione                                                                                                                                                                                                                                                                                                                                | Data             |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| Distribuzione graduale del codice  | AWS Lambda ora supporta l'implementazione sicura di nuove versioni della funzione Lambda utilizzando Code Deploy. Per ulteriori informazioni, consultare <a href="#">Distribuzione graduale del codice</a> .                                                                                                                               | 28 novembre 2017 |
| Regione Cina (Pechino)             | AWS Lambda è ora disponibile nella regione Cina (Pechino). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina <a href="#">Regioni ed endpoint</a> nella Riferimenti generali di AWS.                                                                                                                     | 9 novembre 2017  |
| Introduzione a SAM Local           | AWS Lambda introduce SAM Local (ora noto come SAM CLI), uno AWS CLI strumento che fornisce un ambiente in cui sviluppare, testare e analizzare le applicazioni serverless localmente prima di caricarle nel runtime Lambda. Per ulteriori informazioni, consultare <a href="#">Esecuzione di test e debug di applicazioni serverless</a> . | 11 agosto 2017   |
| Regione Canada (Centrale)          | AWS Lambda è ora disponibile nella regione Canada (centrale). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina <a href="#">Regioni ed endpoint</a> nella Riferimenti generali di AWS.                                                                                                                  | 22 giugno 2017   |
| South America (São Paulo) Region   | AWS Lambda è ora disponibile nella regione Sud America (San Paolo). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina <a href="#">Regioni ed endpoint</a> nella Riferimenti generali di AWS.                                                                                                            | 6 giugno 2017    |
| AWS Lambda supporto per AWS X-Ray. | Lambda introduce il supporto per X-Ray, che consente di rilevare, analizzare e ottimizzare i problemi di prestazioni con le applicazioni Lambda. Per ulteriori informazioni, consulta <a href="#">Visualizza le invocazioni delle funzioni Lambda utilizzando AWS X-Ray</a> .                                                              | 19 aprile 2017   |

| Modifica                                                                                                                                   | Descrizione                                                                                                                                                                                                                                                                                                                                                                | Data             |
|--------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| Asia Pacific (Mumbai) Region                                                                                                               | AWS Lambda è ora disponibile nella regione Asia Pacifico (Mumbai). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina <a href="#">Regioni ed endpoint</a> nella Riferimenti generali di AWS.                                                                                                                                             | 28 marzo 2017    |
| AWS Lambda ora supporta il Node.js runtime v6.10                                                                                           | AWS Lambda aggiunto il supporto per il Node.js runtime v6.10. Per ulteriori informazioni, consulta <a href="#">Creazione di funzioni Lambda con Node.js</a> .                                                                                                                                                                                                              | 22 marzo 2017    |
| Europe (London) Region                                                                                                                     | AWS Lambda è ora disponibile nella regione Europa (Londra). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina <a href="#">Regioni ed endpoint</a> nella Riferimenti generali di AWS.                                                                                                                                                    | 1 febbraio 2017  |
| AWS Lambda supporto per il runtime .NET, Lambda @Edge (Preview), Dead Letter Queues e distribuzione automatica di applicazioni serverless. | AWS Lambda è stato aggiunto il supporto per C#. Per ulteriori informazioni, consulta <a href="#">Compilazione di funzioni Lambda con C#</a> .<br><br>Lambda @Edge consente di eseguire funzioni Lambda nelle AWS edge location in risposta agli eventi. CloudFront Per ulteriori informazioni, consulta <a href="#">Personalizzazione sul perimetro con Lambda @Edge</a> . | 3 dicembre 2016  |
| AWS Lambda aggiunge Amazon Lex come fonte di eventi supportata.                                                                            | Utilizzando Lambda e Amazon Lex, è possibile costruire rapidamente chat bot per vari servizi come Slack e Facebook.                                                                                                                                                                                                                                                        | 30 novembre 2016 |
| US West (N. California) Region                                                                                                             | AWS Lambda è ora disponibile nella regione Stati Uniti occidentali (California settentrionale). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina <a href="#">Regioni ed endpoint</a> nella Riferimenti generali di AWS.                                                                                                                | 21 novembre 2016 |

| Modifica                                                                                                                                                                                                          | Descrizione                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Data             |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| È stata introdotta la funzionalità AWS SAM per la creazione e la distribuzione di Lambda-based applicazioni e l'utilizzo delle variabili di ambiente per le impostazioni di configurazione delle funzioni Lambda. | <p>AWS SAM: è ora possibile utilizzare il AWS SAM per definire la sintassi per esprimere le risorse all'interno di un'applicazione serverless. Per distribuire l'applicazione, è sufficiente specificare le risorse necessarie come parte dell'applicazione, insieme alle politiche di autorizzazione associate in un file AWS CloudFormation modello (scritto in JSON o YAML), impacchettare gli artefatti di distribuzione e distribuire il modello.</p> <p>Variabili di ambiente: si possono utilizzare le variabili di ambiente per specificare le impostazioni di configurazione per la funzione Lambda al di fuori del codice della funzione. Per ulteriori informazioni, consultare <a href="#">Utilizzo delle variabili di ambiente Lambda</a>.</p> | 18 novembre 2016 |
| Regione Asia Pacifico (Seoul)                                                                                                                                                                                     | AWS Lambda è ora disponibile nella regione Asia Pacifico (Seoul). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina <a href="#">Regioni ed endpoint</a> nella Riferimenti generali di AWS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | 29 agosto 2016   |
| Asia Pacific (Sydney) Region                                                                                                                                                                                      | Lambda è ora disponibile nella regione Asia Pacifico (Sydney). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina <a href="#">Regioni ed endpoint</a> nella Riferimenti generali di AWS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 23 giugno 2016   |
| Aggiornamenti alla console Lambda                                                                                                                                                                                 | La console Lambda è stata aggiornata per semplificare il processo di creazione dei ruoli.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 23 giugno 2016   |
| AWS Lambda ora supporta il Node.js runtime v4.3                                                                                                                                                                   | AWS Lambda aggiunto il supporto per il Node.js runtime v4.3. Per ulteriori informazioni, consulta <a href="#">Creazione di funzioni Lambda con Node.js</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 07 aprile 2016   |
| Regione Europa (Francoforte)                                                                                                                                                                                      | Lambda è ora disponibile nella regione Europa (Francoforte). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina <a href="#">Regioni ed endpoint</a> nella Riferimenti generali di AWS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 14 marzo 2016    |

| Modifica                                                                                                                                    | Descrizione                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Data             |
|---------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| Supporto per VPC                                                                                                                            | Ora è possibile configurare una funzione Lambda per accedere alle risorse del proprio VPC. Per ulteriori informazioni, consultare <a href="#">Consentire alle funzioni Lambda l'accesso alle risorse in un Amazon VPC</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | 11 febbraio 2016 |
| Il runtime Lambda è stato aggiornato.                                                                                                       | L' <a href="#">ambiente di esecuzione</a> è stato aggiornato.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | 4 novembre 2015  |
| Supporto per il controllo delle versioni, Python per lo sviluppo di codice per le funzioni Lambda, eventi pianificati e aumento del runtime | <p>Ora è possibile sviluppare il codice della funzione Lambda tramite Python. Per ulteriori informazioni, consultare <a href="#">Compilazione di funzioni Lambda con Python</a>.</p> <p>Funzione Versioni multiple: è possibile gestire una o più versioni della funzione Lambda. La funzione Versioni multiple consente di controllare quale versione della funzione Lambda viene eseguita in ambienti diversi (ad esempio ambienti di sviluppo, test o produzione). Per ulteriori informazioni, consultare <a href="#">Gestire le versioni delle funzioni Lambda</a>.</p> <p>Eventi pianificati: si può anche configurare Lambda in modo da richiamare il proprio codice a intervalli regolari e pianificati utilizzando la console Lambda. È possibile specificare una frequenza fissa (numero di ore, giorni o settimane) oppure un'espressione Cron. Per ulteriori informazioni, consulta <a href="#">Richiamo di una funzione Lambda su una pianificazione</a>.</p> <p>Aumento del runtime: è ora possibile configurare le funzioni Lambda per essere eseguite per un massimo di cinque minuti consentendo tempi di esecuzione superiori per alcune funzioni, ad esempio caricamento di grandi volumi di dati e processi di elaborazione.</p> | 08 ottobre 2015  |

| Modifica                                                                                     | Descrizione                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Data           |
|----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| Supporto per DynamoDB Streams                                                                | DynamoDB Streams è ora disponibile a livello generale e può essere utilizzato in tutte le regioni in cui è disponibili le DynamoDB. È possibile utilizzare DynamoDB Streams per la tabella in uso e utilizzare una funzione Lambda come trigger per tale tabella. I trigger sono operazioni personalizzate che si effettuano in risposta ad aggiornamenti eseguiti sulla tabella DynamoDB. Per un esempio di procedura guidata, consulta la pagina <a href="#">Tutorial: Utilizzo AWS Lambda con stream Amazon DynamoDB</a> .                                                                                                                                                                                                                                                                                                                                          | 14 luglio 2015 |
| Lambda ora supporta l'invocazione di funzioni Lambda con i client. REST-compatible           | <p>Fino ad ora, per richiamare la funzione Lambda dalla tua applicazione web, mobile o IoT erano necessari gli AWS SDK (ad esempio, SDK per Java, AWS SDK per Android o AWS SDK per iOS). AWS Ora, Lambda supporta l'invocazione di una funzione Lambda con REST-compatible i client tramite un'API personalizzata che puoi creare utilizzando Amazon API Gateway. È possibile inviare richieste all'URL dell'endpoint della funzione Lambda. Puoi configurare la sicurezza sull'endpoint per consentire l'accesso aperto, utilizzare AWS Identity and Access Management (IAM) per autorizzare l'accesso o utilizzare le chiavi API per misurare l'accesso alle tue funzioni Lambda da parte di altri.</p> <p>Per un esercizio di Nozioni di base di esempio, consultare e <a href="#">Richiamo di funzione Lambda utilizzando un endpoint Gateway Amazon API</a>.</p> | 09 luglio 2015 |
| La console Lambda ora fornisce piani con cui creare facilmente e funzioni Lambda e testarle. | La console Lambda fornisce un set di piani. Ogni blueprint fornisce un esempio di configurazione dell'origine degli eventi e un codice di esempio per la funzione Lambda che puoi utilizzare per creare facilmente applicazioni. Lambda-based Tutti gli esercizi delle Nozioni di base di Lambda ora utilizzano i piani.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | 09 luglio 2015 |

| Modifica                                                                                                                                   | Descrizione                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Data             |
|--------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| Lambda ora supporta Java per la creazione delle funzioni Lambda.                                                                           | Ora è possibile creare codice Lambda in Java. Per ulteriori informazioni, consultare <a href="#">Compilazione di funzioni Lambda con Java</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 15 giugno 2015   |
| Lambda ora supporta la specifica di un oggetto Amazon S3 come funzione .zip durante la creazione o l'aggiornamento di una funzione Lambda. | È possibile caricare un pacchetto di distribuzione della funzione Lambda (file .zip) in un bucket Amazon S3 nella stessa regione in cui si desidera creare una funzione Lambda, quindi specificare il nome del bucket e della chiave dell'oggetto quando si crea o si aggiorna una funzione Lambda.                                                                                                                                                                                                                                                                                                                                                                            | 28 maggio 2015   |
| Disponibilità generale di Lambda con supporto aggiunto per back-end per dispositivi mobili                                                 | Lambda è ora disponibile a livello generale per l'utilizzo in ambiente di produzione. Questa versione introduce inoltre nuove funzionalità che semplificano la creazione mediante Lambda di back-end per dispositivi mobili, tablet e Internet of Things (IoT) che scalano automaticamente senza provisioning o gestione dell'infrastruttura. Lambda ora supporta sia eventi in tempo reale (sincroni) che asincroni. Funzionalità aggiuntive includono una configurazione e una gestione più semplici delle origini eventi. Il modello di autorizzazione e di programmazione sono stati semplificati grazie all'introduzione di policy per le risorse per le funzioni Lambda. | 9 aprile 2015    |
| Versione di anteprima                                                                                                                      | Versione di anteprima della Guida per gli sviluppatori di AWS Lambda.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | 13 novembre 2014 |

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.