Guida per gli sviluppatori

AWS Lambda

AWS Lambda: Guida per gli sviluppatori

Table of Contents

Che cos'è AWS Lambda?

È possibile utilizzarlo AWS Lambda per eseguire il codice senza effettuare il provisioning o gestire i server.

Lambda esegue il codice su un'infrastruttura di elaborazione ad alta disponibilità e gestisce tutta l'amministrazione delle risorse di elaborazione, compresa la manutenzione del server e del sistema operativo, il provisioning e la scalabilità automatica della capacità e la registrazione. Con Lambda, tutto quello che occorre fare è fornire il proprio codice in uno dei runtime di linguaggio supportati da Lambda.

Il codice viene organizzato in funzioni Lambda. Il servizio Lambda esegue la funzione solo quando necessario e si dimensiona automaticamente. Verrà addebitato soltanto il tempo di calcolo utilizzato e non verrà addebitato alcun costo quando il codice non è in esecuzione. Per ulteriori informazioni, consulta la sezione Prezzi di AWS Lambda.

> **Tip**
>
> Per scoprire come creare soluzioni serverless, consulta la Guida allo sviluppo serverless.

Quando usare Lambda

Lambda è un servizio di calcolo ideale per scenari applicativi che richiedono un aumento rapido quando occorre una maggiore capacità e una riduzione a zero quando non è necessaria. Ad esempio, si può utilizzare Lambda per:

- Elaborazione di file: utilizza Amazon Simple Storage Service (Amazon S3) per avviare l'elaborazione dei dati Lambda in tempo reale dopo un caricamento.
- Elaborazione in streaming: utilizza Lambda e Amazon Kinesis per elaborare dati in streaming in tempo reale per il monitoraggio delle attività delle applicazioni, l'elaborazione degli ordini delle transazioni, l'analisi dei clickstream, la pulizia dei dati, il filtraggio dei log, l'indicizzazione, l'analisi dei social media, la telemetria dei dati dei dispositivi Internet delle cose (IoT) e la misurazione.
- Applicazioni Web: combina Lambda con altri AWS servizi per creare potenti applicazioni Web che scalano automaticamente verso l'alto e verso il basso e vengono eseguite in una configurazione ad alta disponibilità su più data center.

- Back-end IoT: crea back-end serverless utilizzando Lambda per gestire richieste di API Web, per dispositivi mobili, IoT e di terze parti.
- Back-end per dispositivi mobili: crea back-end utilizzando Lambda e Gateway Amazon API per autenticare ed elaborare le richieste API. AWS Amplify Usalo per integrarti facilmente con i tuoi frontend iOS, Android, Web e React Native.

Quando si utilizza Lambda, si è responsabili solo del proprio codice. Lambda gestisce il parco istanze di calcolo che offre un bilanciamento di memoria, CPU, rete e altre risorse necessarie per eseguire il codice. Poiché è Lambda a gestire queste risorse, non è possibile accedere alle istanze di calcolo o personalizzare il sistema operativo sui runtime forniti. Lambda svolge attività operative e amministrative per conto dell'utente, tra cui la gestione della capacità, il monitoraggio e la registrazione delle funzioni Lambda.

Funzionalità principali

Le seguenti funzionalità chiave consentono di sviluppare applicazioni Lambda dimensionabili, sicure e facilmente estendibili:

Variabili di ambiente

: Utilizza le variabili di ambiente per regolare il comportamento della funzione senza aggiornare il codice.

Versioni

: Gestisci l'implementazione delle funzioni con le versioni, in modo che, ad esempio, una nuova funzione possa essere utilizzata per il beta testing senza influire sugli utenti della versione di produzione stabile.

Immagini di container

: Crea un'immagine contenitore per una funzione Lambda utilizzando un'immagine di base AWS fornita o un'immagine di base alternativa in modo da poter riutilizzare gli strumenti del contenitore esistenti o distribuire carichi di lavoro più grandi che si basano su dipendenze considerevoli, come l'apprendimento automatico.

Livelli

: Confeziona le librerie e altre dipendenze per ridurre le dimensioni degli archivi di implementazione e accelerare l'implementazione del codice.

Estensioni Lambda

Potenzia le funzioni Lambda con strumenti per il monitoraggio, l'osservabilità, la sicurezza e la governance.

URL delle funzioni

Aggiungi un endpoint HTTP(S) dedicato alla funzione Lambda.

Streaming delle risposte

Configura gli URL delle funzioni Lambda per trasmettere i payload di risposta ai client dalle funzioni Node.js, per migliorare le prestazioni del time to first byte (TTFB) o per restituire payload più grandi.

Controlli di simultaneità e dimensionamento

Applica un controllo granulare al dimensionamento e alla velocità di reazione delle applicazioni di produzione.

Firma del codice

Verifica che solo gli sviluppatori approvati pubblichino codice inalterato e affidabile nelle tue funzioni Lambda.

Reti private

Crea una rete privata per le risorse come database, istanze di cache o servizi interni.

Accesso al file system

Configura una funzione per montare un Amazon Elastic File System (Amazon EFS) in una directory locale, in modo che il codice della funzione possa accedere alle risorse condivise e modificarle in modo sicuro e con un'elevata simultaneità.

Lambda SnapStart per Java

Migliora le prestazioni di startup per i runtime Java fino a 10 volte senza costi aggiuntivi, in genere senza modifiche al codice della funzione.

Guida introduttiva a 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, apprenderai alcuni concetti fondamentali di Lambda, ad esempio come passare argomenti a una funzione tramite l'oggetto evento di Lambda. Imparerai anche come restituire gli output di registro dalla tua funzione e come visualizzare i registri delle chiamate della funzione in Logs. CloudWatch

A scopo di semplificazione, crea la funzione utilizzando il runtime Python o 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#, devi creare un pacchetto di implementazione sulla macchina di sviluppo locale e caricarlo su Lambda. Per informazioni sull'implementazione di funzioni in Lambda tramite altri runtime, consulta i link nella sezione the section called "Risorse aggiuntive e fasi successive".

> **Tip**
>
> Per scoprire come creare soluzioni serverless, consulta la Guida allo sviluppo serverless.

Prerequisiti

Registrati per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come procedura

consigliata in materia di sicurezza, assegnate l'accesso amministrativo a un utente e utilizzate solo l'utente root per eseguire attività che richiedono l'accesso da parte dell'utente root.

AWS ti invia un'e-mail di conferma dopo il completamento della procedura di registrazione. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .
2. In IAM Identity Center, concedi l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con le impostazioni predefinite IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accedi come utente con accesso amministrativo

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso ad altri utenti

1. In IAM Identity Center, crea un set di autorizzazioni che segua la migliore pratica di applicazione delle autorizzazioni con privilegi minimi.

   Per istruzioni, consulta Creare un set di autorizzazioni nella Guida per l'utente.AWS IAM Identity Center
2. Assegna gli utenti a un gruppo, quindi assegna l'accesso Single Sign-On al gruppo.

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente.AWS IAM Identity Center

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 funzione stampa anche l'area calcolata, insieme al nome del relativo gruppo di CloudWatch log. Più avanti nel tutorial, imparerai a usare CloudWatch Logs per visualizzare i record di invocazione delle tue funzioni.

Per creare la funzione, usa prima la console per creare una funzione Hello world base. Nella fase successiva, aggiungi il codice della tua funzione.

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 20.x o Python 3.12
6. Lascia l'architettura impostata su x86_64 e scegli Crea funzione.

Lambda crea una funzione 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' Servizi AWS accesso e alle risorse. Per la tua funzione, il ruolo creato da Lambda concede le autorizzazioni di base per la scrittura nei registri. CloudWatch

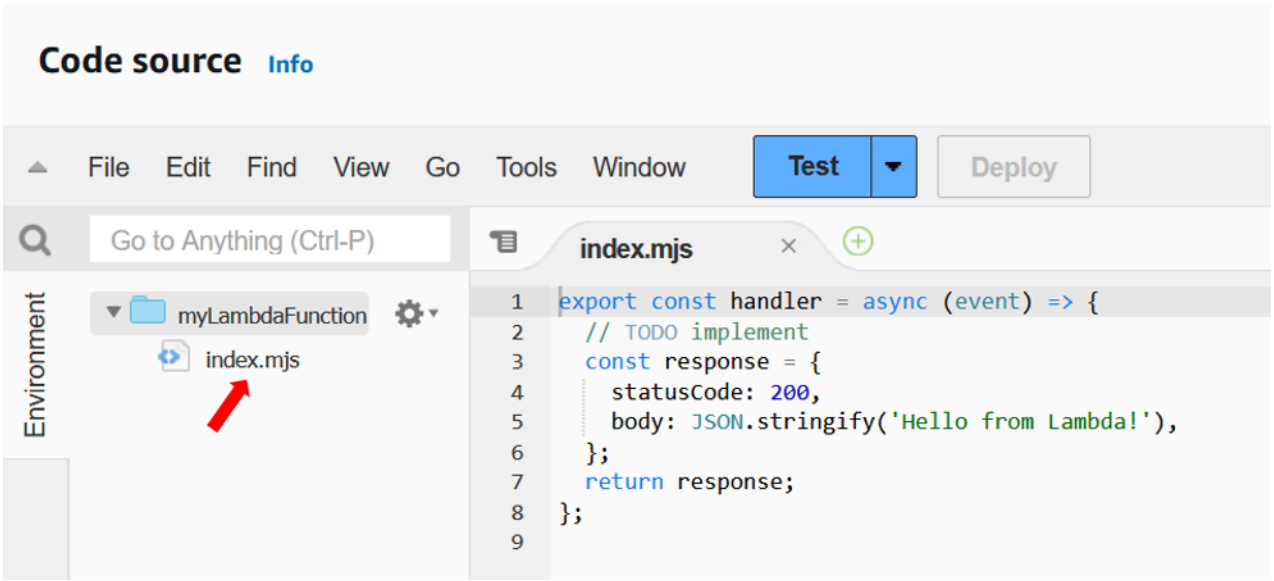
A questo punto, utilizza l'editor di codice integrato nella console per sostituire il codice Hello world creato da Lambda con il codice della tua funzione.

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. Scegli Implementa per aggiornare il codice della funzione. Quando Lambda ha implementato le modifiche, la console visualizza un banner che indica che la funzione è stata aggiornata correttamente.

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 Node.js denominata `handler`. Una funzione Lambda in Node.js può contenere più di una funzione Node.js, 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 `handler` il nome del metodo del gestore per la funzione. Non modificare il nome di questa funzione Node.js. Se modifichi il nome, Lambda non sarà in grado di eseguire il codice quando richiami la funzione.

  Per ulteriori informazioni sul gestore Lambda in Node.js, consulta 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 funzione viene richiamata da un'altra Servizio AWS, l'oggetto evento contiene informazioni sull'evento che ha causato l'invocazione. Ad esempio, se un bucket Amazon Simple Storage Service (Amazon S3) richiama la funzione quando viene caricato un oggetto, l'evento conterrà il nome del bucket Amazon S3 e la chiave dell'oggetto.

In questo esempio verrà creato 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 CloudWatch gruppo di log.

  Per ulteriori informazioni sull'oggetto contestuale Lambda in Node.js, consulta the section called "Context".

- Accesso a Lambda:

  Con Node.js, puoi usare metodi della console come `console.log` e `console.error` per inviare informazioni al log della funzione. Il codice di esempio utilizza `console.log` 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`.

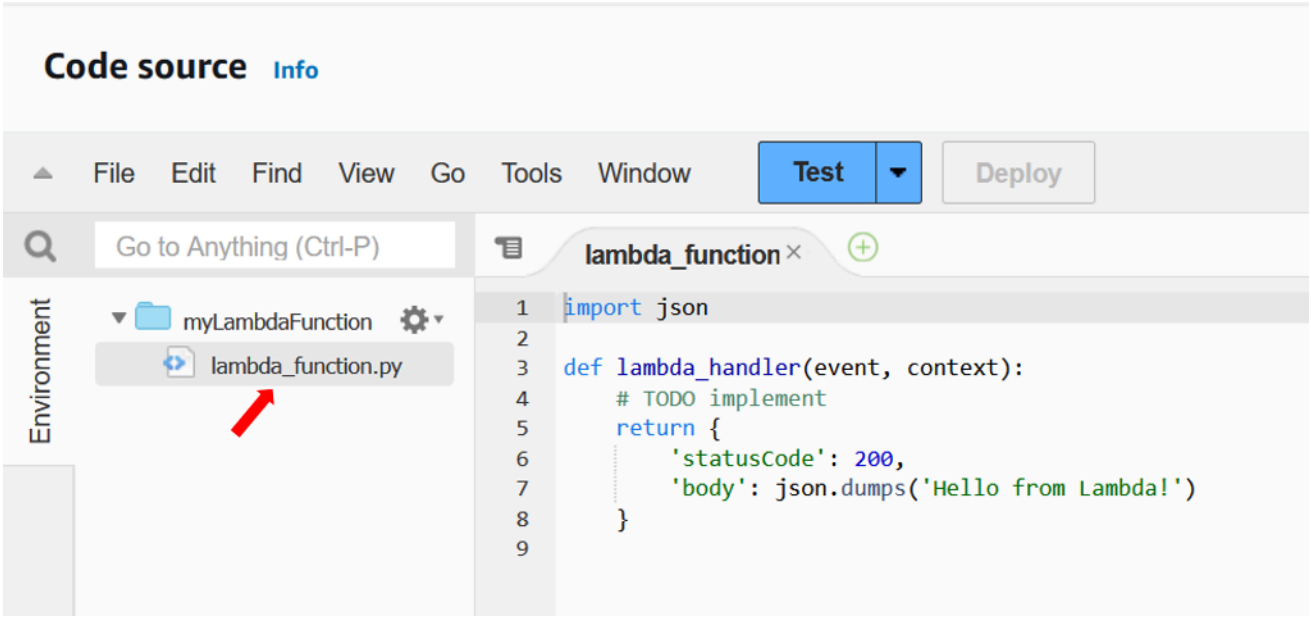
  Per ulteriori informazioni, consulta the section called "Registrazione". Per informazioni sui log in altri runtime, consulta le pagine "Creazione con" per i runtime a cui sei interessato.

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. Scegli Implementa per aggiornare il codice della funzione. Quando Lambda ha implementato le modifiche, la console visualizza un banner che indica che la funzione è stata aggiornata correttamente.

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

  In questo esempio verrà creato 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 CloudWatch gruppo di log.

Per ulteriori informazioni sull'oggetto contestuale Lambda in Python, consulta the section called "Context".

- 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". Per informazioni sui log in altri runtime, consulta le pagine "Creazione con" per i runtime a cui sei interessato.

Richiamare la funzione Lambda con la console

Per richiamare la funzione utilizzando la 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. Nel riquadro Origine del codice, scegli Test.
2. Seleziona Crea nuovo evento.
3. Per Nome evento, inserisci **`myTestEvent`**.
4. Nel pannello Evento JSON, sostituisci i valori predefiniti incollando quanto segue:

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

5. Selezionare Salva.

Ora testate la vostra funzione e utilizzate la console Lambda e CloudWatch Logs per visualizzare i record della chiamata della funzione.

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

- Nel riquadro Origine del codice, scegli Test. Al termine dell'esecuzione della funzione, verranno visualizzati i log della risposta e della funzione nella scheda Risultati dell'esecuzione. Dovresti visualizzare risultati simili a quelli indicati di seguito.

  Node.js

  ```
  Test Event Name
  myTestEvent

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

  Function Logs
  START RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a Version: $LATEST
  2023-08-31T23:39:45.313Z 5c012b0a-18f7-4805-b2f6-40912935034a INFO The area is
   42
  2023-08-31T23:39:45.331Z 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

  ```
  Test Event Name
  myTestEvent

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

  Function Logs
  START RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b Version: $LATEST
  The area is 42
  ```

```
[INFO] 2023-08-31T23:43:26.428Z 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
```

In questo esempio hai richiamato il codice utilizzando la funzionalità di test della console. Ciò implica che puoi visualizzare i risultati dell'esecuzione della funzione direttamente nella console. Quando la funzione viene richiamata all'esterno della console, è necessario utilizzare Logs. CloudWatch

Per visualizzare i record di invocazione della funzione in Logs CloudWatch

1. Apri la pagina Log groups della console. CloudWatch
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. Nella scheda Flussi di log, scegli il flusso di log per l'invocazione della funzione.

   Verrà visualizzato un output simile al seguente:

   Node.js

```
INIT_START Runtime Version: nodejs:20.v13    Runtime Version ARN:
 arn:aws:lambda:us-
west-2::runtime:e3aaabf6b92ef8755eaae2f4bfdcb7eb8c4536a5e044900570a42bdba7b869d9
START RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20 Version: $LATEST
2023-08-23T22:04:15.809Z    5c012b0a-18f7-4805-b2f6-40912935034a  INFO The area
 is 42
2023-08-23T22:04:15.810Z    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:3.12.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] 2023-09-01T00:05:22.464Z 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.

Come eliminare una funzione Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegli Operazioni > Elimina.
4. Nella finestra di dialogo Delete function (Elimina funzione), digita delete, quindi seleziona Delete (Elimina).

Come eliminare il gruppo di log

1. Apri la pagina Registra gruppi della CloudWatch console.
2. Scegli il gruppo di log della funzione (`/aws/lambda/my-function`).
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. Scegli Elimina.
4. Nella finestra di dialogo Delete role (Elimina ruolo), immetti il nome del ruolo, quindi scegli Delete (Elimina).

Puoi automatizzare la creazione e la pulizia di funzioni, gruppi di log e ruoli con AWS CloudFormation and the AWS Command Line Interface ()AWS CLI.

Risorse aggiuntive e fasi successive

Una volta creata e testata una semplice funzione Lambda con la console, effettua le seguenti fasi successive:

- Scopri come aggiungere dipendenze al codice e implementarlo utilizzando un pacchetto di implementazione .zip. Scegli tra i seguenti link per le lingue a cui sei interessato.

  Node.js

  Consulta the section called “Implementazione di archivi di file .zip”.

  Typescript

  Per informazioni, consulta the section called “Implementare archivi di file .zip”

  Python

  Per informazioni, consulta the section called “Implementazione di archivi di file .zip”

  Ruby

  Per informazioni, consulta the section called “Implementazione di archivi di file .zip”

  Java

  Per informazioni, consulta the section called “Implementazione di archivi di file .zip”

  Go

  Per informazioni, consulta the section called “Implementazione di archivi di file .zip”

  C#

  Consulta la sezione the section called “Pacchetto di implementazione”

- Esegui il tutorial Utilizzo di un trigger Amazon S3 per richiamare una funzione Lambda per apprendere come configurare una funzione Lambda affinché venga richiamata da un altro Servizio AWS.
- Scegli uno dei seguenti tutorial per un esempio più complesso di utilizzo di Lambda con altri Servizi AWS.
  - Utilizzo di Lambda con un gateway API: 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.

AWS Lambda fondazioni

La funzione Lambda è la risorsa principale del servizio Lambda.

Puoi configurare le tue funzioni utilizzando la console Lambda, l'API Lambda o. AWS CloudFormation AWS SAM Basta creare il codice per la funzione e caricarlo utilizzando un pacchetto di implementazione. Quando si verifica un evento, Lambda richiama la funzione. Lambda esegue più istanze della funzione in parallelo, governate da limiti di simultaneità e dimensionamento.

Argomenti

Concetti Lambda

Lambda esegue istanze della funzione per elaborare gli eventi. Puoi richiamare la funzione direttamente tramite l'API Lambda oppure configurare un servizio o una risorsa AWS per richiamarla.

Concetti

- Funzione
- Trigger
- Evento
- Ambiente di esecuzione
- Architettura del set di istruzioni
- Pacchetto di implementazione
- Runtime
- Livello
- Estensione
- Simultaneità
- Qualificatore
- Destinazione

Funzione

Una funzione è una risorsa che è possibile invocare per eseguire il codice in Lambda. Una funzione dispone di codice per elaborare gli eventi trasmessi da te o inviati da altri servizi AWS.

Trigger

Un trigger è una risorsa o una configurazione che richiama una funzione Lambda. Le attivazioni includono AWS i servizi che possono essere configurati per richiamare una funzione e le mappature delle fonti eventi. Una mappatura delle origini eventi è una risorsa in Lambda che legge gli elementi da un flusso o da una coda e invoca una funzione. Per ulteriori informazioni, consultare Comprensione dei metodi di invocazione della funzione Lambda e Richiamare Lambda con eventi di altri servizi AWS.

Evento

Un evento è un documento in formato JSON formattato che contiene i dati che una funzione Lambda deve elaborare. Il runtime converte l'evento in un oggetto e lo passa al codice della funzione. Quando si invoca una funzione, si determina la struttura e il contenuto dell'evento.

Example evento personalizzato – dati meteo

```
{
  "TemperatureK": 281,
  "WindKmh": -3,
  "HumidityPct": 0.55,
  "PressureHPa": 1020
}
```

Quando un servizio AWS richiama la funzione, il servizio definisce la forma dell'evento.

Example Evento di servizio – Notifica Amazon SNS

```
{
  "Records": [
    {
      "Sns": {
        "Timestamp": "2019-01-02T12:45:07.000Z",
        "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkAi6RibDsvpi+tE/1+82j...65r==",
        "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
        "Message": "Hello from SNS!",
        ...
```

Per ulteriori informazioni sugli eventi dei servizi AWS, consulta Richiamare Lambda con eventi di altri servizi AWS.

Ambiente di esecuzione

Un ambiente di esecuzione fornisce un ambiente di runtime sicuro e isolato per la funzione Lambda. Un ambiente di esecuzione gestisce i processi e le risorse necessari per eseguire la funzione. L'ambiente di esecuzione fornisce il supporto del ciclo di vita per la funzione e per tutte le estensioni ad essa associate.

Per ulteriori informazioni, consulta Ambiente di esecuzione Lambda.

Architettura del set di istruzioni

L'architettura del set di istruzioni 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.

Per ulteriori informazioni, consulta Configurazione dell'architettura del set di istruzioni per una funzione Lambda.

Pacchetto di implementazione

Implementa il codice della funzione Lambda tramite un pacchetto di implementazione. Lambda supporta due tipi di pacchetti di implementazione:

- Un archivio di file .zip contenente il codice della funzione e le relative dipendenze. Lambda fornisce il sistema operativo e il runtime per la funzione.
- Un'immagine di container compatibile con la specifica OCI (Open Container Initiative). Aggiungi il codice della funzione e le dipendenze all'immagine. È inoltre necessario includere il sistema operativo e un runtime Lambda.

Per ulteriori informazioni, consulta Pacchetti di implementazione Lambda.

Runtime

Il runtime fornisce un ambiente specifico del linguaggio di programmazione che viene eseguito in un ambiente di esecuzione. Il runtime inoltra eventi di chiamata, informazioni di contesto e risposte tra Lambda e la funzione. Puoi utilizzare i runtime forniti da Lambda o puoi crearne uno personalizzato. Se impacchetti il codice come archivio di file .zip, devi configurare la funzione per utilizzare un runtime che corrisponda al linguaggio di programmazione. Includi il runtime per un'immagine del container nel momento in cui la crei.

Per ulteriori informazioni, consulta Runtime Lambda.

Livello

Un livello Lambda è un archivio di file con estensione .zip che può contenere codice o dati aggiuntivi. Un livello può contenere librerie, un runtime personalizzato, dati o file di configurazione.

I livelli offrono un metodo pratico per impacchettare le librerie e altre dipendenze che è possibile utilizzare insieme alle funzioni Lambda. L'uso dei livelli riduce le dimensioni degli archivi di implementazione caricati e accelera la distribuzione del codice. I livelli promuovono anche la condivisione del codice e la separazione delle responsabilità in modo da poter iterare più velocemente la scrittura della logica di business.

Puoi includere fino a cinque livelli per funzione. I livelli influiscono sulle quote delle dimensioni di implementazione standard di Lambda. Quando si include un livello in una funzione, il contenuto viene estratto nella directory `/opt` nell'ambiente di esecuzione.

Per impostazione predefinita, i livelli creati sono privati per il tuo account AWS. Puoi decidere di condividere un livello con altri account o di rendere il livello pubblico. Se le funzioni consumano un livello che un altro account ha pubblicato, le funzioni possono continuare a utilizzare la versione del livello dopo che questo è stato eliminato, o dopo che l'autorizzazione ad accedere al livello è stata revocata. Non è tuttavia possibile creare una nuova funzione o aggiornare le funzioni utilizzando la versione di un livello eliminato.

Le funzioni implementate come immagine container non utilizzano i livelli. Il runtime, le librerie e altre dipendenze preferite vanno impacchettati invece nell'immagine del container nel momento quando la crei.

Per ulteriori informazioni, consulta Livelli Lambda.

Estensione

Le estensioni Lambda consentono di aumentare le funzioni. Ad esempio, è possibile utilizzare le estensioni per integrare le funzioni con gli strumenti di monitoraggio, osservabilità, sicurezza e strumenti di governance preferiti. È possibile scegliere tra un ampio set di strumenti forniti dai Partner AWS Lambda oppure creare estensioni Lambda personalizzate.

Un'estensione interna viene eseguita nel processo di runtime e condivide lo stesso ciclo di vita del runtime. Un'estensione esterna viene eseguita come processo separato nell'ambiente di esecuzione. L'estensione esterna viene inizializzata prima che la funzione venga richiamata, viene eseguita in parallelo con il runtime della funzione e continua a funzionare al termine dell'invocazione della funzione.

Per ulteriori informazioni, consulta Potenzia le funzioni Lambda utilizzando le estensioni Lambda.

Simultaneità

Simultaneità è il numero di richieste che la funzione sta elaborando in un dato momento. Quando la funzione viene richiamata, Lambda effettua il provisioning di un'istanza per elaborare l'evento. Quando il codice della funzione termina l'esecuzione, può gestire un'altra richiesta. Se la funzione viene invocata nuovamente mentre una richiesta è ancora in fase di elaborazione, viene effettuato il provisioning di un'altra istanza, aumentando la simultaneità della funzione.

La simultaneità è soggetta a quote a livello di regione AWS. È anche possibile configurare singole funzioni per limitare la loro simultaneità o per garantire che possano raggiungere un determinato livello di simultaneità. Per ulteriori informazioni, consulta Configurazione della concorrenza riservata per una funzione.

Qualificatore

Quando si richiama o si visualizza una funzione, è possibile includere un qualificatore per specificare una versione o un alias. Una versione è uno snapshot immutabile del codice e della configurazione di una funzione che ha un qualificatore numerico. Ad esempio, `my-function:1`. Un alias è un puntatore a una versione che è possibile aggiornare per eseguire il mapping a una versione diversa o dividere il traffico tra due versioni. Ad esempio, `my-function:BLUE`. Puoi utilizzare versioni e alias contemporaneamente per fornire un'interfaccia stabile ai client per richiamare la funzione.

Per ulteriori informazioni, consulta Versioni delle funzioni Lambda.

Destinazione

Una destinazione è una risorsa AWS in cui Lambda può inviare eventi da una chiamata asincrona. Puoi configurare una destinazione per gli eventi la cui elaborazione non è riuscita. Alcuni servizi supportano anche una destinazione per gli eventi che vengono elaborati correttamente.

Per ulteriori informazioni, consulta Configurazione delle destinazioni per l'invocazione asincrona.

Modello di programmazione Lambda

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.

Quando l'handler termina l'elaborazione del primo evento, il runtime ne invia un altro. 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 storage locale nella directory `/tmp`. Il contenuto della directory rimane quando il contesto di esecuzione è bloccato, fornendo così una cache transitoria utilizzabile per più invocazioni. Per ulteriori informazioni, consulta Ambiente di esecuzione di Lambda.

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 CloudWatch Logs. 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 log. CloudWatch È possibile perdere i dati di log a causa del throttling o, in alcuni casi, quando un'istanza della funzione viene interrotta.

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.

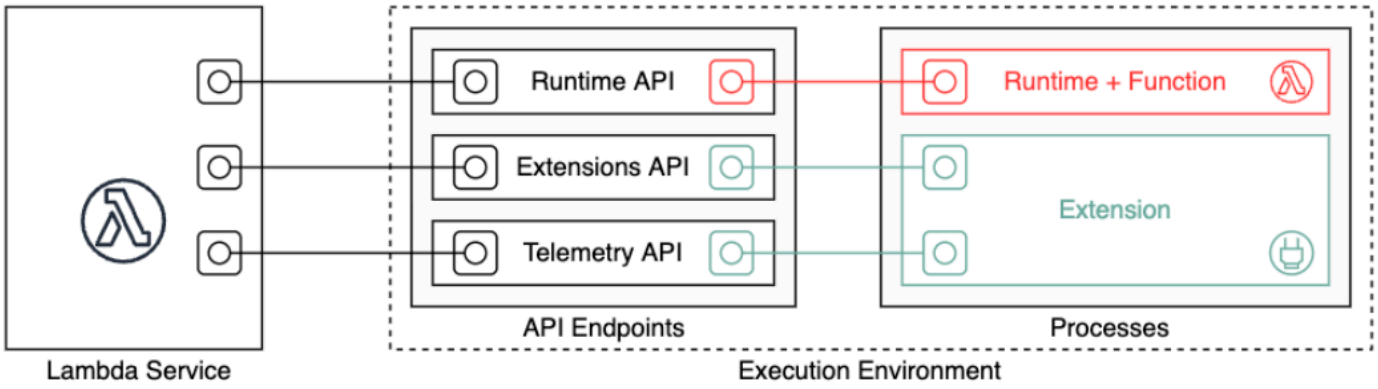
Per un'introduzione pratica al modello di programmazione preferito, consulta i seguenti capitoli.

- Compilazione 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

Ambiente di esecuzione Lambda

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.

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.



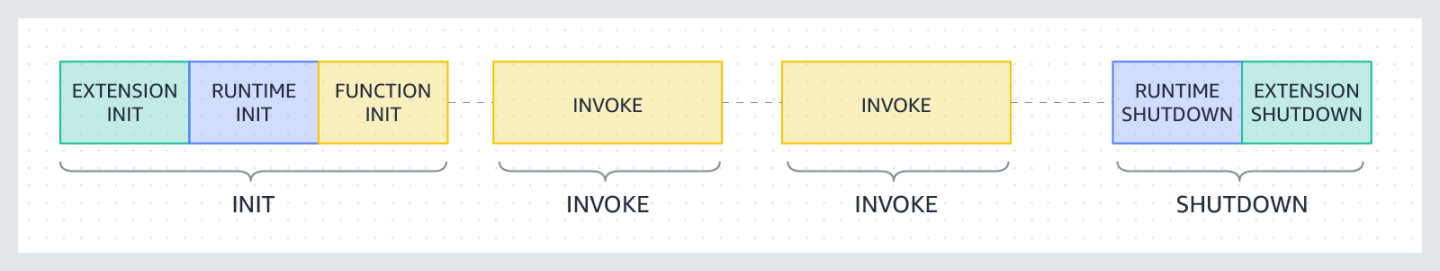
Quando si crea la funzione Lambda, si specificano le informazioni di configurazione, ad esempio la quantità di memoria disponibile e il tempo massimo di esecuzione consentito per la funzione. Lambda utilizza queste informazioni per impostare l'ambiente di esecuzione.

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
- Implementazione dell'apolidia nelle funzioni

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.

Argomenti

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 `beforeCheckpoint` di runtime (solo SnapStart Lambda)

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 Lambda 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 `beforeCheckpoint`, il codice viene eseguito alla fine della fase `Init`.

> **Note**
>
> Il timeout di 10 secondi non si applica alle funzioni che utilizzano la concorrenza fornita o. SnapStart Per la concorrenza e SnapStart le funzioni assegnate, 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 con provisioning, 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. Potrebbero verificarsi lacune tra l'invocazione della funzione e le fasi di inizializzazione. A seconda del runtime della funzione e della configurazione della memoria, può anche verificarsi una latenza delle variabili nella prima invocazione in un ambiente di esecuzione inizializzato.

Per le funzioni che utilizzano la simultaneità on demand, Lambda può inizializzare occasionalmente gli ambienti di esecuzione prima delle richieste di invocazione. Quando ciò si verifica, puoi riscontrare una lacuna temporale 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 SnapStart, a meno che non sia attivata. SnapStart le funzioni vengono sempre emesse. `INIT_REPORT` Per ulteriori informazioni, consulta Monitoraggio per Lambda SnapStart.

Fase di ripristino (solo Lambda SnapStart)

Quando richiami una SnapStartfunzione 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 `afterRestore()`, il codice viene eseguito alla fine della fase `Restore`. Ti sarà addebitata la durata degli hook di runtime `afterRestore()`. 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 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.

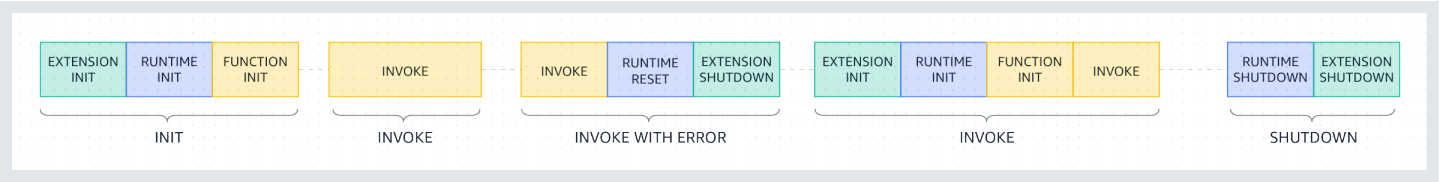
L'impostazione di timeout della funzione limita la durata dell'intera fase `Invoke`. Ad esempio, se si imposta il timeout della funzione come 360 secondi, la funzione e tutte le estensioni devono essere

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.
- Supponi che in questo passaggio la tua funzione presenti un errore di invocazione (ad esempio un timeout della funzione o un errore di runtime). 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.

  > **Note**
  >
  > Il reset Lambda non cancella il contenuto della directory `/tmp` prima della fase di init successiva. Questo comportamento è coerente con la normale fase di arresto.

- 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 init soppressi, Lambda non riporta esplicitamente una fase INIT aggiuntiva nei log. CloudWatch Infatti potresti

notare che la durata nella riga REPORT include una durata INIT aggiuntiva + la durata INVOKE. Ad esempio, supponiamo di visualizzare i seguenti log in: 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 API di telemetria Lambda. 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 soppresse 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`.

Durata: L'intera fase `Shutdown` è limitata a 2 secondi. Se il runtime o qualsiasi estensione non risponde, Lambda lo termina tramite 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 in fase di esecuzione, 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 dell'apolidia nelle funzioni.

Quando la funzione viene richiamata di nuovo, Lambda scongela 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.

Implementazione dell'apolidia nelle funzioni

Quando scrivi il codice della funzione Lambda, considera l'ambiente di esecuzione come stateless, supponendo che esista solo per una singola chiamata. Lambda termina gli ambienti di esecuzione ogni poche ore per consentire gli aggiornamenti e la manutenzione in fase di esecuzione, anche per le funzioni che vengono richiamate continuamente. Inizializza qualsiasi stato richiesto (ad esempio, il recupero di un carrello della spesa da una tabella Amazon DynamoDB) all'avvio della funzione. Prima di uscire, esegui modifiche permanenti ai dati in archivi durevoli come Amazon Simple Storage Service (Amazon S3), DynamoDB o Amazon Simple Queue Service (Amazon SQS). Evita di fare affidamento su strutture di dati esistenti, file temporanei o stati che comprendono invocazioni, come contatori o aggregati. Ciò garantisce che la funzione gestisca ogni chiamata in modo indipendente.

Pacchetti di implementazione Lambda

Il codice della 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.

Argomenti

Immagini di container

Un'immagine di container include il sistema operativo di base, il runtime, le estensioni Lambda, il codice dell'applicazione e le relative dipendenze. È inoltre possibile aggiungere all'immagine dati statici, come i modelli di machine learning.

Lambda fornisce un set di immagini di base open source che è possibile utilizzare per creare l'immagine di container. Per creare e testare le immagini dei contenitori, puoi utilizzare l' AWS Serverless Application Model interfaccia a riga di comando (CLI) o strumenti contenitore nativi come la CLI Docker.AWS SAM

Carica le immagini dei container su Amazon Elastic Container Registry (Amazon ECR), un servizio AWS gestito di registro delle immagini dei container. Per distribuire l'immagine nella tua funzione, specifica l'URL dell'immagine Amazon ECR utilizzando la console Lambda, l'API Lambda, gli strumenti a riga di comando o gli SDK. AWS

Per ulteriori informazioni sulle immagini di container Lambda, consulta Creare una funzione Lambda utilizzando un'immagine del contenitore.

archivi di file .zip

Un archivio di file .zip include il codice dell'applicazione e le relative dipendenze. Quando si creano funzioni utilizzando la console Lambda o un kit di strumenti, Lambda crea automaticamente un archivio di file .zip del codice.

Quando crei funzioni con l'API Lambda, gli strumenti da riga di comando o gli AWS SDK, devi creare un pacchetto di distribuzione. È inoltre necessario creare un pacchetto di distribuzione se la funzione utilizza un linguaggio compilato o per aggiungere dipendenze alla funzione. Per distribuire il codice della tua funzione, carica il pacchetto di distribuzione da Amazon Simple Storage Service (Amazon S3) o dal tuo computer locale.

Puoi caricare un file.zip come pacchetto di distribuzione utilizzando la console Lambda AWS Command Line Interface ,AWS CLI() o su un bucket Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3).

Uso della console Lambda

Nella procedura seguente viene illustrato come caricare un file .zip come pacchetto di distribuzione utilizzando la console Lambda.

Per caricare un file .zip sulla console Lambda

1. Aprire la pagina Functions (Funzioni) nella console Lambda.
2. Seleziona una funzione.
3. Nel riquadro Code source (Origine codice), scegliere Upload from (Carica da) e quindi file .zip.
4. Scegli Carica per selezionare il file .zip locale.
5. Selezionare Salva.

Usando il AWS CLI

È possibile caricare un file.zip come pacchetto di distribuzione utilizzando AWS Command Line Interface (AWS CLI). Per istruzioni specifiche del linguaggio, vedi i seguenti argomenti.

Node.js

- Distribuisci funzioni Lambda in Node.js con archivi di file .zip

Python

- Utilizzo di archivi di file .zip per le funzioni Lambda in Python

Ruby

- Utilizzo di archivi di file .zip per le funzioni Lambda in Ruby

Java

- Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR

Go

- Distribuisci 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

- Implementa le funzioni PowerShell Lambda con archivi di file.zip

Uso di Amazon S3

Puoi caricare un file .zip come pacchetto di implementazione utilizzando Amazon Simple Storage Service (Amazon S3). Per ulteriori informazioni, consulta .

Livelli

Se si distribuisce il codice della funzione utilizzando un archivio di file .zip, è possibile utilizzare i livelli Lambda come meccanismo di distribuzione per librerie, runtime personalizzati e altre dipendenze delle funzioni. I livelli consentono di gestire il codice della funzione in fase di sviluppo indipendentemente dal codice e dalle risorse immutabili utilizzate. È possibile configurare la funzione in modo da utilizzare i livelli creati dall'utente, i livelli AWS forniti dall'utente o i livelli di altri AWS clienti.

Non è possibile utilizzare livelli con immagini di contenitori. Al contrario, inserite il runtime, le librerie e le altre dipendenze preferite nell'immagine del contenitore quando create l'immagine.

Per ulteriori informazioni sui livelli, consulta Livelli Lambda.

Utilizzo di altri AWS servizi per creare un pacchetto di distribuzione

La sezione seguente descrive altri AWS servizi che puoi usare per impacchettare le dipendenze per la tua funzione Lambda.

Pacchetti di distribuzione con librerie C o C++

Se il pacchetto di distribuzione contiene librerie native, puoi creare il pacchetto di distribuzione con AWS Serverless Application Model ()AWS SAM. È possibile utilizzare il `sam build` comando AWS

SAM CLI con `--use-container` per creare il pacchetto di distribuzione. Questa opzione crea un pacchetto di implementazione all'interno di un'immagine Docker compatibile con l'ambiente di esecuzione Lambda.

Per ulteriori informazioni, consulta sam build nella Guida per gli sviluppatori di AWS Serverless Application Model .

Pacchetti di distribuzione superiori a 50 MB

Se la dimensione del pacchetto di implementazione è superiore a 50 MB, carica il codice della funzione e le dipendenze in un bucket Amazon S3.

Puoi creare un pacchetto di distribuzione e caricare il file.zip nel tuo bucket Amazon S3 nella regione in AWS cui desideri creare una funzione Lambda. Quando si crea la funzione Lambda, specificare il nome del bucket S3 e il nome della chiave oggetto sulla console Lambda oppure utilizzando il AWS CLI.

Per creare un bucket utilizzando la console Amazon S3, consulta la sezione Creazione di un bucket nella Guida per l'utente di Amazon Simple Storage Service.

Utilizzo di Lambda con l'infrastructure as code (IaC)

Lambda offre diversi modi per implementare il codice e creare funzioni. Ad esempio, puoi utilizzare la console Lambda o la AWS Command Line Interface (AWS CLI) per creare o aggiornare manualmente le funzioni Lambda. Oltre a queste opzioni manuali, AWS offre una serie di soluzioni per l'implementazione di funzioni Lambda e applicazioni serverless utilizzando l'infrastructure as code (IaC). Con l'IaC, puoi fornire e gestire funzioni Lambda e altre risorse AWS utilizzando il codice anziché servirti di processi e impostazioni manuali.

La maggior parte delle volte, le funzioni Lambda non vengono eseguite in modo isolato. Fanno invece parte di un'applicazione serverless con altre risorse come database, code e spazio di archiviazione. Con l'IaC, è possibile automatizzare i processi di implementazione per implementare e aggiornare in modo rapido e ripetibile intere applicazioni serverless che coinvolgono molte risorse AWS separate. Questo approccio accelera il ciclo di sviluppo, semplifica la gestione della configurazione e garantisce che le risorse vengano implementate sempre allo stesso modo.

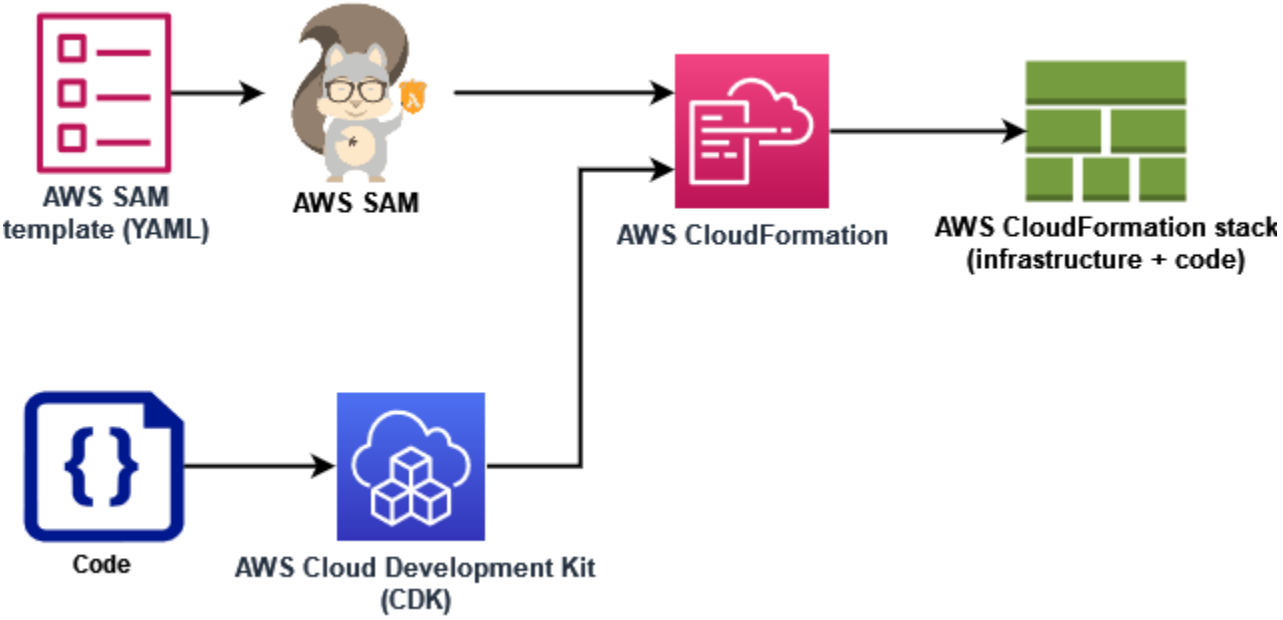
Argomenti

Strumenti di IaC per Lambda

Per implementare funzioni Lambda e applicazioni serverless utilizzando l'IaC, AWS offre una gamma di strumenti e servizi.

AWS CloudFormation è stato il primo servizio offerto da AWS per creare e configurare le risorse cloud. Con AWS CloudFormation, puoi creare modelli di testo per definire l'infrastruttura e il codice. A fronte dell'introduzione di nuovi servizi da parte di AWS e dell'aumento della complessità della creazione dei modelli di AWS CloudFormation, sono stati rilasciati altri due strumenti. AWS SAM è un altro framework basato su modelli per la definizione di applicazioni serverless. Il AWS Cloud Development Kit (AWS CDK) è un approccio basato sul codice per la definizione e il provisioning dell'infrastruttura utilizzando costrutti di codice in molti linguaggi di programmazione popolari.

Con AWS SAM e il AWS CDK, AWS CloudFormation opera dietro le quinte per creare e implementare l'infrastruttura. Il diagramma seguente illustra la relazione tra questi strumenti, mentre i paragrafi successivi al diagramma ne illustrano le funzionalità principali.



- AWS CloudFormation- Modella e configura insieme AWS le tue risorse utilizzando un modello YAML o JSON che descrive le tue risorse e le loro proprietà. CloudFormation CloudFormation fornisce le risorse in modo sicuro e ripetibile, consentendoti di creare frequentemente l'infrastruttura e le applicazioni senza passaggi manuali. Quando si modifica la configurazione, CloudFormation determina le operazioni corrette da eseguire per aggiornare lo stack. CloudFormation può persino ripristinare le modifiche.
- AWS Serverless Application Model (AWS SAM): AWS SAM è un framework open source per la definizione di applicazioni serverless. I modelli AWS SAM utilizzano una sintassi abbreviata per definire funzioni, API, database e strumenti di mappatura dell'origine degli eventi con poche righe di testo (YAML) per ciascuna risorsa. Durante l'implementazione, AWS SAM trasforma ed espande la sintassi AWS SAM nella sintassi AWS CloudFormation. Per questo motivo, è possibile aggiungere qualsiasi CloudFormation sintassi ai AWS SAM modelli. Ciò offre AWS SAM tutta la potenza di CloudFormation, ma con meno righe di configurazione.
- AWS Cloud Development Kit (AWS CDK)- ConAWS CDK, definisci la tua infrastruttura utilizzando costrutti di codice e eseguendo AWS CloudFormation il provisioning della stessa. AWS CDKconsente di modellare l'infrastruttura applicativa con Python TypeScript, Java, .NET e Go (in Developer Preview) utilizzando l'IDE, gli strumenti di test e i modelli di flusso di lavoro esistenti.

Ottieni tutti i vantaggi di AWS CloudFormation, tra cui l'implementazione ripetibile, il rollback semplificato e il rilevamento delle deviazioni.

Inoltre, AWS fornisce un servizio chiamato Strumento AWS per la creazione di applicazioni per sviluppare modelli di IaC utilizzando una semplice interfaccia grafica. Con Strumento per la creazione di applicazioni, è possibile progettare un'architettura applicativa trascinando, raggruppando e connettendo i Servizi AWS in un canvas visivo. Strumento per la creazione di applicazioni crea quindi un modello AWS SAM oppure un modello AWS CloudFormation a partire dal tuo progetto che puoi utilizzare per implementare l'applicazione.

Nella sezione the section called “Guida introduttiva all'IaC per Lambda” seguente, utilizzi Strumento per la creazione di applicazioni per sviluppare un modello per un'applicazione serverless basata su una funzione Lambda esistente.

Guida introduttiva all'IaC per Lambda

In questo tutorial, puoi iniziare a utilizzare l'IaC con Lambda creando un modello AWS SAM da una funzione Lambda esistente e successivamente creando un'applicazione serverless in Strumento per la creazione di applicazioni mediante l'aggiunta di altre risorse AWS.

Se preferisci iniziare con un tutorial AWS SAM oppure AWS CloudFormation per imparare a lavorare con i modelli senza utilizzare Strumento per la creazione di applicazioni, troverai i link ad altre risorse nella sezione the section called “Passaggi successivi” al fondo di questa pagina.

Man mano che procedi in questo tutorial, imparerai alcuni concetti fondamentali, ad esempio come vengono specificate le risorse AWS in AWS SAM. Imparerai anche a utilizzare Strumento per la creazione di applicazioni per creare un'applicazione serverless che puoi implementare tramite AWS SAM oppure AWS CloudFormation.

Per completare questo tutorial, eseguirai le seguenti attività:

- Creazione di una funzione Lambda di esempio
- Utilizzo della console Lambda per visualizzare il modello AWS SAM della funzione
- Esportazione della configurazione della funzione in Strumento AWS per la creazione di applicazioni e progettazione di una semplice applicazione serverless basata sulla configurazione della funzione
- Salvataggio di un modello AWS SAM aggiornato che può essere utilizzato come base per implementare un'applicazione serverless

Nella sezione the section called “Passaggi successivi”, troverai le risorse che puoi utilizzare per ottenere maggiori informazioni su AWS SAM e Strumento per la creazione di applicazioni. Queste risorse includono collegamenti a tutorial più avanzati che insegnano come implementare un'applicazione serverless utilizzando AWS SAM.

Prerequisiti

In questo tutorial, utilizzi la funzionalità di sincronizzazione locale di Strumento per la creazione di applicazioni per salvare i file di modello e codice sul 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.

Visualizzazione del modello AWS SAM per la funzione

Prima di esportare la configurazione della funzione in Strumento per la creazione di applicazioni, utilizza la console Lambda per visualizzare la configurazione corrente della funzione come modello AWS SAM. Seguendo i passaggi di questa sezione, imparerai a conoscere la struttura di un modello AWS SAM e a definire risorse come le funzioni Lambda per iniziare a specificare un'applicazione serverless.

Visualizzazione del modello AWS SAM per la 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 funzione, vedrai un modello AWS SAM relativo alla 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:*
```

Esaminiamo il modello YAML per la funzione in uso e apprendiamo alcuni concetti chiave.

Il modello inizia con la dichiarazione `Transform: AWS::Serverless-2016-10-31`. Questa dichiarazione è obbligatoria perché, dietro le quinte, i modelli AWS SAM vengono implementati tramite AWS CloudFormation. L'utilizzo dell'istruzione `Transform` identifica il modello come file di modello AWS SAM.

Dopo la dichiarazione `Transform`, arriva la sezione `Resources`. Qui definisci le risorse AWS che desideri implementare con il modello AWS SAM. I modelli AWS SAM possono contenere una combinazione di risorse AWS SAM e risorse AWS CloudFormation. Questo perché, durante l'implementazione, i modelli AWS SAM si espandono in modelli AWS CloudFormation, quindi a un modello AWS SAM è possibile aggiungere qualsiasi sintassi AWS CloudFormation valida.

Al momento, nella sezione `Resources` del modello è presente soltanto una risorsa definita, la funzione Lambda `LambdaIaCDemo`. Per aggiungere una funzione Lambda a un modello AWS SAM, utilizzi il tipo di risorsa `AWS::Serverless::Function`. La risorsa `Properties` di una funzione Lambda definisce il runtime della funzione, il gestore delle funzioni e altre opzioni di configurazione. Qui viene definito anche il percorso del codice sorgente della funzione che AWS SAM deve utilizzare per implementare la funzione. Per ulteriori informazioni sulle risorse delle funzioni Lambda inAWS SAM, consulta AWS::Serverless::Functionla Guida per gli AWS SAMsviluppatori.

Oltre alle proprietà e alle configurazioni della funzione, il modello specifica anche una policy AWS Identity and Access Management (IAM) per la funzione. Questa politica 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 ulteriori informazioni su

come specificare una policy IAM per una funzione in un AWS SAM modello, consulta la *policies* proprietà nella AWS::Serverless::Functionpagina della AWS SAM Developer Guide.

Per ulteriori informazioni sulla struttura dei modelli AWS SAM, consulta la pagina AWS SAM template anatomy.

Utilizzo di Strumento AWS per la creazione di applicazioni per progettare un'applicazione serverless

Per iniziare a creare una semplice applicazione serverless utilizzando il modello AWS SAM della funzione come punto di partenza, esporta la configurazione della funzione in Strumento per la creazione di applicazioni e attiva la modalità di sincronizzazione locale di Strumento per la creazione di applicazioni. La sincronizzazione locale salva automaticamente il codice della funzione e il modello AWS SAM sul computer di compilazione locale e mantiene sincronizzato il modello salvato man mano che si aggiungono altre risorse AWS a Strumento per la creazione di applicazioni.

Esportazione della funzione in Strumento per la creazione di applicazioni

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 Strumento per la creazione di applicazioni, 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 Strumento per la creazione di applicazioni.
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 Strumento per la creazione di applicazioni. Nel canvas vedrai la 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 nello Strumento per la creazione di applicazioni, 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 Strumento per la creazione di applicazioni 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 l'addebito di costi aggiuntivi sul tuo Account AWS, segui le istruzioni in Deleting a bucket dopo aver esportato la funzione in Strumento per la creazione di applicazioni. Per ulteriori informazioni sul bucket Amazon S3 creato da Lambda, consulta la pagina the section called “Application Composer”.

Progettazione di un'applicazione serverless in Strumento per la creazione di applicazioni

Dopo aver attivato la sincronizzazione locale, le modifiche apportate in Strumento per la creazione di applicazioni si rifletteranno nel modello AWS SAM salvato sul computer di compilazione locale. Ora puoi trascinare e rilasciare altre risorse AWS sul canvas di Strumento per la creazione di applicazioni per creare l'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. Selezionare 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, Strumento per la creazione di applicazioni 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. Selezionare 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 nuove risorse, diamo un'occhiata al modello AWS SAM aggiornato creato da Strumento per la creazione di applicazioni.

Visualizzazione del modello AWS SAM aggiornato

- Nel canvas di Strumento per la creazione di applicazioni, scegli Modello per passare dalla visualizzazione del canvas alla visualizzazione del modello.

Ora il modello AWS SAM dovrebbe 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 Strumento per la creazione di applicazioni, Strumento per la creazione di applicazioni 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 AWS 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 maggiori informazioni sulle diverse configurazioni Servizi AWS che puoi configurare per attivare una funzione Lambda e sulle proprietà che puoi impostare, consulta la AWS SAMDeveloper EventSourceGuide.

- 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 Strumento per la creazione di applicazioni, puoi impostare le chiavi della tabella scegliendo Dettagli nella scheda della tabella DynamoDB e modificando i valori delle chiavi. Strumento per la creazione di applicazioni imposta i valori predefiniti anche per una serie di altre proprietà, tra cui `BillingMode` e `StreamViewType`.

  Per ulteriori informazioni su queste proprietà e sulle altre proprietà che puoi aggiungere al tuo modello AWS SAM, consulta la sezione AWS::DynamoDB::Table nella Guida per l'utente di AWS 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 modello AWS SAM finale completo avrà un aspetto 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
```

Implementazione dell'applicazione serverless utilizzando AWS SAM (facoltativo)

Se desideri utilizzare AWS SAM per implementare un'applicazione serverless mediante il modello appena creato in Strumento per la creazione di applicazioni, devi prima installare la CLI di AWS SAM. Per farlo, segui le istruzioni riportate nella pagina Installing the AWS SAM CLI.

Inoltre, prima di implementare l'applicazione devi aggiornare il codice della funzione che Strumento per la creazione di applicazioni ha salvato insieme al modello. Al momento, il file `lambda_function.py` salvato da Strumento per la creazione di applicazioni 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 Strumento per la creazione di applicazioni ha salvato sul computer di compilazione locale. Hai specificato la directory in cui Strumento per la creazione di applicazioni 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 Strumento per la creazione di applicazioni. 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 implementare l'applicazione utilizzando la CLI di AWS SAM, effettua le seguenti operazioni. 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 Strumento per la creazione di applicazioni 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 implementare l'applicazione e creare le risorse Lambda, Amazon SQS e DynamoDB specificate nel modello AWS SAM, esegui il comando seguente.

   ```
   sam deploy --guided
   ```

   L'utilizzo del flag `--guided` significa che AWS SAM mostrerà le istruzioni per guidarti attraverso il processo di implementazione. Per questa implementazione, accetta le opzioni predefinite premendo Invio.

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

- Uno stack AWS CloudFormation denominato `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.

Per ulteriori informazioni sull'utilizzo di AWS SAM per l'implementazione di applicazioni serverless, consulta le risorse nella sezione the section called “Passaggi successivi”.

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`.

Passaggi successivi

Per ulteriori informazioni sull'utilizzo di Strumento per la creazione di applicazioni con AWS SAM e AWS CloudFormation, consulta la pagina Using Application Composer with AWS CloudFormation and AWS SAM.

Per un tutorial guidato che utilizza AWS SAM per l'implementazione di un'applicazione serverless progettata in Strumento per la creazione di applicazioni, ti consigliamo anche di seguire le istruzioni di Strumento AWS per la creazione di applicazioni tutorial nel AWS Serverless Patterns Workshop.

AWS SAM fornisce un'interfaccia a riga di comando (CLI) che è possibile utilizzare con i modelli AWS SAM e integrazioni di terze parti supportate per creare ed eseguire applicazioni serverless. Con la CLI di AWS SAM, puoi creare e implementare la tua applicazione, eseguire test e debug locali, configurare pipeline CI/CD e altro ancora. Per ulteriori informazioni sull'utilizzo della CLI di AWS SAM, consulta la pagina Getting started with AWS SAM nella Guida per gli sviluppatori di AWS Serverless Application Model.

Per informazioni su come implementare un'applicazione serverless con un modello AWS SAM tramite la console AWS CloudFormation, consulta la pagina Using the AWS CloudFormation console nella Guida per l'utente di AWS CloudFormation.

Regioni che supportano l'integrazione Lambda con Strumento per la creazione di applicazioni

L'integrazione Lambda con Strumento per la creazione di applicazioni è supportata nelle seguenti Regioni AWS:

- Stati Uniti orientali (Virginia settentrionale)
- Stati Uniti orientali (Ohio)
- Stati Uniti occidentali (California settentrionale)
- Stati Uniti occidentali (Oregon)
- Africa (Città del Capo)
- Asia Pacifico (Hong Kong)
- Asia Pacific (Hyderabad)
- Asia Pacifico (Giacarta)
- Asia Pacifico (Melbourne)
- Asia Pacifico (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seul)
- Asia Pacifico (Singapore)
- Asia Pacifico (Sydney)

- Asia Pacifico (Tokyo)
- Canada (Centrale)
- Europa (Francoforte)
- Europa (Zurigo)
- Europa (Irlanda)
- Europe (London)
- Europa (Stoccolma)
- Medio Oriente (Emirati Arabi Uniti)

Rete privata con VPC

Amazon Virtual Private Cloud (Amazon VPC) è una rete virtuale nel AWS cloud, dedicata al tuo AWS account. Puoi utilizzare Amazon VPC per creare una rete privata per le risorse come database, istanze di cache o servizi interni. Per ulteriori informazioni su Amazon VPC, consulta Cos'è Amazon VPC?

Una funzione Lambda viene sempre eseguita all'interno di un VPC di proprietà del servizio Lambda. Lambda applica le regole di accesso alla rete e di sicurezza a questo VPC e mantiene e monitora il VPC automaticamente. Se la funzione Lambda ha bisogno di accedere alle risorse nel VPC del tuo account, configura la funzione per accedere al VPC. Lambda fornisce risorse gestite denominate Hyperplane ENI, utilizzate dalla funzione Lambda per connettersi dal VPC Lambda a un ENI (Elastic Network Interface) nel tuo account VPC.

Non ci sono costi aggiuntivi per l'utilizzo di un VPC o di una Hyperplane ENI. Sono previsti addebiti per alcuni componenti VPC, ad esempio i gateway NAT. Per ulteriori informazioni, consulta la pagina dei Prezzi di Amazon VPC.

Argomenti

Elementi di rete VPC

Le reti Amazon VPC includono i seguenti elementi di rete:

- Interfaccia di rete elastica: una interfaccia di rete elastica è un componente di rete logico in un VPC che rappresenta una scheda di rete virtuale.

- Sottorete: un intervallo di indirizzi IP nel VPC. Puoi aggiungere AWS risorse a una sottorete specificata. Utilizzare una sottorete pubblica per le risorse che devono essere connesse a Internet e una sottorete privata per le risorse da non connettere a Internet.
- Gruppo di sicurezza: utilizza i gruppi di sicurezza per controllare l'accesso alle AWS risorse in ogni sottorete.
- Lista di controllo degli accessi (ACL): utilizza un'ACL di rete per fornire ulteriore protezione in una sottorete. L'ACL della sottorete predefinita permette tutto il traffico in entrata e in uscita.
- Tabella delle rotte: contiene una serie di percorsi che AWS vengono utilizzati per indirizzare il traffico di rete per il tuo VPC. Puoi associare esplicitamente una sottorete a una particolare tabella di instradamento. Per impostazione predefinita, la sottorete è associata alla tabella di instradamento principale.
- Instradamento: ogni instradamento in una tabella di instradamento specifica un intervallo di indirizzi IP e la destinazione in cui Lambda invia il traffico per questo intervallo. L'instradamento specifica inoltre una destinazione, ovvero il gateway, l'interfaccia di rete o la connessione tramite cui inviare il traffico.
- Gateway NAT: un servizio NAT (AWS Network Address Translation) che controlla l'accesso da una sottorete privata VPC privata a Internet.
- Endpoint VPC: puoi utilizzare un endpoint Amazon VPC per creare connettività privata ai servizi ospitati in AWS, senza richiedere l'accesso su Internet o tramite un dispositivo NAT, una connessione VPN o una connessione. AWS Direct Connect Per ulteriori informazioni, consulta AWS PrivateLink ed endpoint VPC.

> **Tip**
>
> Per configurare la funzione Lambda per accedere a un VPC e a una sottorete, è possibile utilizzare l'API o la console Lambda.
> Fai riferimento alla `VpcConfig` sezione in CreateFunctionper configurare la tua funzione.
> Vedi Collegamento di funzioni Lambda a un Amazon VPC nel tuo Account AWS per i passaggi dettagliati.

Per ulteriori informazioni sulle definizioni di rete di Amazon VPC, consulta Come funziona Amazon VPC nella Guida per gli sviluppatori di Amazon VPC e Domande frequenti su Amazon VPC.

Collegamento delle funzioni Lambda al VPC

Una funzione Lambda viene sempre eseguita all'interno di un VPC di proprietà del servizio Lambda. Per impostazione predefinita, una funzione Lambda non è connessa ai VPC del tuo account. Quando connetti una funzione a un VPC nel tuo account, questa non ha accesso a Internet a meno che non venga fornito dal VPC.

Lambda accede alle risorse del VPC utilizzando una Hyperplane ENI. Le Hyperplane ENI forniscono funzionalità NAT dal VPC Lambda al VPC del tuo account utilizzando il NAT (V2N) da VPC a VPC. V2N fornisce la connettività dal VPC Lambda al VPC del tuo account, ma non nella direzione opposta.

Quando crei una funzione Lambda (o aggiorni le relative impostazioni del VPC), Lambda assegna una Hyperplane ENI per ogni sottorete nella configurazione VPC della funzione. Più funzioni Lambda possono condividere un'interfaccia di rete se condividono già la stessa sottorete e gruppo di sicurezza.

Per connetterti a un altro AWS servizio, puoi utilizzare gli endpoint VPC per comunicazioni private tra il tuo VPC e i servizi supportati. AWS Un approccio alternativo consiste nell'utilizzare un gateway NAT per indirizzare il traffico in uscita verso un altro servizio. AWS

Per concedere alla funzione l'accesso a Internet, instradare il traffico in uscita a un gateway NAT in una sottorete pubblica. Il gateway NAT dispone di un indirizzo IP pubblico e può connettersi a Internet tramite il gateway Internet del VPC. Per informazioni, consulta Abilita l'accesso a Internet per le funzioni Lambda connesse a VPC.

Sottoreti condivise

La condivisione di VPC consente a più AWS account di creare le proprie risorse applicative, come le istanze Amazon EC2 e le funzioni Lambda, in cloud privati virtuali (VPC) condivisi e gestiti centralmente. In questo modello, l'account proprietario del VPC (proprietario) condivide una o più sottoreti con altri account (partecipanti) che appartengono alla stessa organizzazione. AWS

Per accedere alle risorse private, connetti la tua funzione a una sottorete privata condivisa nel VPC. Il proprietario della sottorete deve condividerla con te prima che tu possa connettervi una funzione. Il proprietario della sottorete può anche annullarne la condivisione in un secondo momento, rimuovendo così la connettività. Per dettagli su come condividere, annullare la condivisione e gestire le risorse VPC nelle sottoreti condivise, consulta la pagina Come condividere un VPC con altri account nella Guida di Amazon VPC.

Hyperplane ENI di Lambda

Una Hyperplane ENI è una risorsa di rete gestita creata e gestita dal servizio Lambda. Più ambienti di esecuzione nel VPC Lambda possono utilizzare una Hyperplane ENI per accedere in modo sicuro alle risorse all'interno dei VPC nel tuo account. Le Hyperplane ENI forniscono funzionalità NAT dal VPC Lambda al VPC del tuo account.

Per ogni sottorete, Lambda crea un'interfaccia di rete per ogni set unico di gruppi di sicurezza. Le funzioni dell'account che condividono la stessa combinazione di sottorete e gruppo di sicurezza utilizzano le stesse interfacce di rete. Le connessioni effettuate tramite il livello Hyperplane vengono monitorate automaticamente, anche se la configurazione del gruppo di sicurezza non lo richiede. I pacchetti in entrata dal VPC che non corrispondono alle connessioni stabilite vengono eliminati al livello Hyperplane. Per ulteriori informazioni, consulta Tracciamento delle connessioni dei gruppi di sicurezza nella Guida per l'utente di Amazon EC2.

Poiché le funzioni del tuo account condividono le risorse dell'ENI, il ciclo di dell'vita ENI è più complesso di altre risorse Lambda. Nelle sezioni seguenti viene descritto il ciclo di vita dell'ENI.

Ciclo di vita dell'ENI

- Creazione di ENI
- Gestione delle ENI
- Eliminazione di ENI

Creazione di ENI

Lambda può creare risorse Hyperplane ENI per una nuova funzione abilitata al VPC o per una modifica della configurazione VPC a una funzione esistente. La funzione rimane in sospeso mentre Lambda crea le risorse richieste. Quando l'Hyperplane ENI è pronta, la funzione passa allo stato attivo e l'ENI diventa disponibile per l'uso. Lambda può richiedere alcuni minuti per creare l'Hyperplane ENI.

Per una nuova funzione abilitata per il VPC, tutte le invocazioni o altre operazioni API che agiscono sulla funzione non funzionano fino a quando lo stato della funzione non passa ad attivo.

Per una modifica della configurazione VPC a una funzione esistente, qualsiasi invocazione di funzione continua a utilizzare l'Hyperplane ENI associata alla vecchia configurazione della sottorete e del gruppo di sicurezza fino a quando lo stato della funzione non passa ad attivo.

Se una funzione Lambda rimane inattiva per 30 giorni, Lambda recupera gli ENI Hyperplane non utilizzati e imposta lo stato della funzione su inattivo. La successiva invocazione fa sì che Lambda riattivi la funzione di inattività. L'invocazione ha esito negativo e la funzione entra nello stato in sospeso fino a quando Lambda non completa la creazione o l'allocazione di una Hyperplane ENI.

Per ulteriori informazioni sugli stati delle funzioni, consulta Stati funzione Lambda.

Gestione delle ENI

Lambda utilizza le autorizzazioni nel ruolo di esecuzione della funzione per creare e gestire le interfacce di rete. Lambda crea una Hyperplane ENI quando viene definita una combinazione univoca di sottorete e gruppo di sicurezza per una funzione abilitata al VPC in un account. Lambda riutilizza l'Hyperplane ENI per altre funzioni abilitate al VPC nel tuo account che utilizzano la stessa combinazione di sottorete e gruppo di sicurezza.

Non vi è una quota sul numero di funzioni Lambda che possono utilizzare la stessa Hyperplane ENI. Tuttavia, ogni Hyperplane ENI supporta fino a 65.000 connessioni/porte. Se il numero di connessioni supera 65.000, Lambda crea una nuova Hyperplane ENI per fornire connessioni aggiuntive.

Se aggiorni la configurazione della funzione per accedere a un VPC diverso, Lambda interrompe la connettività all'Hyperplane ENI nel VPC precedente. Il processo per aggiornare la connettività a un nuovo VPC può richiedere alcuni minuti. Durante questo periodo, le invocazioni della funzione continuano a utilizzare il VPC precedente. Al termine dell'aggiornamento, le nuove invocazioni iniziano a utilizzare l'Hyperplane ENI nel nuovo VPC. A questo punto, la funzione Lambda non è più collegata al VPC precedente.

Eliminazione di ENI

Quando aggiorni una funzione per rimuovere la configurazione VPC, Lambda richiede fino a 20 minuti per eliminare l'Hyperplane ENI collegata. 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.

Lambda non elimina le interfacce di rete utilizzate dalle funzioni o dalle versioni delle funzioni nel tuo account. Puoi utilizzare il Finder ENI Lambda per identificare le funzioni o le versioni delle funzioni

che utilizzano una Hyperplane ENI. Per tutte le funzioni o le versioni di funzioni che non sono più necessarie, puoi rimuovere la configurazione VPC in modo che Lambda elimini l'Hyperplane ENI.

Connessioni

Lambda supporta due tipi di connessioni: TCP (Transmission Control Protocol) e UDP (User Datagram Protocol).

Quando crei un VPC, Lambda crea automaticamente un set di opzioni DHCP e le associa al VPC. Puoi configurare le tue opzioni DHCP impostate per il VPC. Per ulteriori informazioni, consulta Opzioni DHCP di Amazon VPC.

Amazon fornisce un server DNS (il risolutore Amazon Route 53) per il tuo VPC. Per ulteriori informazioni, consulta Supporto DNS per il tuo VPC.

Supporto IPv6

Lambda supporta connessioni in entrata agli endpoint dual-stack pubblici di Lambda e connessioni in uscita a sottoreti VPC dual-stack su IPv6.

In entrata

Per richiamare la funzione su IPv6, usa gli endpoint dual-stack pubblici di Lambda. Endpoint dual-stack che supportano sia IPv4 sia IPv6. Gli endpoint dual-stack Lambda utilizzano la seguente sintassi:

```
protocol://lambda.us-east-1.api.aws
```

È inoltre possibile utilizzare URL della funzione Lambda 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
```

In uscita

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 connessioni IPv6 in uscita per sottoreti solo IPv6 in un VPC, connessioni IPv6 in uscita per funzioni che non sono connesse a un VPC o connessioni IPv6 in entrata che utilizzano endpoint VPC (AWS PrivateLink).

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()
```

Sicurezza

AWS fornisce gruppi di sicurezza e ACL di rete 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 ulteriori informazioni, consulta Riservatezza del traffico Internet in Amazon VPC. Ogni sottorete creata viene automaticamente associata alla lista di controllo accessi di rete predefinita del VPC. Puoi modificare l'associazione E modificare il contenuto delle lista di controllo accessi di rete.

Per le best practice generali relative alla sicurezza, consulta Best practice relative alla sicurezza del VPC. Per ulteriori informazioni su come utilizzare IAM per gestire l'accesso all'API e alle risorse Lambda, consulta Autorizzazioni AWS Lambda.

È possibile utilizzare chiavi di condizione Lambda specifiche per le impostazioni VPC per fornire controlli di autorizzazione aggiuntivi per le funzioni Lambda. Per ulteriori informazioni sulle chiavi di condizione del VPC, consulta Utilizzo delle chiavi di condizione IAM per le impostazioni del VPC.

> **Note**
>
> Le funzioni Lambda possono essere richiamate dalla rete Internet pubblica o dagli endpoint AWS PrivateLink. Puoi accedere all'URL della funzione solo tramite Internet pubblico. Sebbene le funzioni Lambda lo supportino AWS PrivateLink, gli URL delle funzioni no.

Osservabilità

I flussi di log VPC possono essere utilizzati per acquisire informazioni sul traffico IP verso e dalle interfacce di rete nel VPC. Puoi pubblicare i dati dei log di flusso su Amazon CloudWatch Logs o Amazon S3. Dopo aver creato un log di flusso, puoi recuperare e visualizzare i relativi dati nella destinazione scelta.

Nota: quando si collega una funzione a un VPC, i messaggi di CloudWatch registro non utilizzano le route VPC. Lambda li invia utilizzando l'instradamento comune per i log.

Configurazione dell'architettura del set di istruzioni per una 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 Graviton2 AWS .
- 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 la sezione Prezzi di AWS Lambda. Nella tabella dei prezzi della memoria, scegli la scheda Arm Price, quindi apri l'elenco a discesa Region per vedere quali Regioni AWS supportano arm64 con Lambda.
> Per un esempio di come creare una funzione con l'architettura arm64, vedi AWS Lambda Functions Powered by Graviton2 Processor. AWS

Argomenti

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 Armv8.2 (incluse le estensioni CRC e crittografiche) oltre a diverse altre estensioni architettoniche.

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. Inoltre, Graviton2 offre prestazioni di crittografia migliorate e supporta set di istruzioni che migliorano la latenza dell'inferenza di machine learning basata sulla CPU.

Per ulteriori informazioni su Graviton2, vedere Graviton Processor. AWSAWS

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:

- Attualmente la funzione utilizza un runtime Lambda Amazon Linux 2.
- 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 delle 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 AWS SDK, () o. AWS Command Line Interface AWS CLI AWS

CloudFormation Completa questa procedura per modificare l'architettura del set di istruzioni per una funzione Lambda esistente dalla console.

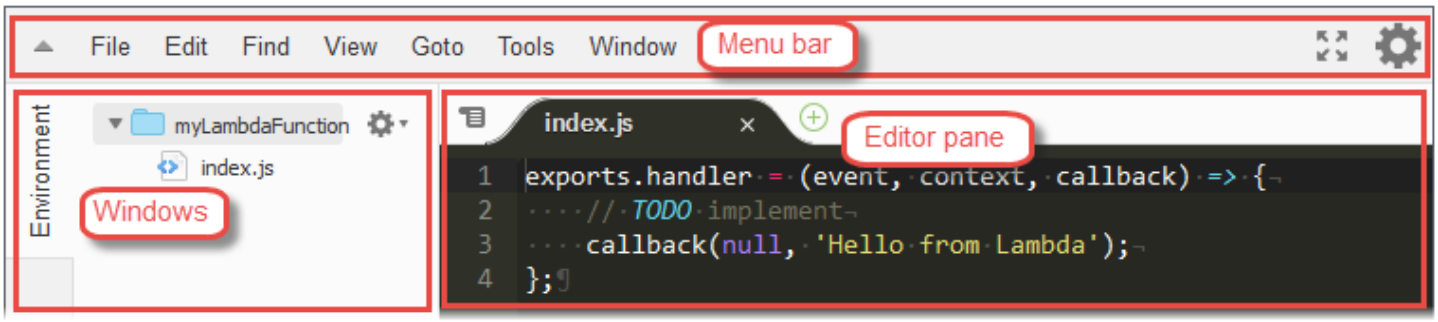
1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione per la quale desideri configurare l'architettura del set di istruzioni.
3. Nella scheda Codice principale, per la sezione Impostazioni di runtime, scegli Modifica.
4. In Architettura, scegli l'architettura del set di istruzioni da utilizzare per la funzione.
5. Selezionare Salva.

> **Note**
>
> Tutti i runtime Amazon Linux 2 supportano entrambe le architetture CPU x86_64 e ARM. I runtime che non utilizzano il sistema operativo Amazon Linux 2, come Go 1.x, non supportano l'architettura arm64. Per utilizzare l'architettura arm64 con Go 1.x, è possibile eseguire la funzione con un runtime provided.al2. Per ulteriori informazioni, consulta le istruzioni per l'implementazione dei pacchetti .zip e delle immagini di container.

Modifica il codice utilizzando l'editor della console Lambda

Puoi utilizzare l'editor di codice nella console Lambda per scrivere, testare e visualizzare i risultati dell'esecuzione del codice della funzione Lambda. L'editor di codice supporta linguaggi che non richiedono la compilazione, come Node.js o Python. L'editor di codice supporta solo i pacchetti di implementazione dell'archivio .zip di dimensioni inferiori a 3 MB.

L'editor di codice è costituito dalla barra dei menu, dalle finestre e dal riquadro dell'editor.



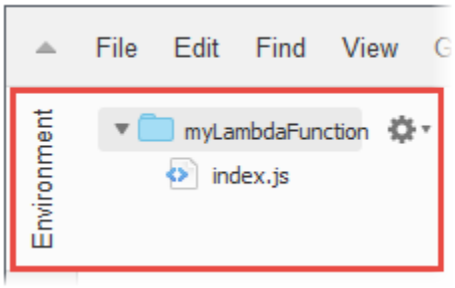
Per un elenco delle operazioni dei comandi, consulta la documentazione di riferimento dei comandi di menu nella Guida per l'utente di AWS Cloud9 . Alcuni comandi elencati in tale riferimento non sono disponibili nell'editor di codice.

Argomenti

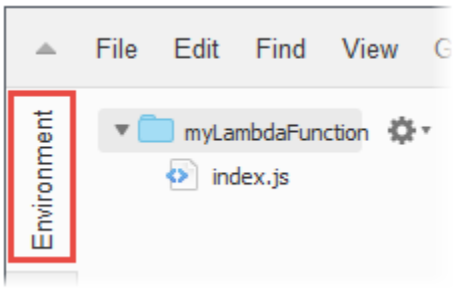
- Utilizzo di file e cartelle
- Utilizzo del codice
- Utilizzo della modalità a schermo intero
- Utilizzo delle preferenze

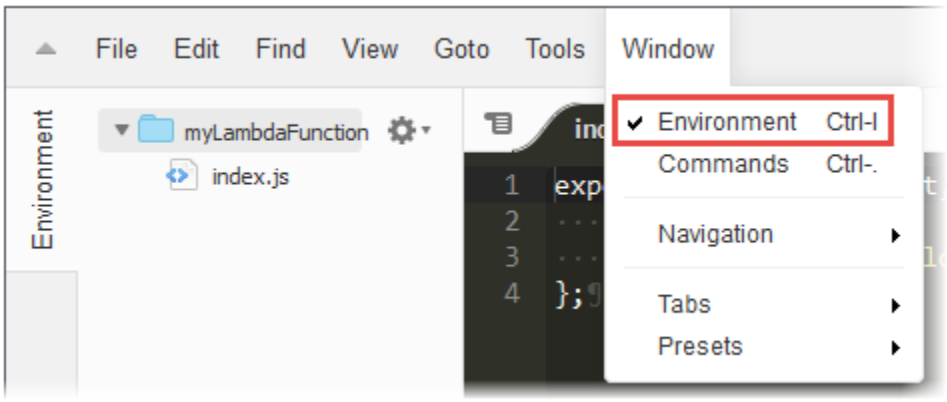
Utilizzo di file e cartelle

La finestra Environment (Ambiente) nell'editor di codice consente di creare, aprire e gestire file per la funzione.



Per mostrare o nascondere la finestra Environment (Ambiente), scegliere il pulsante Environment (Ambiente). Se il pulsante Environment (Ambiente) non è visibile, scegliere Window, Environment (Finestra, Ambiente) nella barra dei menu.



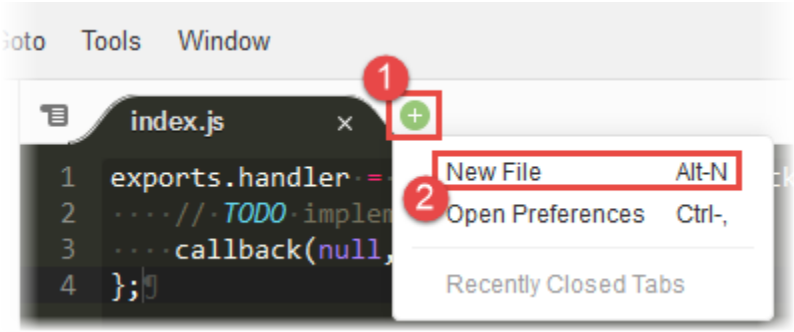


Per aprire un singolo file e visualizzarne il contenuto nel riquadro dell'editor, fare doppio clic sul file nella finestra Environment (Ambiente).

Per aprire più file e visualizzarne il contenuto nel riquadro dell'editor, scegliere i file nella finestra Environment (Ambiente). Fare clic con il pulsante destro del mouse sulla selezione, quindi scegliere Open (Apri).

Per creare un nuovo file, effettuare una delle operazioni indicate di seguito:

- Nella finestra Environment (Ambiente) fare clic con il pulsante destro del mouse sulla cartella in cui inserire il nuovo file, quindi scegliere New File (Nuovo file). Inserire il nome e l'estensione del file, quindi premere Invio.
- Scegliere File, New File (Nuovo file) nella barra dei menu. Per salvare il file, scegliere File, Save (Salva) o File, Save As (Salva con nome) nella barra dei menu, quindi utilizzare la finestra di dialogo Save As (Salva con nome) in cui viene chiesto di assegnare un nome al file e scegliere il percorso in cui salvarlo.
- Nella barra dei pulsanti della scheda nel riquadro dell'editor scegliere il pulsante +, quindi scegliere New File (Nuovo file). Per salvare il file, scegliere File, Save (Salva) o File, Save As (Salva con nome) nella barra dei menu, quindi utilizzare la finestra di dialogo Save As (Salva con nome) in cui viene chiesto di assegnare un nome al file e scegliere il percorso in cui salvarlo.



Per creare una nuova cartella, fare clic con il pulsante destro del mouse nella finestra Environment (Ambiente) in cui posizionare la nuova cartella, quindi scegliere New Folder (Nuova cartella). Inserire il nome della cartella, quindi premere Invio.

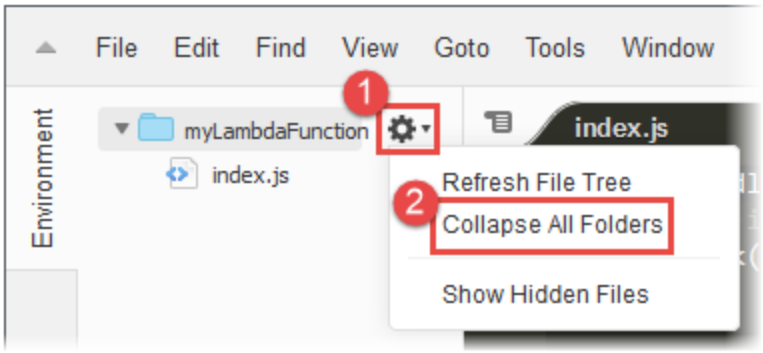
Per salvare un file, scegliere File, Save (Salva) nella barra dei menu mantenendo il file aperto e il relativo contenuto visibile nel riquadro dell'editor.

Per rinominare un file o una cartella, fare clic con il pulsante destro del mouse sul file o sulla cartella nella finestra Environment (Ambiente). Inserire il nome sostitutivo, quindi premere Invio.

Per eliminare file o cartelle, scegliere i file o le cartelle nella finestra Environment (Ambiente). Fare clic con il pulsante destro del mouse sulla selezione, quindi scegliere Delete (Elimina). Per confermare l'eliminazione, scegliere Yes (Sì), se la selezione è unica, o Yes to All (Sì per tutti).

Per tagliare, incollare o duplicare file o cartelle, scegliere i file o le cartelle nella finestra Environment (Ambiente). Fare clic con il pulsante destro del mouse sulla selezione, quindi scegliere Cut (Taglia), Copy (Copia), Paste (Incolla) o Duplicate (Duplica), rispettivamente.

Per comprimere le cartelle, scegliere l'icona a forma di ingranaggio nella finestra Environment (Ambiente), quindi scegliere Collapse All Folders (Comprimi tutte le cartelle).



Per mostrare file nascosti o nasconderli nuovamente, scegliere l'icona a forma di ingranaggio nella finestra Environment (Ambiente), quindi scegliere Show Hidden Files (Mostra file nascosti).

Per visualizzare le variabili di ambiente configurate per la funzione, procedi come segue:

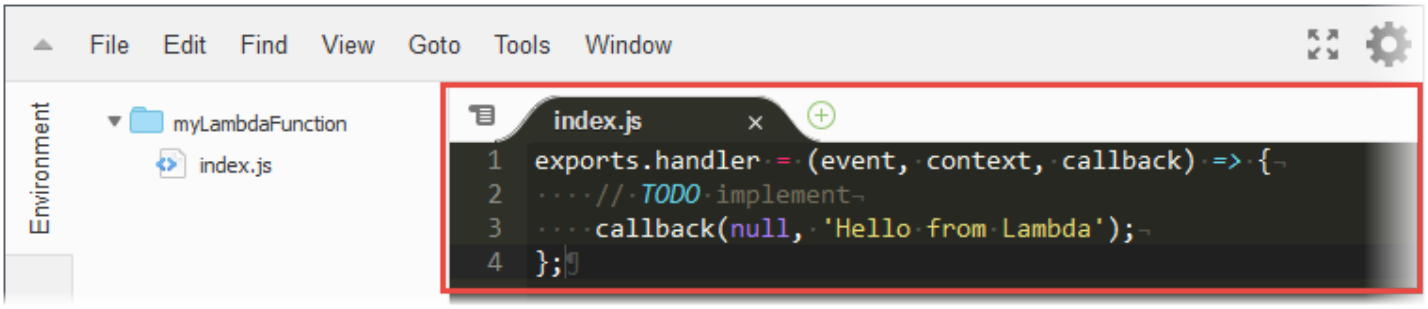
1. Scegli la scheda Codice.
2. Scegli la scheda Variabili di ambiente.
3. Scegli Strumenti, quindi Mostra variabili di ambiente.

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.

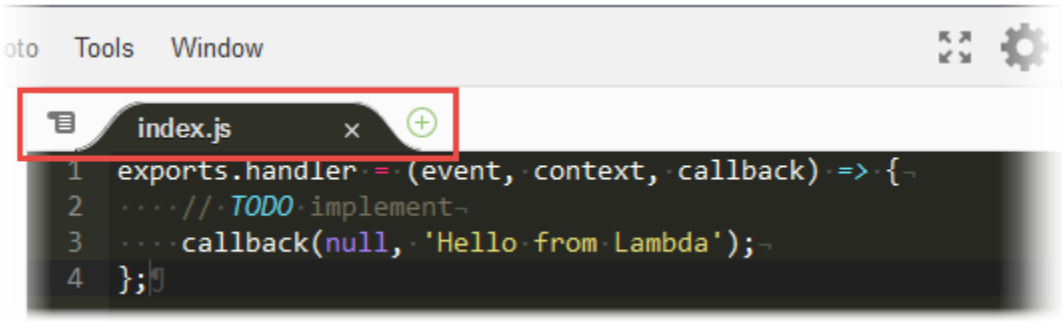
Utilizzo del codice

Per visualizzare e scrivere codice, utilizzare il riquadro dell'editor nell'editor di codice.



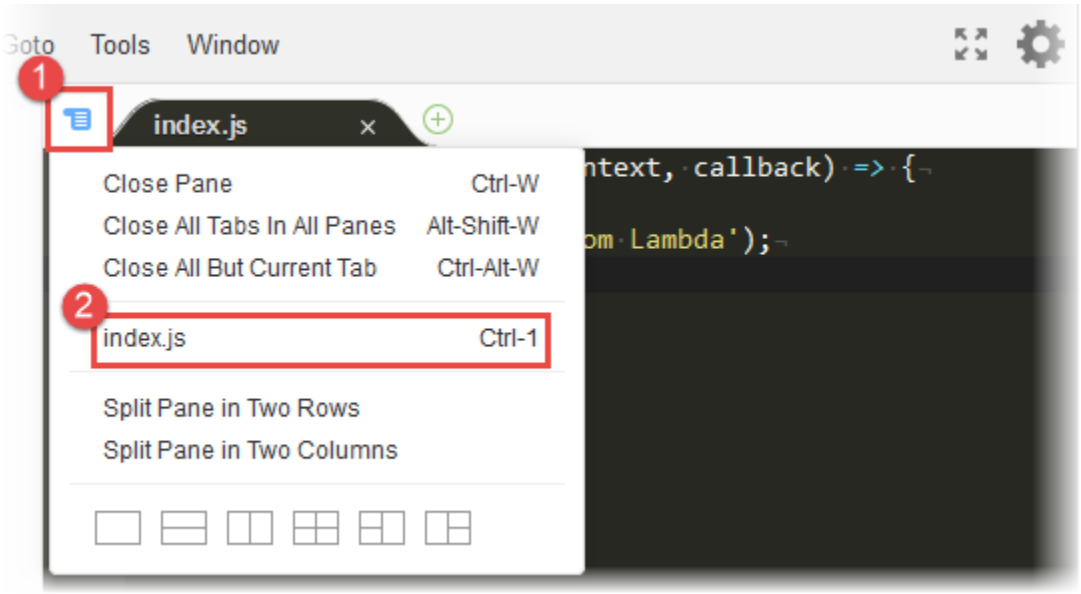
Utilizzo dei pulsanti della scheda

Per selezionare, visualizzare e creare file, utilizzare la barra dei pulsanti della scheda.



Per visualizzare il contenuto di un file aperto, effettuare una delle operazioni indicate di seguito:

- Scegliere la scheda del file.
- Nella barra dei pulsanti della scheda scegliere il pulsante relativo al menu a discesa, quindi scegliere il nome del file.



Per chiudere un file aperto, effettuare una delle operazioni indicate di seguito:

- Scegliere l'icona X nella scheda del file.
- Scegliere la scheda del file. Nella barra dei pulsanti della scheda scegliere il pulsante relativo al menu a discesa, quindi scegliere Close Pane (Chiudi riquadro).

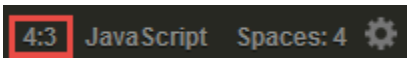
Per chiudere più file aperti, scegliere il menu a discesa nella barra dei pulsanti della scheda, quindi scegliere Close All Tabs in All Panes (Chiudi tutte le schede in tutti i riquadri) o Close All But Current Tab (Chiudi tutte le schede ad eccezione di quella corrente), in base alle esigenze.

Per creare un nuovo file, scegliere il pulsante + nella barra dei pulsanti della scheda, quindi scegliere New File (Nuovo file). Per salvare il file, scegliere File, Save (Salva) o File, Save As (Salva con nome) nella barra dei menu, quindi utilizzare la finestra di dialogo Save As (Salva con nome) in cui viene chiesto di assegnare un nome al file e scegliere il percorso in cui salvarlo.

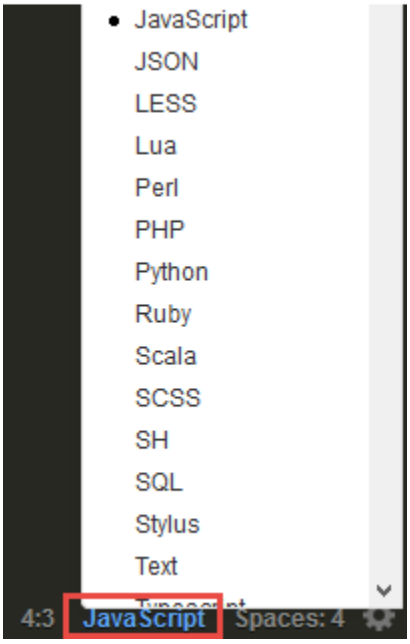
Utilizzo della barra di stato

La barra di stato consente di spostarsi rapidamente su una riga nel file attivo e di modificare le modalità di visualizzazione del codice.

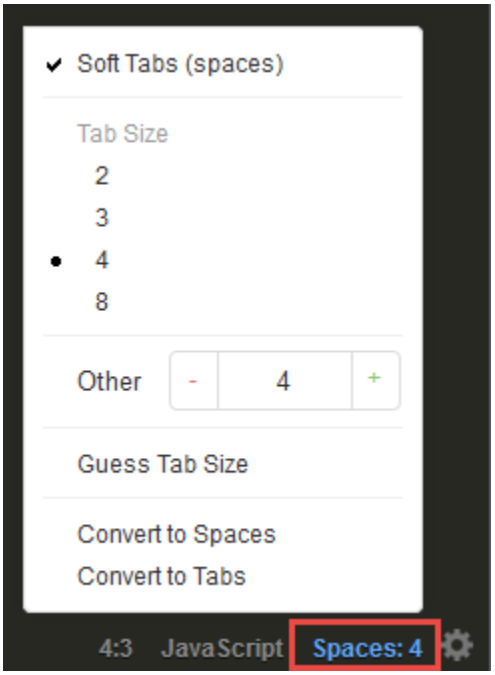
Per spostarsi rapidamente su una riga nel file attivo, scegliere il selettore di riga, digitare il numero di riga in cui spostarsi e quindi premere Invio.



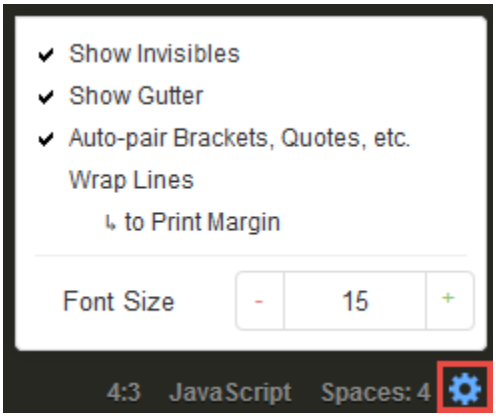
Per modificare lo schema di colori del codice nel file attivo, scegliere il selettore di schema, quindi il nuovo schema di colore del codice.



Per indicare se nel file attivo vengono utilizzati spazi o se eseguire la conversione in spazi o tabulazioni o per modificare le dimensioni di tabulazione, scegliere il selettore di spazi e tabulazioni, quindi scegliere le nuove impostazioni.



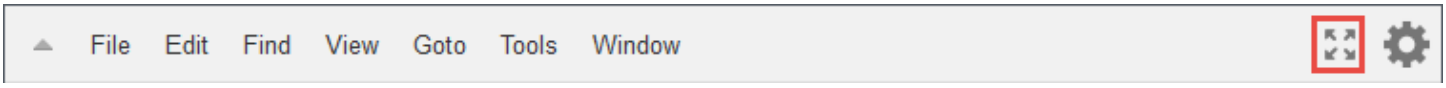
Per indicare se i caratteri invisibili o il margine, le parentesi o le virgolette ad accoppiamento automatico, le righe a capo o le dimensioni del carattere devono essere visualizzati o nascosti, scegliere l'icona a forma di ingranaggio, quindi scegliere le nuove impostazioni.



Utilizzo della modalità a schermo intero

L'editor di codice può essere espanso per avere a disposizione un'area di lavoro maggiore.

Per espandere l'editor di codice fino ai bordi della finestra del browser Web, scegliere il pulsante Toggle fullscreen (Attiva/disattiva schermo intero) nella barra dei menu.



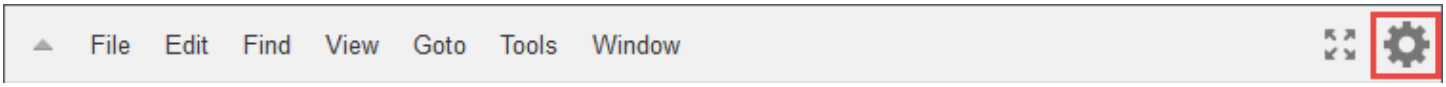
Per restringere l'editor di codice fino alle dimensioni originali, scegliere nuovamente il pulsante Toggle fullscreen (Attiva/disattiva schermo intero).

In modalità a schermo intero, vengono visualizzate altre opzioni sulla barra dei menu, Save (Salva) e Test. Save (Salva) consente di salvare il codice della funzione, mentre Test o Configure Events (Configura eventi) consente di creare o modificare gli eventi di test della funzione.

Utilizzo delle preferenze

È possibile modificare diverse impostazioni dell'editor di codice, ad esempio i suggerimenti e gli avvisi di codifica da visualizzare, il funzionamento per la riduzione e il completamento automatico del codice e altro ancora.

Per modificare le impostazioni dell'editor di codice, scegliere l'icona a forma di ingranaggio Preferences (Preferenze) nella barra dei menu.



Per un elenco degli effetti delle impostazioni, consultare i riferimenti seguenti nella Guida per l'utente di AWS Cloud9 .

- Impostazioni di progetto che è possibile modificare
- Modifiche alle impostazioni utente che è possibile eseguire

Alcune impostazioni elencate in tali riferimenti non sono disponibili nell'editor di codice.

Funzionalità Lambda aggiuntive

Lambda fornisce una console di gestione e un'API per la gestione e l'invocazione di funzioni. Fornisce runtime che supportano un set standard di caratteristiche, in modo da poter passare facilmente da linguaggi a framework a seconda delle esigenze. Oltre alle funzioni, puoi anche creare versioni, alias, livelli e runtime personalizzati.

Funzionalità avanzate

- Dimensionamento
- Controlli di simultaneità
- URL della funzione
- Invocazione asincrona
- Mappatura delle origini di eventi
- Destinazioni
- Piani delle funzioni
- Strumenti di test e distribuzione
- Modelli di applicazione

Dimensionamento

Lambda gestisce l'infrastruttura che esegue il codice, si dimensiona automaticamente in risposta alle richieste in entrata. Quando la funzione viene invocata più rapidamente di quanto una singola istanza della funzione possa elaborare eventi, Lambda si dimensiona eseguendo istanze aggiuntive. Quando il traffico diminuisce, le istanze inattive vengono congelate o interrotte. Paghi solo per il tempo impiegato dalla tua funzione a inizializzare o elaborare gli eventi.

Per ulteriori informazioni, consulta Comprendere il ridimensionamento delle funzioni Lambda.

Controlli di simultaneità

Utilizzare le impostazioni di simultaneità per garantire che le applicazioni di produzione abbiano disponibilità e reattività elevate.

Per impedire a una funzione di utilizzare troppa simultaneità e per riservare una parte della simultaneità disponibile dell'account per una funzione, utilizzare la simultaneità riservata. La

simultaneità riservata divide il pool di simultaneità disponibile in sottoinsiemi. Una funzione con simultaneità riservata utilizza solo la simultaneità dal suo sottoinsieme dedicato.

Per abilitare il dimensionamento delle funzioni senza fluttuazioni della latenza, utilizzare la simultaneità fornita. Per le funzioni che richiedono molto tempo per l'inizializzazione o che necessitano di una latenza estremamente bassa per tutte le invocazioni, la simultaneità fornita consente di preinizializzare le istanze della funzione e mantenerle in esecuzione in ogni momento. Lambda si integra con Application Auto Scaling per supportare la scalabilità automatica per la concorrenza di provisioning in base all'utilizzo.

Per ulteriori informazioni, consulta Configurazione della concorrenza riservata per una funzione.

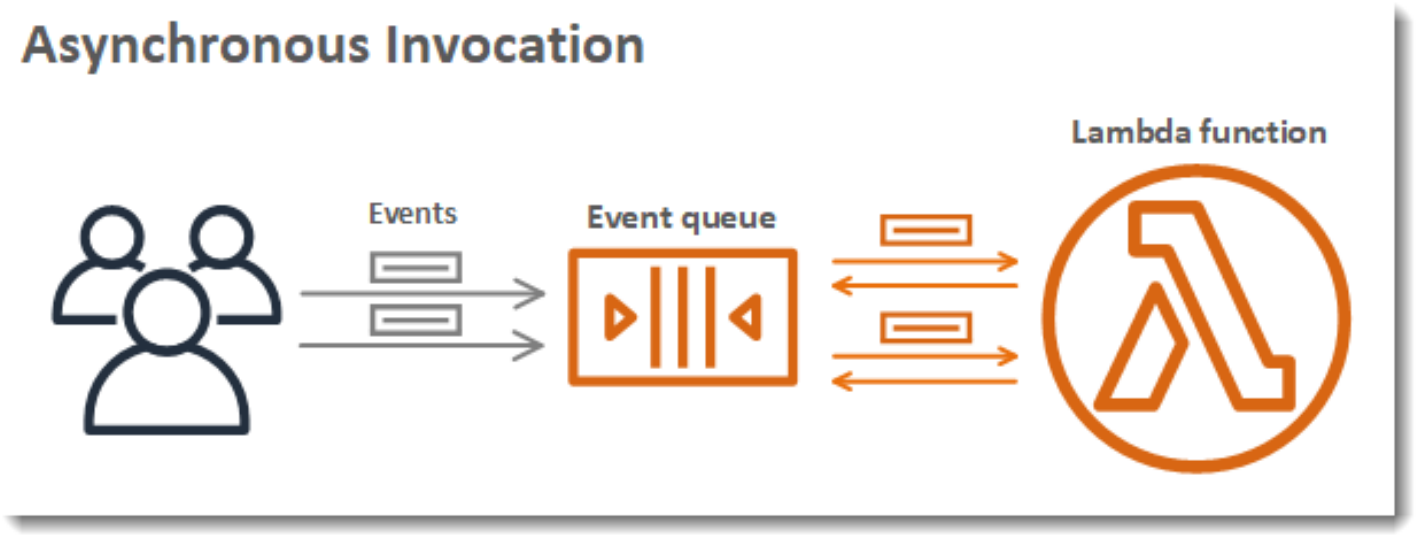
URL della funzione

Lambda offre il supporto degli endpoint HTTP(S) integrato tramite URL della funzione. Con gli URL della funzione, è possibile assegnare un endpoint HTTP dedicato alla funzione Lambda. Quando l'URL della funzione è configurato, è possibile utilizzarlo per invocare la funzione tramite un browser Web, curl, Postman o un client HTTP.

È possibile aggiungere un URL di funzione a una funzione esistente o creare una nuova funzione con un URL di funzione. Per ulteriori informazioni, consulta Richiamo di URL di funzioni Lambda.

Invocazione asincrona

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.

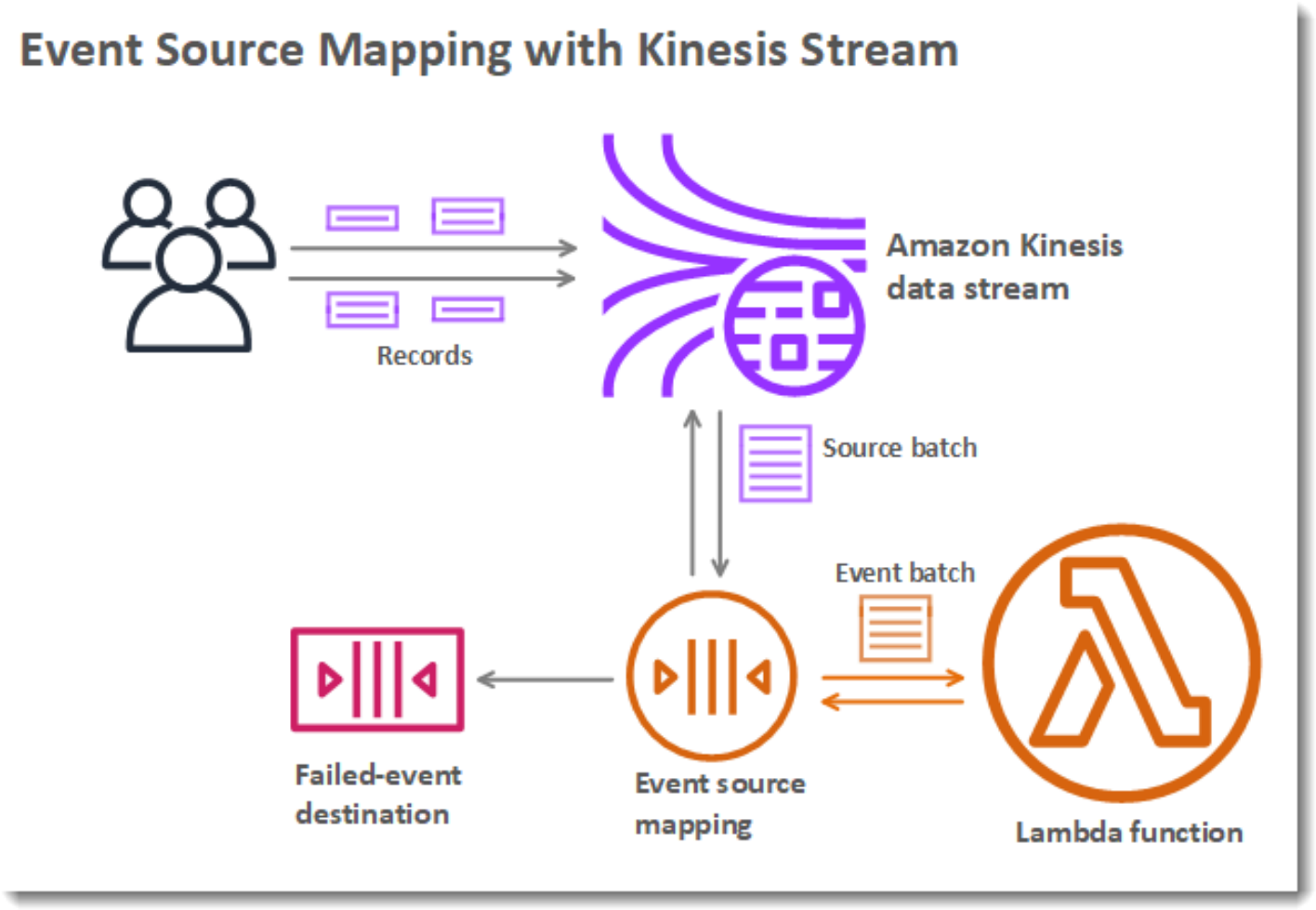


Per le invocazioni asincrone, Lambda gestisce i nuovi tentativi se la funzione restituisce un errore o è sottoposta a throttling. Per personalizzare questo comportamento, puoi configurare le impostazioni di gestione degli errori per una funzione, una versione o un alias. È inoltre possibile configurare Lambda per inviare gli eventi che non sono stati elaborati in una coda DLQ o per inviare un record di qualsiasi invocazione a una destinazione.

Per ulteriori informazioni, consulta Invocazione asincrona.

Mappatura delle origini di eventi

Per elaborare gli elementi da un flusso o una coda, è possibile creare una mappatura delle origini eventi. Una mappatura origine evento è una risorsa in Lambda che legge gli elementi da una coda Amazon Simple Queue Service (Amazon SQS), un flusso Amazon Kinesis o un flusso Amazon DynamoDB e li invia alla funzione in batch. Ogni evento che la funzione elabora può contenere centinaia o migliaia di elementi.

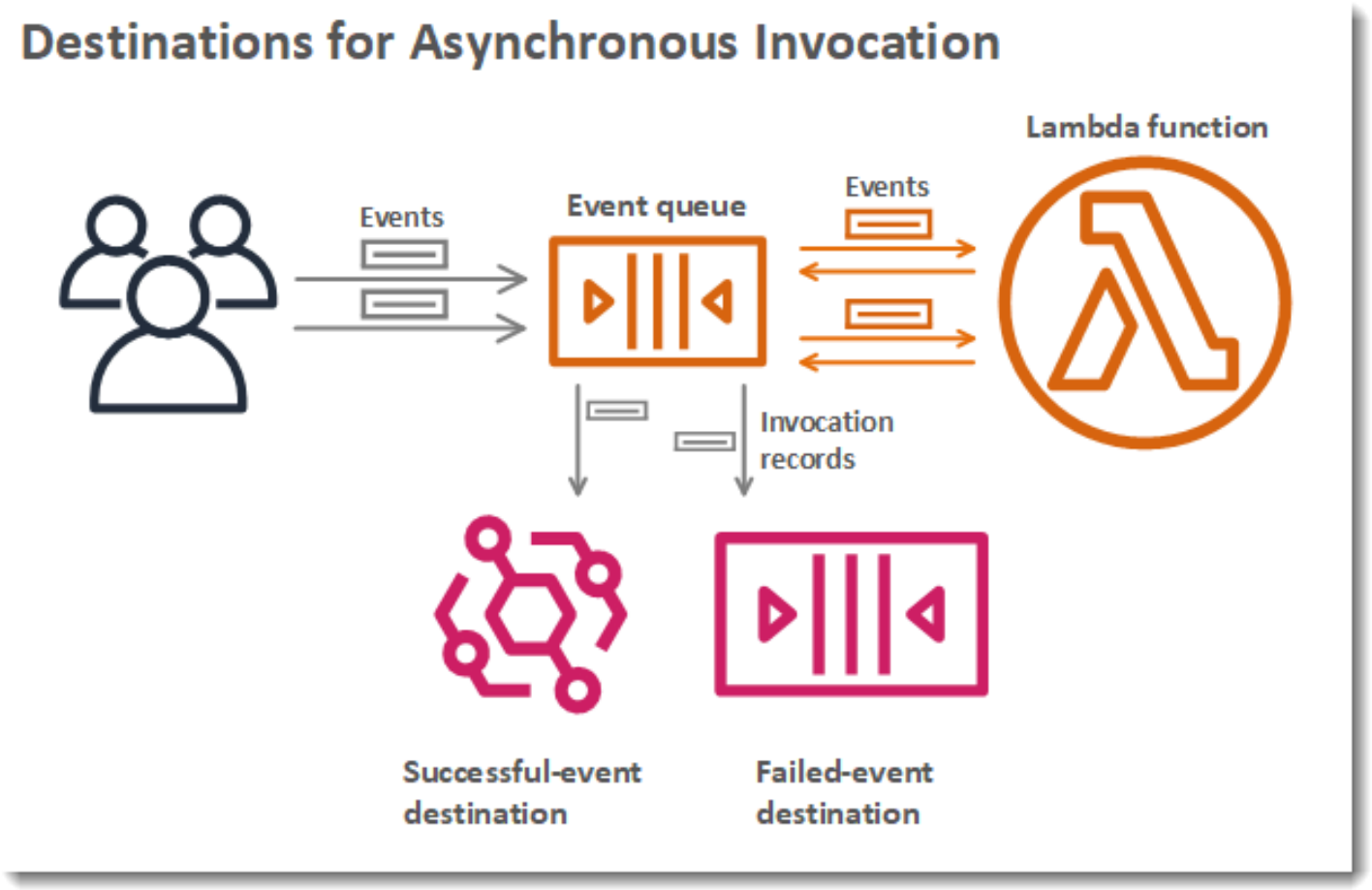


Le mappature delle origini eventi gestiscono una coda locale di elementi non elaborati e gestiscono i nuovi tentativi se la funzione restituisce un errore o viene limitata. È possibile configurare una mappatura delle origini eventi per personalizzare il comportamento di invio in batch e la gestione degli errori o per inviare a una destinazione un record di elementi che non sono stati elaborati.

Per ulteriori informazioni, consulta In che modo Lambda elabora i record provenienti da fonti di eventi basate su stream e code.

Destinazioni

Una destinazione è una AWS risorsa che riceve i record di chiamata per una funzione. Per le chiamate asincrone, puoi configurare Lambda per l'invio dei record di invocazione a una coda, un argomento, una funzione o un bus di eventi. Puoi configurare destinazioni separate per le invocazioni riuscite e gli eventi che non sono stati elaborati. Il record di invocazione contiene i dettagli sull'evento, la risposta della funzione e il motivo per cui il record è stato inviato.



Per le mappature dell'origine evento che leggono dai flussi, puoi configurare Lambda per l'invio di un record dei batch che non sono stati elaborati a una coda o un argomento. Un record di errore per una mappatura dell'origine eventi contiene i metadati relativi al batch e punta agli elementi nel flusso.

Per ulteriori informazioni, consulta Configurazione delle destinazioni per l'invocazione asincrona e le sezioni relative alla gestione degli errori di Utilizzo AWS Lambda con Amazon DynamoDB e In che modo Lambda elabora i record di Amazon Kinesis Data Streams.

Piani delle funzioni

Quando si crea una funzione nella console Lambda, è possibile scegliere di iniziare da zero, utilizzare uno schema o un'immagine di container. Un blueprint fornisce un codice di esempio che mostra come usare Lambda con AWS un servizio o una popolare applicazione di terze parti. I piani includono il codice di esempio e la configurazione della funzione preimpostati per i runtime Node.js e Python.

I blueprint sono forniti per l'uso con la licenza Amazon Software License. Sono disponibili solo nella console Lambda.

Strumenti di test e distribuzione

Lambda supporta l'implementare del codice così com'è o come immagini di container. Puoi utilizzare AWS servizi e strumenti di community popolari come l'interfaccia a riga di comando (CLI) di Docker per creare, creare e distribuire le tue funzioni Lambda. Per configurare la CLI Docker, consulta la pagina Get Docker (Ottieni Docker) sul sito Web Docker Docs. Per un'introduzione all'uso di Docker con AWS, consulta la Guida introduttiva all'utilizzo di Amazon ECR AWS CLI nella Amazon Elastic Container Registry User Guide.

La AWS CLI e la CLI di AWS SAM SAM sono gli strumenti a riga di comando per la gestione degli stack di applicazioni Lambda. Oltre ai comandi per la gestione degli stack di applicazioni con l' AWS CloudFormation API, AWS CLI supporta comandi di livello superiore che semplificano attività come il caricamento di pacchetti di distribuzione e l'aggiornamento dei modelli. La AWS SAM CLI fornisce funzionalità aggiuntive, tra cui la convalida dei modelli, il test locale e l'integrazione con i sistemi CI/CD.

- Installazione della AWS SAM CLI
- Test ed esecuzione del debug di applicazioni serverless con AWS SAM
- Implementazione di applicazioni serverless utilizzando sistemi CI/CD con AWS SAM

Modelli di applicazione

Puoi utilizzare la console Lambda per creare un'applicazione con una pipeline di distribuzione continua. I modelli di applicazione nella console Lambda includono codice per una o più funzioni, un modello di applicazione che definisce funzioni e AWS risorse di supporto e un modello di infrastruttura che definisce una AWS CodePipeline pipeline. La pipeline ha fasi di compilazione e distribuzione che vengono eseguite ogni volta che si eseguono modifiche al repository Git incluso.

I modelli di applicazione sono forniti per l'uso con la licenza MIT No Attribution. Sono disponibili solo nella console Lambda.

Per ulteriori informazioni, consulta Gestione delle applicazioni nella console di AWS Lambda.

Informazioni su come creare soluzioni serverless

> **Tip**
>
> Per scoprire come creare soluzioni serverless, consulta la Guida allo sviluppo serverless.

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.

Ogni versione del linguaggio di programmazione principale ha un runtime separato, con un identificatore di runtime univoco, ad esempio `nodejs20.x` o `python3.12`. 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 compatibilità con le versioni precedenti tra le versioni principali, si tratta di un'operazione gestita dal 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 utilizzare altri linguaggi in Lambda, come Go o Rust, utilizza un runtime solo per il sistema operativo. 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 in fase di esecuzione dopo

la deprecazione". La tabella riporta le date attualmente previste per il ritiro dei runtime. Tali date vengono indicate a scopo di pianificazione e sono soggette a modifiche.

Runtime supportati

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Node.js 20	`nodejs20.x`	Amazon Linux 2023			
Node.js 18	`nodejs18.x`	Amazon Linux 2			
Node.js 16	`nodejs16.x`	Amazon Linux 2	12 giugno 2024	28 febbraio 2025	31 marzo 2025
Python 3.12	`python3.12`	Amazon Linux 2023			
Python 3.11	`python3.11`	Amazon Linux 2			
Python 3.10	`python3.10`	Amazon Linux 2			
Python 3.9	`python3.9`	Amazon Linux 2			
Python 3.8	`python3.8`	Amazon Linux 2	14 ottobre 2024	28 febbraio 2025	31 marzo 2025
Java 21	`java21`	Amazon Linux 2023			
Java 17	`java17`	Amazon Linux 2			

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Java 11	`java11`	Amazon Linux 2			
Java 8	`java8.al2`	Amazon Linux 2			
.NET 8	`dotnet8`	Amazon Linux 2023			
.NET 6	`dotnet6`	Amazon Linux 2	12 novembre 2024	28 febbraio 2025	31 marzo 2025
Rubino 3.3	`ruby3.3`	Amazon Linux 2023			
Ruby 3.2	`ruby3.2`	Amazon Linux 2			
Runtime solo per il sistema operativo	`provided.al2023`	Amazon Linux 2023			
Runtime solo per il sistema operativo	`provided.al2`	Amazon Linux 2			

> **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.

Lambda continua a supportare il linguaggio di programmazione Go dopo il ritiro del runtime Go 1.x. Per ulteriori informazioni, consulta la sezione Migrazione AWS Lambda delle funzioni dal runtime Go1.x al runtime personalizzato su Amazon Linux 2 sul blog di Compute AWS .

Tutti i runtime Lambda supportati supportano sia le architetture x86_64 che 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 rilascio di Node.js, 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.

- Python 3.13: novembre 2024
- Node.js 22: novembre 2024

Policy di deprecazione del runtime

Runtime Lambda per gli archivi di file .zip sono costruiti attorno a una combinazione di sistema operativo, linguaggio di programmazione e librerie software soggette a aggiornamenti di manutenzione e sicurezza. La politica di deprecazione standard di Lambda prevede di rendere obsoleto un runtime quando uno dei componenti principali del runtime raggiunge la fine del supporto comunitario a lungo termine (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 possono più applicare patch o aggiornamenti di sicurezza 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 sulla obsolescenza, consulta le seguenti sezioni e la gestione degli aggiornamenti di runtime sul 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. Una volta raggiunta la data di fine del supporto, Lambda non applica le patch di sicurezza ai runtime dei linguaggi di programmazione.

Deprecazione del runtime per Node.js 16

In risposta al feedback dei clienti, AWS sta posticipando la deprecazione del runtime di Node.js 16 fino a 9 mesi dopo la fine della community LTS. Il runtime Node.js 16 diventerà obsoleto alla data fornita nella tabella Runtime supportati. Come indicato nella nota precedente, tra la fine dell'LTS l'11 settembre 2023 e la data di deprecazione, Lambda applicherà al runtime soltanto le patch del sistema operativo. Durante questo periodo, non verranno applicate patch di sicurezza per il runtime del linguaggio.

Il ritardo della deprecazione di Node.js 16 offre ai clienti che utilizzano questo runtime l'opportunità di migrare le proprie funzioni direttamente a Node.js 20, saltando Node.js 18.

Modello di responsabilità condivisa

Lambda è responsabile della cura e della pubblicazione degli aggiornamenti di sicurezza per tutti i runtime gestiti e le immagini di base dei container supportati. Per impostazione predefinita, Lambda applicherà 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 distribuite utilizzando immagini del contenitore, sei responsabile della ricostruzione dell'immagine del

contenitore della funzione dall'immagine di base più recente e della ridistribuzione dell'immagine del contenitore.

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 vostre 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	Fornisci aggiornamenti di runtime regolari con patch di sicurezza e altri aggiornamenti. Applica automaticamente gli aggiornamenti di runtime per impostazione predefinita (vedi the section called “Controlli di gestione del runtime” per i comportamenti non predefiniti).	Aggiorna il codice della funzione, comprese le dipendenze, per risolvere eventuali vulnerabilità di sicurezza.
Immagine del contenitore supportata	Fornisci aggiornamenti regolari all'immagine di base del contenitore con patch di sicurezza e altri aggiornamenti.	Aggiorna il codice della funzione, comprese le dipendenze, per risolvere eventuali vulnerabilità di sicurezza. Ricostruisci e ridistribuisci regolarmente l'immagine del contenitore utilizzando l'immagine di base più recente.

Fase del ciclo di vita di runtime	Responsabilità di Lambda	Le tue responsabilità
Runtime gestito prossimo alla deprecazione	Avvisate i clienti prima che il runtime diventi obsoleto tramite documentazione, e-mail e. AWS Health Dashboard Trusted Advisor La responsabilità per gli aggiornamenti in fase di esecuzione termina quando diventa obsoleta.	Monitora la documentazione di Lambda AWS Health Dashboard, la posta elettroni ca o le informazioni sull'obso lescenza in fase Trusted Advisor di esecuzione. Aggiorna le funzioni a un runtime supportato prima che il runtime precedente diventi obsoleto.
L'immagine del contenitore sta per diventare obsoleta	Le notifiche di obsolesce nza non sono disponibili per le funzioni che utilizzano immagini del contenitore. La responsabilità per gli aggiornamenti delle immagini di base del contenitore termina quando diventa obsoleta.	Fai attenzione alle pianifica zioni di obsolescenza e alle funzioni di aggiornamento a un'immagine di base supportat a prima che l'immagine precedente diventi obsoleta.

Utilizzo in fase di esecuzione dopo la deprecazione

Dopo che un runtime è diventato obsoleto, non AWS possono più applicare patch o aggiornamenti di sicurezza a quel runtime e le funzioni che utilizzano quel runtime non sono più idonee al supporto tecnico. Tali 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 causare il loro malfunzionamento.

Per almeno 30 giorni dopo il ritiro di un runtime, puoi ancora creare nuove funzioni Lambda utilizzando quel runtime. A partire da 30 giorni dopo il ritiro, Lambda comincia a bloccare la creazione di nuove funzioni.

Per almeno 60 giorni dopo che un runtime è diventato obsoleto, puoi comunque aggiornare il codice e la configurazione della funzione per le funzioni esistenti. A partire da 60 giorni dopo la deprecazione, Lambda inizia a bloccare l'aggiornamento del codice della funzione e della configurazione per le funzioni esistenti.

> **Note**
>
> Per alcuni runtime, AWS posticipa block-function-update le date di fine oltre i normali 30 block-function-create 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 di esecuzione.

È possibile aggiornare una funzione per utilizzare un runtime più recente supportato all'infinito dopo che un runtime è diventato obsoleto. È necessario verificare che la funzione funzioni con il nuovo runtime prima di applicare la modifica del runtime negli ambienti di produzione, poiché non sarà possibile tornare al runtime obsoleto una volta trascorso il periodo di 60 giorni. Consigliamo di utilizzare le versioni e gli alias delle funzioni per consentire una distribuzione sicura con rollback.

Tieni presente che il periodo di tempo esatto per cui potrai continuare a creare e aggiornare le funzioni non è fisso. Questo periodo può variare per ogni deprecazione e per diversa. Regioni AWS Le date nominali per il blocco della creazione e degli aggiornamenti delle funzioni sono indicate nella tabella Runtime supportati nella prima sezione di questa pagina. Lambda non comincerà a bloccare la creazione o gli aggiornamenti delle funzioni prima delle date riportate nella tabella.

Puoi continuare a richiamare le funzioni all'infinito dopo che il runtime è stato ritirato. Tuttavia, si consiglia AWS vivamente di migrare le funzioni a un runtime supportato in modo che le funzioni continuino a ricevere patch di sicurezza e rimangano idonee al supporto tecnico.

Ricezione di notifiche di ritiro del runtime

Quando un runtime si avvicina alla data di deprecazione, Lambda invia un avviso e-mail se alcune funzioni del runtime utilizzano quel runtime. Account AWS Le notifiche vengono visualizzate anche in e in. AWS Health Dashboard 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 seethe section called “Elenca le funzioni che utilizzano un runtime obsoleto”.

Lambda invia una notifica via e-mail 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: AWS Health Dashboard

  AWS Health Dashboard 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 up-to-date elenco completo delle versioni delle funzioni interessate, usa Trusted Advisor or see. the section called “Elenca le funzioni che utilizzano un runtime obsoleto”

  AWS Health Dashboard le notifiche scadono 90 giorni dopo che il runtime interessato è diventato obsoleto.

- Usando AWS Trusted Advisor

  Trusted Advisor visualizza una notifica 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 settimanali dalla Trusted Advisor pagina Preferenze della console. Trusted Advisor

Elenca le funzioni che utilizzano un runtime obsoleto

Oltre Trusted Advisor a utilizzare per visualizzare un elenco in tempo reale delle funzioni interessate dalle deprecazioni di runtime pianificate, puoi anche utilizzare il AWS Command Line Interface (AWS CLI) o uno degli AWS SDK per elencare tutte le versioni delle funzioni che utilizzano un determinato runtime.

Per generare questo elenco utilizzando AWS CLI, esegui il comando seguente. Sostituiscilo `RUNTIME_IDENTIFIER` con il nome del runtime che è diventato obsoleto e scegli il 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 determinata regione. Account AWS Dovrai ripetere questo comando per ogni regione in cui il tuo account ha funzioni e per ognuna delle tue. Account AWS

Per ulteriori informazioni sull'utilizzo di un AWS SDK per elencare le funzioni utilizzando l'ListFunctionsazione, consulta la documentazione SDK relativa al linguaggio di programmazione preferito. Puoi anche utilizzare uno degli AWS SDK per raccogliere statistiche sulle funzioni e sulle most-recently-invoked funzioni più richiamate utilizzando le azioni dell'DescribeLogAPI Streams and Statistics. GetMetric

Puoi anche utilizzare la funzionalità Interrogazioni AWS Config avanzate per elencare tutte le funzioni che utilizzano un runtime interessato. Questa query restituisce solo le versioni della funzione $LATEST, ma è possibile aggregare le query per elencare le funzioni in tutte le regioni e più regioni Account AWS con un solo comando. Per ulteriori informazioni, consulta Interrogazione dello stato di configurazione corrente delle AWS Auto Scaling risorse nella Guida per gli sviluppatori.AWS Config

Runtime obsoleti

Per i seguenti runtime è stata raggiunta la fine del supporto:

Runtime obsoleti

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
.NET 7 (solo container)	`dotnet7`	Amazon Linux 2	14 maggio 2024		
Java 8	`java8`	Amazon Linux	8 gennaio 2024	8 febbraio 2024	28 febbraio 2025
Go 1.x	`go1.x`	Amazon Linux	8 gennaio 2024	8 febbraio 2024	28 febbraio 2025
Runtime solo per il sistema operativo	`provided`	Amazon Linux	8 gennaio 2024	8 febbraio 2024	28 febbraio 2025
Ruby 2.7	`ruby2.7`	Amazon Linux 2	7 dicembre 2023	9 gennaio 2024	28 febbraio 2025
Node.js 14	`nodejs14.x`	Amazon Linux 2	4 dicembre 2023	9 gennaio 2024	28 febbraio 2025
Python 3.7	`python3.7`	Amazon Linux	4 dicembre 2023	9 gennaio 2024	28 febbraio 2025
.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
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		

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
.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		6 marzo 2020
Node.js 4.3	`nodejs4.3`	Amazon Linux	5 marzo 2020		5 marzo 2020
Node.js 4.3 edge	`nodejs4.3-edge`	Amazon Linux	5 marzo 2020		30 aprile 2019
Node.js 6.10	`nodejs6.10`	Amazon Linux	12 agosto 2019	12 agosto 2019	
.NET Core 1.0	`dotnetcore1.0`	Amazon Linux	27 giugno 2019		30 luglio 2019
.NET Core 2.0	`dotnetcore2.0`	Amazon Linux	30 maggio 2019		30 maggio 2019
Node.js 0.10	`nodejs`	Amazon Linux			31 ottobre 2016

Nella quasi totalità dei casi, la end-of-life data della versione linguistica o del sistema operativo è nota con largo anticipo. I seguenti collegamenti forniscono le end-of-life pianificazioni per ogni lingua supportata da Lambda come runtime gestito.

Policy di supporto su linguaggio e framework

- Node.js – github.com
- Python – devguide.python.org
- Ruby – www.ruby-lang.org
- Java – www.oracle.com e Domande frequenti su Corretto
- Go – golang.org
- .NET – dotnet.microsoft.com

Aggiornamenti del runtime Lambda

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. Per ulteriori informazioni, consulta Controlli 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. 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 riga `INIT_START` dei log della funzione e nella console Lambda.

Le versioni di runtime non devono essere confuse con gli identificatori di runtime. Ogni runtime ha un identificatore di runtime univoco, ad esempio `python3.9` o `nodejs18.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 tra le Regioni AWS e le architetture di CPU.

Argomenti

Controlli di gestione 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 ($LATESTo 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 Rollout della versione runtime in due fasi. 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.

La responsabilità dell'applicazione degli aggiornamenti di runtime alle funzioni varia in base alla modalità di aggiornamento del runtime scelta. Per ulteriori informazioni, consulta Modello di responsabilità condivisa.

Rollout della versione runtime in due fasi

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 UpdateFunctionCodeo UpdateFunctionConfigurationAPI.
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 si stanno sviluppando e implementando attivamente le funzioni, è molto probabile che vengano acquistate 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 ti permette di intraprendere azioni correttive in maniera tempestiva. 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 Auto (Aggiornamento automatico) o Function update (Aggiornamento della funzione), è possibile che le funzioni implementate nello stesso momento in regioni diverse o in momenti diversi nella stessa regione abbiano 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.

Rollback di una versione di runtime

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 possono alla fine presentare un peggioramento delle prestazioni 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 Manual (Manuale) per l'aggiornamento del runtime
- Utilizzo delle versioni pubblicate della funzione

Per ulteriori informazioni, consulta Introduzione ai controlli di gestione del runtime AWS Lambda sul blog di AWS Compute.

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.

> **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. Questo approccio può essere integrato nell'integrazione continua e distribuzione continua (CI/CD) per un rollback completamente automatico nel raro caso di incompatibilità degli aggiornamenti di runtime. 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 Modello di responsabilità condivisa.

Identificazione delle modifiche alla versione di runtime

Il numero di versione del runtime e l'ARN vengono registrati nella riga di `INIT_START` registro, 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 registro a CloudWatch Logs, ma non è visibile nella console.

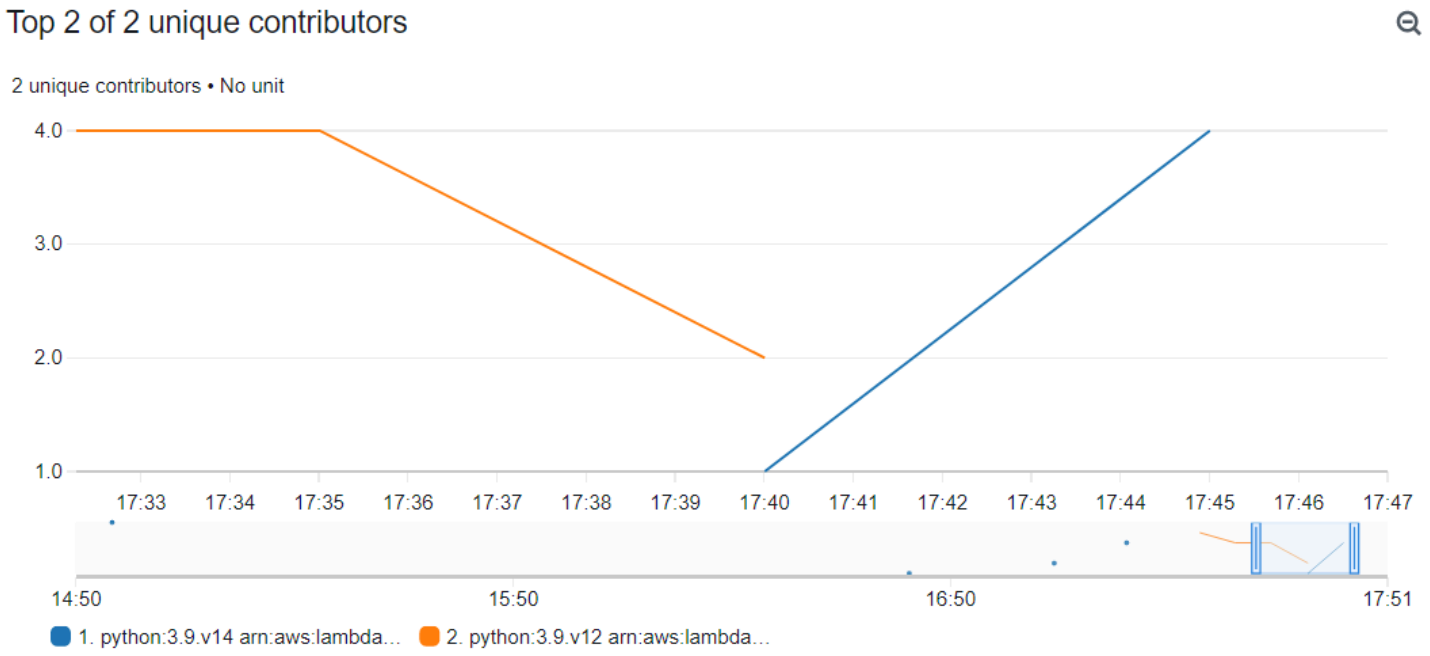
Example Riga di log INIT_START di esempio

```
INIT_START Runtime Version: python:3.9.v14    Runtime Version ARN: arn:aws:lambda:eu-
south-1::runtime:7b620fc2e66107a1046b140b9d320295811af3ad5d4c6a011fad1fa65127e9e6I
```

Invece di 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 report di CloudWatch Contributor Insights mostra un esempio di transizione di una versione di runtime come previsto dalla regola precedente. La riga arancione mostra l'inizializzazione dell'ambiente di esecuzione per la versione di runtime precedente (python:3.9.v12) mentre la linea blu mostra l'inizializzazione dell'ambiente di esecuzione per la nuova versione di runtime (python:3.9.v14).



Configurazione delle impostazioni di gestione del runtime

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).

   > **Note**
   >
   > 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.

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, è possibile utilizzare il comando `put-runtime-management-config` della AWS CLI insieme alla modalità di aggiornamento del runtime. Quando si utilizza la modalità `Manual`, è necessario fornire anche l'ARN della versione di runtime.

```
aws lambda put-runtime-management-config --function-name arn:aws:lambda:eu-west-1:069549076217:function:myfunction --update-runtime-on Manual --runtime-version-arn arn:aws:lambda:eu-west-1::runtime:8eeff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1
```

Verrà visualizzato un output simile al seguente:

```
{
  "UpdateRuntimeOn": "Manual",
  "FunctionArn": "arn:aws:lambda:eu-west-1:069549076217:function:myfunction",
  "RuntimeVersionArn": "arn:aws:lambda:eu-west-1::runtime:8eeff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1"
}
```

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. 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 (Modalità distribuzione)	Responsabilità di Lambda	Responsabilità del cliente
Runtime gestito,	Pubblica nuove versioni di runtime contenenti le patch più recenti.	Torna a una versione di runtime precedente nel raro caso in cui si verifichi un problema di compatibilità con l'aggiornamento del runtime.

Deployment mode (Modalità distribuzione)	Responsabilità di Lambda	Responsabilità del cliente
modalità Auto (Automatico)	Applica le patch di runtime alle funzioni esistenti.	
Runtime gestito, modalità Function update (Aggiornamento della funzione)	Pubblica nuove versioni di runtime contenenti le patch più recenti.	Aggiorna regolarmente le funzioni per utilizzare la versione di runtime più recente. Passa una funzione alla modalità Auto (Automatico) quando questa non viene aggiornata regolarmente. Torna a una versione di runtime precedente nel raro caso in cui si verifichi un problema di compatibilità con l'aggiornamento del runtime.
Runtime gestito, modalità Manual (Manuale)	Pubblica nuove versioni di runtime contenenti le patch più recenti.	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. 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.
Immagine di container	Pubblica nuove immagini di container contenenti le patch più recenti.	Implementa nuovamente le funzioni in maniera regolare utilizzando l'immagine di base del container più recente per utilizzare le patch più recenti.

Per ulteriori informazioni sulla responsabilità condivisa con AWS, consulta Modello di responsabilità condivisa sul sito della sicurezza del Cloud AWS.

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 al funzionamento dell'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 politica su base individuale. case-by-case

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.

Variabili di ambiente specifiche della lingua

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 nei runtime di Node.js.
- `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. Uno script wrapper consente di impostare parametri di configurazione che non possono essere impostati tramite variabili di ambiente specifiche della lingua.

> **Note**
>
> Le chiamate potrebbero non riuscire 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 Runtime solo per il sistema operativo (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 con Python 3.8

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 con Python 3.8

1. Per creare lo script wrapper, incollare il codice seguente in un file denominato `importtime_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[@]}"
```

2. Per assegnare le autorizzazioni eseguibili dello script, immettere `chmod +x importtime_wrapper` dalla riga di comando.

3. Distribuire lo script come livello Lambda.
4. Crea una funzione utilizzando la console Lambda.
    a. Aprire la console Lambda.
    b. Scegli Crea funzione.
    c. In Basic information (Informazioni di base) , per Function name (Nome funzione) , inserisci **`wrapper-test-function`**.
    d. In Runtime, scegliere Python 3.8.
    e. Scegli Crea funzione.
5. Aggiungi il livello alla tua funzione.
    a. Scegliere la funzione, quindi scegliere Codice se non è già selezionato.
    b. Scegliere Add a layer (Aggiungi un livello).
    c. In Scegli un layer, scegliere il nome e la versione del layer compatibile creato in precedenza.
    d. Scegli Aggiungi.
6. Aggiungi il codice e la variabile di ambiente alla tua funzione.
    a. Nell'editor del codice della funzione, incollare il seguente codice funzione:

```
import json

  def lambda_handler(event, context):
      # TODO implement
      return {
          'statusCode': 200,
          'body': json.dumps('Hello from Lambda!')
      }
```

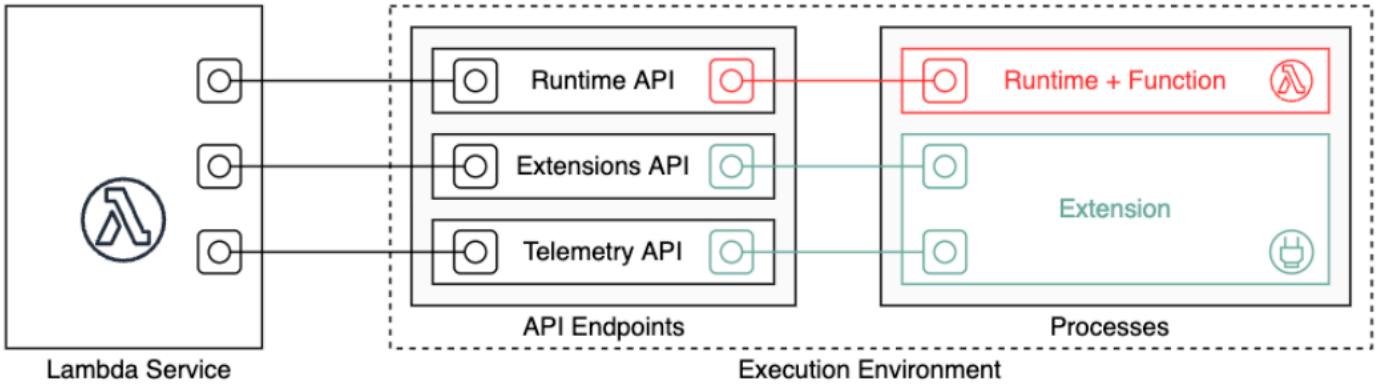
    b. Selezionare Salva.
    c. In Environment variables (Variabili di ambiente), scegliere Edit (Modifica).
    d. Scegli Add environment variable (Aggiungi variabile d'ambiente).
    e. In Chiave, inserire `AWS_LAMBDA_EXEC_WRAPPER`.
    f. In Valore, specifica `/opt/importtime_wrapper`.
    g. Selezionare Salva.
7. Per eseguire la funzione, scegliere Test.

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:

```
...
2020-06-30T18:48:46.780+01:00 import time: 213 | 213 | simplejson
2020-06-30T18:48:46.780+01:00 import time: 50 | 263 | simplejson.raw_json
...
```

API di runtime Lambda

AWS Lambda fornisce un'API HTTP per permettere ai runtime personalizzati di ricevere gli eventi di chiamata da Lambda e restituire i dati della risposta nell'ambiente di esecuzione di 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

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'ID della richiesta che identifica la richiesta che ha attivato la chiamata della funzione.

  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 dall'SDK AWS Mobile, i dati sull'applicazione client e sul dispositivo.
- `Lambda-Runtime-Cognito-Identity` – Per le chiamate dall'SDK Mobile AWS, i dati relativi al provider di identità Amazon Cognito.

Non impostare un timeout sulla richiesta `GET` in quanto la risposta potrebbe essere ritardata. Nell'intervallo di tempo che va dal bootstrap del runtime di Lambda al momento in cui il runtime dispone di un evento da restituire, il processo di runtime potrebbe rimanere bloccato per alcuni secondi.

L'ID della richiesta tiene traccia della chiamata in Lambda. Utilizzalo per specificare la chiamata quando invii la risposta.

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. Con l'SDK X-Ray legge questo valore 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.

Example Richiesta con esito positivo

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response"  -d "SUCCESS"
```

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` – Tipo di errore rilevato dal tempo di esecuzione. Campo obbligatorio: no.

L'intestazione è costituita da un valore stringa. Lambda accetta qualsiasi stringa, ma si consiglia di utilizzare il formato <categoria.motivo>. Per esempio:

- Tempo di esecuzione. NoSuchHandler
- Runtime. API KeyNotFound
- Runtime. ConfigInvalid
- Tempo di esecuzione. UnknownReason

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` – Ulteriori informazioni sull'errore, inviate con i codici di risposta agli errori. ErrorResponse contiene un tipo di errore e un messaggio di errore.

Codice di risposta

- 202 – Accettato
- 403 – Non consentito
- 500 – Errore del container. Stato non recuperabile. Il tempo di esecuzione 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/invocation/AwsRequestId/error`

Metodo – POST

Headers

`Lambda-Runtime-Function-Error-Type` – Tipo di errore rilevato dal tempo di esecuzione. Campo obbligatorio: no.

L'intestazione è costituita da un valore stringa. Lambda accetta qualsiasi stringa, ma si consiglia di utilizzare il formato <categoria.motivo>. Per esempio:

- Runtime. NoSuchHandler
- Runtime. API KeyNotFound
- Runtime. ConfigInvalid
- Tempo di esecuzione. UnknownReason

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` – Ulteriori informazioni sull'errore, inviate con i codici di risposta agli errori. 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 del container. Stato non recuperabile. 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://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/error"
 -d "$ERROR" --header "Lambda-Runtime-Function-Error-Type: Unhandled"
```

Quando usare i runtime solo per il sistema operativo di Lambda

Lambda fornisce runtime gestiti per Java, Python, Node.js, .NET e Ruby. Per creare funzioni Lambda in un linguaggio di programmazione non disponibile come runtime gestito, utilizza un runtime solo per il sistema operativo (la famiglia di runtime `provided`). Esistono tre casi d'uso principali per i runtime solo per il sistema operativo:

- Compilazione nativa ahead-of-time (AOT): linguaggi come Go, Rust e C++ vengono compilati in modo nativo in un binario eseguibile, che non richiede un runtime linguistico dedicato. Questi linguaggi richiedono solo un ambiente di sistema operativo in cui sia possibile eseguire il file binario compilato. Puoi anche utilizzare runtime solo per il sistema operativo Lambda per implementare file binari compilati con .NET Native AOT e Java GraalVM Native.

  È necessario includere un client dell'interfaccia di runtime nel file binario. Il client dell'interfaccia di runtime chiama API di runtime Lambda per recuperare le invocazioni della funzione e quindi esegue la chiamata al gestore della funzione. Lambda fornisce client dell'interfaccia di runtime per Go, .NET Native AOT, C++ e Rust (sperimentale).

  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).
- Runtime di terze parti: puoi eseguire funzioni Lambda off-the-shelf utilizzando runtime come Bref per PHP o Swift Runtime per Swift. AWS Lambda
- Runtime personalizzati: puoi creare il tuo runtime personale per un linguaggio o una versione di linguaggio per cui Lambda non fornisce un runtime gestito, come Node.js 19. Per ulteriori informazioni, consulta Creazione di un runtime personalizzato per AWS Lambda. Questo è il caso d'uso meno comune per i runtime solo per il sistema operativo.

Lambda supporta i seguenti runtime solo per il sistema operativo:

Solo per il sistema operativo

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Runtime solo per il sistema operativo	`provided.al2023`	Amazon Linux 2023			

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Runtime solo per il sistema operativo	`provided.al2`	Amazon Linux 2			

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 sul 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 runtime solo per il sistema operativo (la famiglia di runtime `provided`).

> **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'utilizzo di un off-the-shelf runtime di terze parti, consultaQuando usare i runtime solo per il sistema operativo di Lambda.

Per una procedura guidata sul processo di implementazione del runtime personalizzato, consulta Tutorial: Creazione di un runtime personalizzato. Puoi anche esplorare un runtime personalizzato implementato in C++ su aws-lambda-cppawslabs/ on. GitHub

Argomenti

- Requisiti
- Implementazione dello streaming delle risposte in un runtime personalizzato

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.
- Propagazione dell'intestazione di traccia – Recupera l'intestazione di traccia X-Ray dall'intestazione `Lambda-Runtime-Trace-Id` nella risposta dell'API. Imposta la variabile di ambiente `_X_AMZN_TRACE_ID` locale allo stesso valore. L'SDK X-Ray utilizza questo valore per associare i dati di tracciamento 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.

Di seguito è riportato 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/invocation/AwsRequestId/response`: propaga l'intestazione `Content-Type` dal runtime per inviarla al client. Lambda restituisce il payload di risposta in blocchi tramite codifica di trasferimento in blocchi HTTP/1.1. Il flusso della risposta può avere una dimensione massima di 20 MiB. 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`.
  - Scrivere 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/invocation/AwsRequestId/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/invocation/AwsRequestId/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. Si tratta di un requisito della specifica di codifica di trasferimento in blocchi HTTP/1.1.

- `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 si consiglia di utilizzare il formato *<categoria.motivo>*. Ad esempio, `Runtime.APIKeyNotFound`.
- `Lambda-Runtime-Function-Error-Body`: informazioni sull'errore nella codifica base64.

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 l'AWS Command Line Interface (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.

È necessario un ruolo IAM per creare una funzione Lambda. Il ruolo richiede l'autorizzazione per inviare registri a CloudWatch Logs e accedere ai AWS servizi 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 AWSLambdaBasicExecutionRolepolitica 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)

  # Run the handler function from the script
  RESPONSE=$($(echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")

  # Send the response
  curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response"  -d "$RESPONSE"
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.

> **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'opzione cli-binary-format è necessaria 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 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 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 `boostrap` 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'opzione cli-binary-format è necessaria 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 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=$(grep -Fi Lambda-Runtime-Aws-Request-Id "$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
  curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response"  -d "$RESPONSE"
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 concedere l'autorizzazione per l'utilizzo del livello a un altro account, aggiungi un'istruzione alla policy delle autorizzazioni della versione del livello con il comando add-layer-version-permission. In ogni istruzione, puoi concedere l'autorizzazione a un singolo account, a tutti gli account o a un'organizzazione.

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 --statement-id
 xaccount \
--action lambda:GetLayerVersion --principal 111122223333 --version-number 2 --output
 text
```

Verrà visualizzato un output simile al seguente:

```
e210ffdc-e901-43b0-824b-5fcd0dd26d16    {"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
```

Utilizzo della vettorizzazione AVX2 in Lambda

Advanced Vector Extensions 2 (AVX2) è un'estensione vettoriale del set di istruzioni Intel x86 che può eseguire istruzioni SIMD (Single Instruction Multiple Data) su vettori di 256 bit. Per algoritmi vettorizzabili con funzionamento altamente parallelizzabile, l'utilizzo di AVX2 può migliorare le prestazioni della CPU, con conseguente latenze inferiori e velocità effettiva superiore. Utilizzare il set di istruzioni AVX2 per carichi di lavoro ad alta intensità di elaborazione come l'inferenziazione del machine learning, l'elaborazione multimediale, le simulazioni scientifiche e le applicazioni di modellazione finanziaria.

> **Note**
>
> Lambda arm64 utilizza l'architettura NEON SIMD e non supporta le estensioni AVX2 x86.

Per utilizzare AVX2 con la funzione Lambda, assicurarsi che il codice funzione stia accedendo al codice ottimizzato AVX2. Per alcune lingue, è possibile installare la versione supportata da AVX2 di librerie e pacchetti. Per altre lingue, è possibile ricompilare il codice e le dipendenze con i flag del compilatore appropriati impostati (se il compilatore supporta la vettorizzazione automatica). È inoltre possibile compilare il codice con librerie di terze parti che utilizzano AVX2 per ottimizzare le operazioni matematiche. Ad esempio, Intel Math Kernel Library (Intel MKL), OpenBLAS (Basic Linear Algebra Subprograms) e BLIS (Library Instantiation Software) AMD BLAS-like. I linguaggi vettorizzati automaticamente, come Java, utilizzano automaticamente AVX2 per i calcoli.

È possibile creare nuovi carichi di lavoro Lambda o spostare carichi di lavoro esistenti abilitati per AVX2 su Lambda senza costi aggiuntivi.

Per ulteriori informazioni su AVX2, consulta Advanced Vector Extensions 2 in Wikipedia.

Compilazione dall'origine

Se la funzione Lambda utilizza una libreria C o C ++ per eseguire operazioni vettoriali ad uso intensivo di calcolo, è possibile impostare i flag del compilatore appropriati e ricompilare il codice della funzione. Quindi, il compilatore vettorizza automaticamente il codice.

Per il compilatore `gcc` o `clang`, aggiungere `-march=haswell` al comando o impostare `-mavx2` come opzione di comando.

```
~ gcc -march=haswell main.c
or
~ gcc -mavx2 main.c

~ clang -march=haswell main.c
or
~ clang -mavx2 main.c
```

Per utilizzare una libreria specifica, seguire le istruzioni nella documentazione della libreria per compilare e compilare la libreria. Ad esempio, per creare TensorFlow dal codice sorgente, puoi seguire le istruzioni di installazione sul TensorFlow sito Web. Assicurarsi di utilizzare l'opzione di compilazione `-march=haswell`.

Abilitazione di AVX2 per Intel MKL

Intel MKL è una libreria di operazioni matematiche ottimizzate che utilizzano implicitamente le istruzioni AVX2 quando la piattaforma di elaborazione le supporta. Framework come PyTorch build with Intel MKL per impostazione predefinita, quindi non è necessario abilitare AVX2.

Alcune librerie, ad esempio TensorFlow, forniscono opzioni nel processo di compilazione per specificare l'ottimizzazione Intel MKL. Ad esempio, con TensorFlow, utilizzate l'`--config=mkl`opzione.

Puoi anche creare librerie Python scientifiche popolari, come SciPy e NumPy, con Intel MKL. Per istruzioni sulla creazione di queste librerie con Intel MKL, consulta Numpy/Scipy con Intel MKL e compilatori Intel sul sito Web Intel.

Per ulteriori informazioni su Intel MKL e librerie simili, consultate Math Kernel Library in Wikipedia, il sito Web OpenBLAS e il repository AMD BLIS su. GitHub

Supporto AVX2 in altre lingue

Se non si utilizzano librerie C o C++ e non si compila con Intel MKL, è comunque possibile ottenere un miglioramento delle prestazioni AVX2 per le applicazioni. Si noti che il miglioramento effettivo dipende dalla capacità del compilatore o dell'interprete di utilizzare le funzionalità AVX2 sul codice.

Python

Gli utenti di Python generalmente utilizzano le NumPy librerie SciPy and per carichi di lavoro ad alta intensità di calcolo. È possibile compilare queste librerie per abilitare AVX2 oppure utilizzare le versioni abilitate per Intel MKL delle librerie.

Nodo

Per i carichi di lavoro a elaborazione intensiva, utilizzare le versioni abilitate per AVX2 o abilitate per Intel MKL delle librerie necessarie.

Java

Il compilatore JIT di Java può vettorizzare automaticamente il codice per essere eseguito con le istruzioni AVX2. Per informazioni sul rilevamento di codice vettoriale, consulta la vettorizzazione del codice nella presentazione JVM sul sito Web OpenJDK.

Go

Il compilatore Go standard non supporta attualmente la vettorizzazione automatica, ma è possibile utilizzare gccgo, il compilatore GCC per Go. Impostare l'opzione `-mavx2`:

```
gcc -o avx2 -mavx2 -Wall main.c
```

Intrinseca

È possibile utilizzare funzioni intrinseche in molte lingue per vettorizzare manualmente il codice per utilizzare AVX2. Tuttavia, non consigliamo questo approccio. La scrittura manuale di codice vettoriale richiede uno sforzo significativo. Inoltre, il debug e la manutenzione di tale codice è più difficile rispetto all'utilizzo di codice che dipende dalla vettorizzazione automatica.

Configurazione delle funzioni AWS Lambda

Scopri come configurare le funzionalità e le opzioni principali per la funzione Lambda tramite l'API o la console Lambda.

Memoria

Scopri come e quando aumentare la memoria funzionale.

Archiviazione effimera

Scopri come e quando aumentare la capacità di archiviazione temporanea della tua funzione.

Timeout

Scopri come e quando aumentare il valore di timeout della tua 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 AWS risorse in un Amazon VPC. La connessione della funzione a un VPC consente di accedere alle risorse in una sottorete privata, ad esempio 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.

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.

Configurazione della memoria funzionale 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 allocate alla funzione utilizzando l'impostazione Memoria. È possibile configurare la memoria tra 128 MB e 10.240 MB con incrementi di 1 MB. A 1.769 MB, una funzione ha l'equivalente di una vCPU (un vCPU-secondo di crediti al secondo).

Questa pagina descrive come e quando aggiornare l'impostazione della memoria per una funzione Lambda.

Sections

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 solo 128 MB per funzioni Lambda semplici, come quelle che trasformano e instradano gli eventi verso altri AWS servizi. Un'allocazione di memoria più elevata 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 legata alla CPU, alla rete o alla memoria, l'aumento dell'impostazione della memoria può migliorarne notevolmente le prestazioni.

Per trovare la configurazione di memoria giusta per le tue funzioni, ti consigliamo di utilizzare lo strumento open source AWS Lambda Power Tuning. 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 nell' AWS account ed esegue

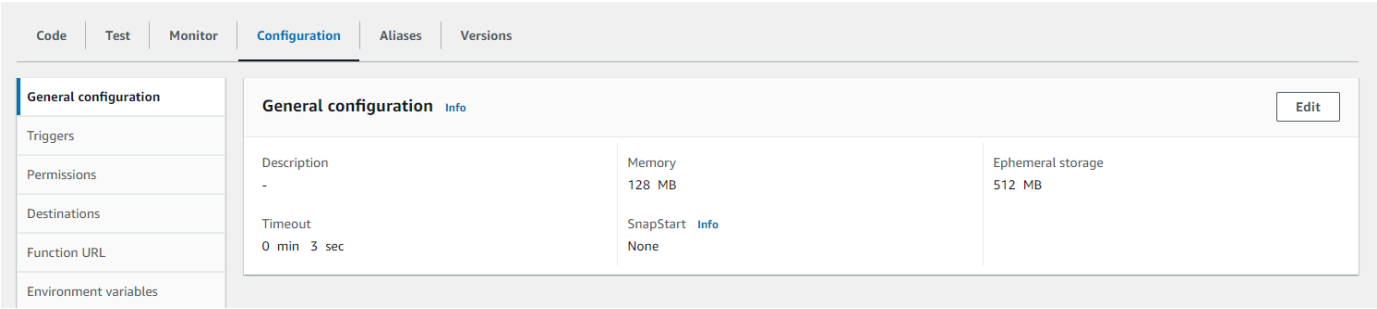
chiamate HTTP in tempo reale e interazioni SDK, per misurare le probabili prestazioni in uno scenario di produzione live. Puoi anche implementare un processo CI/CD per utilizzare questo strumento per misurare automaticamente le prestazioni delle nuove funzioni che distribuisci.

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. Scegli la scheda Configurazione, quindi scegli Configurazione generale.



4. In Configurazione generale, scegli Modifica.
5. Per Memoria, imposta un valore compreso tra 128 MB e 10.240 MB.
6. Selezionare Salva.

Configurazione della funzione memory ()AWS CLI

È possibile utilizzare il update-function-configurationcomando per configurare la memoria della funzione.

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
  --memory-size 1024
```

Configurazione della funzione memory ()AWS SAM

È possibile utilizzare il AWS Serverless Application Modelper configurare la memoria per la propria funzione. Aggiorna la MemorySizeproprietà 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 della funzione 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.

Configura lo storage temporaneo per le funzioni Lambda

Lambda fornisce una memorizzazione temporanea per le funzioni nella directory. `/tmp` Questo storage 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 di archiviazione temporanea. È possibile configurare lo storage temporaneo tra 512 MB e 10.240 MB, con incrementi di 1 MB. Tutti i dati archiviati vengono crittografati quando sono inattivi con `/tmp` una chiave gestita da. AWS

Questa pagina descrive i casi d'uso comuni e come aggiornare lo storage temporaneo per una funzione Lambda.

Sections

Casi d'uso comuni per aumentare lo storage temporaneo

Ecco alcuni casi d'uso comuni che traggono vantaggio dall'aumento dello spazio di archiviazione effimero:

- Job E xtract-transform-load (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 machine learning (ML): molte attività di inferenza si basano su file di dati di riferimento di grandi dimensioni, tra cui librerie e modelli. Con uno storage più temporaneo, puoi scaricare modelli più grandi da Amazon Simple Storage Service (Amazon S3) e `/tmp` utilizzarli nell'elaborazione.
- Elaborazione dei dati: per i carichi di lavoro che scaricano oggetti da Amazon S3 in risposta a eventi S3, `/tmp` più spazio 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 uno storage più effimero.
- Elaborazione grafica: l'elaborazione delle immagini è un caso d'uso comune per le applicazioni basate su Lambda. Per i carichi di lavoro che elaborano file TIFF di grandi dimensioni o immagini

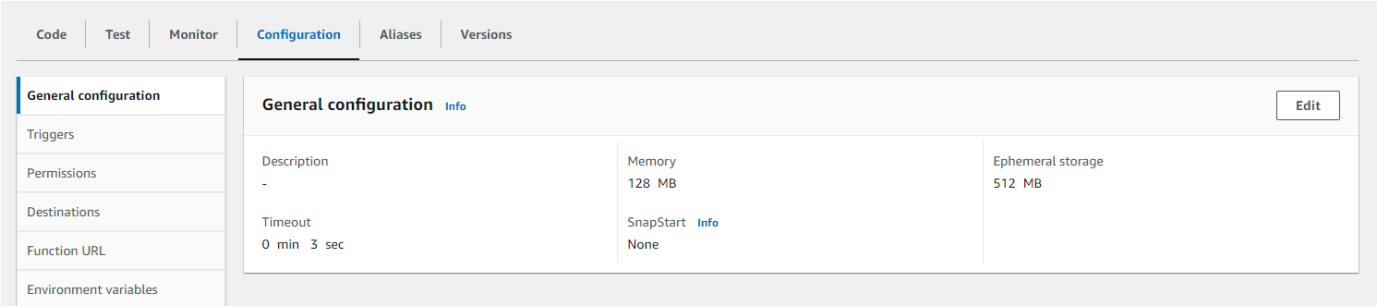
satellitari, uno storage più temporaneo semplifica l'uso delle librerie e l'esecuzione del calcolo in Lambda.

Configurazione dell'archiviazione temporanea (console)

Puoi configurare lo storage temporaneo nella console Lambda.

Per modificare la memorizzazione temporanea per una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegli la scheda Configurazione, quindi scegli Configurazione generale.



4. In Configurazione generale, scegli Modifica.
5. Per l'archiviazione temporanea, imposta un valore compreso tra 512 MB e 10.240 MB, con incrementi di 1 MB.
6. Selezionare Salva.

AWS CLI Configurazione dello storage effimero ()

È possibile utilizzare il update-function-configurationcomando per configurare l'archiviazione temporanea.

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
  --ephemeral-storage '{"Size": 1024}'
```

Configurazione dello storage temporaneo ()AWS SAM

Puoi usare il AWS Serverless Application Modelper configurare la memorizzazione temporanea per la tua funzione. Aggiorna la EphemeralStorageproprietà 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: 128
      Timeout: 120
      Handler: index.handler
      Runtime: nodejs20.x
      Architectures:
        - x86_64
      EphemeralStorage:
        Size: 10240
      # Other function properties...
```

Configura 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).

Questa pagina descrive come e quando aggiornare l'impostazione del timeout per una funzione Lambda.

Sections

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 più grandi 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.

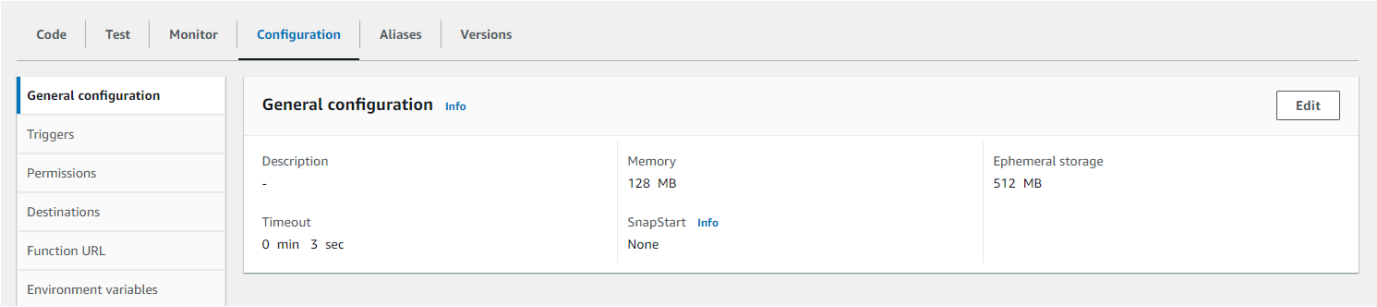
Quando testate l'applicazione, assicuratevi che i test riflettano accuratamente la dimensione e la quantità di dati e i valori realistici dei parametri. I test utilizzano spesso campioni di piccole dimensioni per comodità, ma è consigliabile utilizzare set di dati al limite massimo di quanto ragionevolmente 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. Scegli 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. Selezionare Salva.

Configurazione del timeout ()AWS CLI

È possibile utilizzare il update-function-configurationcomando per configurare il valore di timeout, in secondi. 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

Puoi usare il AWS Serverless Application Modelper configurare il valore di timeout per la tua funzione. Aggiorna la proprietà Timeout 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: 128
      Timeout: 120
      # Other function properties...
```

Usa le variabili di ambiente Lambda per configurare i valori nel codice

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, 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 Creare e gestire segreti con. AWS Secrets Manager

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.

Puoi configurare le variabili di ambiente in Lambda utilizzando la console Lambda, (), AWS Command Line Interface AWS Serverless Application Model (AWS CLI) o un AWS SAM SDK. AWS

Console

Puoi definire le variabili di ambiente sulla versione non pubblicata della funzione. Quando pubblicate una versione, le variabili di ambiente vengono bloccate per quella versione insieme ad altre impostazioni di configurazione 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. Selezionare 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. Scegli la scheda Variabili di ambiente.
3. Scegli Strumenti, quindi Mostra variabili di ambiente.

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=DOC-EXAMPLE-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": "nodejs20.x",
    "Role": "arn:aws:iam::111122223333:role/lambda-role",
    "Environment": {
        "Variables": {
            "BUCKET": "DOC-EXAMPLE-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

Puoi usare il AWS Serverless Application Modelper configurare le variabili di ambiente per la tua funzione. Aggiorna le proprietà Environment e Variables nel tuo `template.yaml` file, 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: nodejs18.x
      Architectures:
        - x86_64
      EphemeralStorage:
        Size: 10240
      Environment:
        Variables:
          BUCKET: DOC-EXAMPLE-BUCKET
          KEY: file.txt
      # Other function properties...
```

AWS SDKs

Per gestire le variabili di ambiente utilizzando un AWS SDK, utilizza le seguenti operazioni API.

- UpdateFunctionConfigurazione
- GetFunctionConfigurazione
- CreateFunction

Per ulteriori informazioni, consulta la documentazione AWS SDK relativa al linguaggio di programmazione preferito.

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 traccia X-Ray. Questa variabile di ambiente cambia a ogni invocazione.
  - Questa variabile di ambiente non è definita per i runtime solo per il sistema operativo (la famiglia di runtime `provided`). Puoi impostare `_X_AMZN_TRACE_ID` per runtime personalizzati utilizzando l'intestazione di risposta `Lambda-Runtime-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 agli 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 runtime solo per il sistema operativo (la famiglia di runtime `provided`).
- `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`. Per informazioni, consulta Configurazione della simultaneità con provisioning o Migliorare le prestazioni di avvio con Lambda SnapStart.
- `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 funzioni Lambda SnapStart .
- `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.

Le seguenti variabili di ambiente aggiuntive non sono riservate e possono essere estese nella configurazione della funzione.

Variabili d'ambiente non riservate

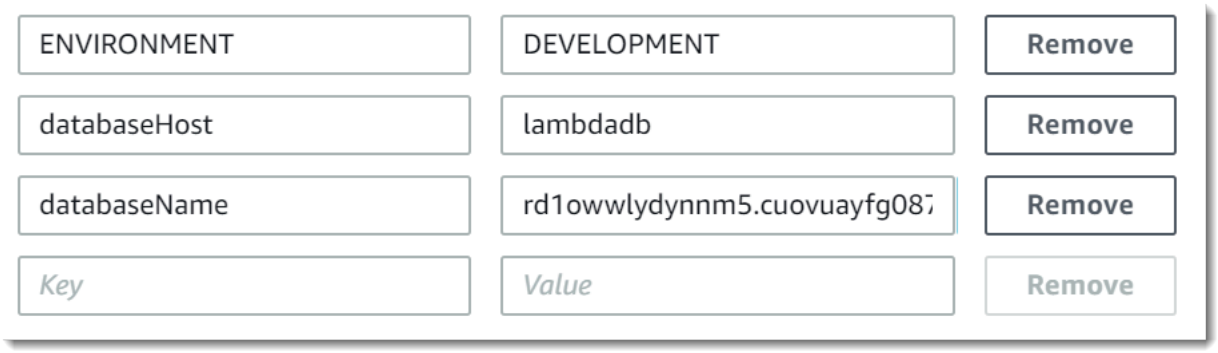
- `LANG` – Le impostazioni locali del runtime (`en_US.UTF-8`).
- `PATH`: il percorso di esecuzione (`/usr/local/bin:/usr/bin/:/bin:/opt/bin`).
- `LD_LIBRARY_PATH`: il percorso della libreria di sistema (`/var/lang/lib:/lib64:/usr/lib64:$LAMBDA_RUNTIME_DIR:$LAMBDA_RUNTIME_DIR/lib:$LAMBDA_TASK_ROOT:$LAMBDA_TASK_ROOT/lib:/opt/lib`).
- `NODE_PATH` – (Node.js) Il percorso della libreria Node.js (`/opt/nodejs/node12/node_modules/:/opt/nodejs/node_modules:$LAMBDA_RUNTIME_DIR/node_modules`).
- `PYTHONPATH` – (Python 2.7, 3.6, 3.8) Il percorso della libreria Python (`$LAMBDA_RUNTIME_DIR`).
- `GEM_PATH` – (Ruby) Il percorso della libreria Ruby (`$LAMBDA_TASK_ROOT/vendor/bundle/ruby/2.5.0:/opt/ruby/gems/2.5.0`).
- `AWS_XRAY_CONTEXT_MISSING` – Per il tracciamento X-Ray, Lambda imposta questo valore su `LOG_ERROR` per evitare di generare errori di runtime dall'SDK X-Ray.
- `AWS_XRAY_DAEMON_ADDRESS` – Per il tracciamento X-Ray, l'indirizzo IP e la porta del daemon X-Ray.
- `AWS_LAMBDA_DOTNET_PREJIT`: per i runtime .NET 6. .NET 7, 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 concorrenza 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.

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.



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.

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 memorizzare le credenziali del database. Per ulteriori informazioni, consulta Utilizzo AWS Lambda con Amazon RDS.

Sicurezza dei dati inattivi

Lambda fornisce sempre la crittografia lato server dei dati inattivi 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. Lambda li crea Chiave gestita da AWS nel tuo account e ne gestisce le autorizzazioni per conto tuo. 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 decritta le variabili di ambiente. Per visualizzare un esempio, scegli Decritta 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. Selezionare 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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Deny",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-east-2:111122223333:key/3be10e2d-xmpl-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 .

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 AWS CloudFormation modello con. AWS Secrets Manager

Offrire alle funzioni Lambda l'accesso alle risorse in un Amazon VPC

Con Amazon Virtual Private Cloud (Amazon VPC), puoi creare reti private Account AWS al tuo interno per ospitare risorse 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 seguenti sezioni per collegare una funzione Lambda a un Amazon VPC utilizzando la console Lambda, il () 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 gestito da Lambda all'interno del quale viene eseguita la funzione.

Sections

Autorizzazioni IAM richieste

Per collegare una funzione Lambda a un Amazon VPC nel tuo, Account AWS Lambda necessita delle autorizzazioni per creare e gestire le interfacce di rete che utilizza per consentire alla funzione di accedere alle risorse del 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 “Comprensione delle interfacce di rete elastiche (ENI) Hyperplane”

Puoi concedere alla tua funzione le autorizzazioni necessarie allegando la policy AWS gestita AWSLambdaVPCAccessExecutionRoleal ruolo di esecuzione della funzione. Quando crei una nuova funzione nella console Lambda e la colleghi a un VPC, Lambda aggiunge automaticamente questa politica di autorizzazioni per te.

Se preferisci creare la tua politica di autorizzazioni IAM, assicurati di aggiungere tutte le seguenti autorizzazioni:

- ec2: Interfaccia CreateNetwork
- ec2: DescribeNetwork Interfacce — Questa azione funziona solo se è consentita su tutte le risorse (). `"Resource": "*"`
- ec2: DescribeSubnets
- ec2: DeleteNetwork Interfaccia — Se non specifichi un ID di risorsa per DeleteNetworkInterface nel ruolo di esecuzione, la tua funzione potrebbe non essere in grado di accedere al VPC. Specificare un ID risorsa univoco o includere tutti gli ID risorsa, ad esempio `"Resource": "arn:aws:ec2:us-west-2:123456789012:*/*"`.
- ec2: AssignPrivate IpAddresses
- ec2: UnassignPrivate IpAddresses

Nota 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 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: Gruppi DescribeSecurity
- ec2: DescribeSubnets
- ec2: DescribeVpcs

> **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 consigli su come seguire le migliori pratiche di sicurezza, consultathe section called “Best practice di sicurezza”.

Collegamento di funzioni Lambda a un Amazon VPC nel tuo Account AWS

Collega la tua funzione a un Amazon VPC del tuo dispositivo Account AWS utilizzando la console Lambda, oppure. AWS CLI AWS SAM Se utilizzi AWS CLI o o AWS SAM colleghi una funzione esistente a un VPC utilizzando la console Lambda, assicurati che il ruolo di esecuzione della 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 VPC per la funzione effettuando le seguenti operazioni:
    a. Espandere Advanced settings (Impostazioni avanzate).
    b. Seleziona Abilita VPC, quindi seleziona 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 richiede l'accesso a Internet, vedi. the section called “Accesso a Internet per le funzioni 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 tua 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 richiede l'accesso a Internet, vedi. the section called “Accesso a Internet per le funzioni VPC” La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet o a un indirizzo IP pubblico.

7. Selezionare 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 seguente comando CLI. `create-function`

```
aws lambda create-function --function-name my-function \
--runtime nodejs20.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-07fd123456788a03
```

Specificate le vostre sottoreti e i vostri gruppi di sicurezza e impostateli su `true` o in `false` base `Ipv6AllowedForDualStack` al vostro caso d'uso.

Per collegare una funzione esistente a un Amazon VPC

- Per collegare una funzione esistente a un VPC, esegui il seguente comando CLI. `update-function-configuration`

```
aws lambda update-function-configuration --function-name my-function \
--vpc-config Ipv6AllowedForDualStack=true,
 SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a036,SecurityGroupIds=sg-085912
```

Per disconnettere la tua funzione da un VPC

- Per scollegare la tua funzione da un VPC, esegui il seguente comando `update-function-configuration` CLI 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 `VpcConfig` proprietà alla definizione della funzione come mostrato nel seguente modello di esempio. Per ulteriori informazioni su questa proprietà, vedere AWS: :Lambda: :Function VpcConfig nella Guida per l'AWS CloudFormation utente (la AWS SAM `VpcConfig` proprietà viene passata direttamente alla `VpcConfig` proprietà di una AWS CloudFormation `AWS::Lambda::Function` risorsa).

```
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:
        - AWSLambdaVPCAccessExecutionRole

  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 tuo VPC AWS SAM in, AWS consulta: :EC2: :VPC nella Guida per l'utente.AWS CloudFormation

Accesso a Internet se collegato a un VPC

Per impostazione predefinita, le funzioni Lambda hanno accesso alla rete Internet pubblica. Quando colleghi la tua funzione a un VPC, può accedere solo alle risorse disponibili all'interno di quel VPC. Per consentire alla tua funzione di accedere a Internet, devi anche configurare il VPC in modo che abbia accesso a Internet. Per ulteriori informazioni, consulta the section called “Accesso a Internet per le funzioni VPC”.

Le migliori pratiche per l'utilizzo di Lambda con Amazon VPC

Per garantire che la configurazione del VPC Lambda soddisfi le linee guida sulle best practice, segui i consigli nelle sezioni seguenti.

Best practice di sicurezza

Per collegare la tua 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 vengono concesse implicitamente anche al codice della funzione. Ciò significa che il codice della funzione è autorizzato a effettuare queste chiamate API Amazon EC2.

Per seguire il principio dell'accesso con privilegi minimi, aggiungi una politica di negazione come quella riportata nell'esempio seguente al ruolo di esecuzione della funzione. Questa policy impedisce alla tua funzione di effettuare chiamate alle API Amazon EC2 utilizzate dal servizio Lambda per collegare la tua funzione a un VPC.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
```

```
                "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 (ACL) di rete 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 le best practice di sicurezza per il tuo VPC nella Amazon Virtual Private Cloud User Guide.

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. Gli ENI Hyperplane sono associati a una particolare combinazione di gruppi di sicurezza e sottoreti VPC. Se hai già collegato una funzione a un VPC, specificare le stesse sottoreti e gli stessi 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 sugli ENI Hyperplane e sul loro ciclo di vita, vedi. the section called “Comprensione delle interfacce di rete elastiche (ENI) Hyperplane”

Comprensione delle interfacce di rete elastiche (ENI) Hyperplane

Un 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 questi 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, conoscerne il funzionamento 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 un ENI Hyperplane. Questo ENI può essere utilizzato anche da altre funzioni dell'account che utilizzano la stessa combinazione di sottorete e gruppo di sicurezza. Ove possibile, Lambda riutilizza gli ENI esistenti per ottimizzare l'utilizzo delle risorse e ridurre al minimo la creazione di nuovi ENI. Ogni Hyperplane ENI supporta fino a 65.000 connessioni/porte. 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 concorrenza.

Per le nuove funzioni, mentre Lambda crea un ENI Hyperplane, la funzione rimane nello stato Pending e non è possibile richiamarla. La funzione passa allo stato Attivo solo quando Hyperplane ENI è pronto, operazione 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.

> **Note**
>
> Se una funzione Lambda rimane inattiva per 30 giorni, Lambda recupera tutti gli ENI Hyperplane non utilizzati e imposta lo stato della funzione su inattivo. Il prossimo tentativo di chiamata avrà esito negativo e la funzione rientrerà nello stato Pending finché Lambda non completerà la creazione o l'allocazione di un ENI Hyperplane. Per ulteriori informazioni sugli stati delle funzioni Lambda, vedere. the section called “Stati delle funzioni”

Per ulteriori informazioni sui cicli di vita di Hyperplane ENI, vedere. the section called “Hyperplane ENI di Lambda”

Utilizzo dei tasti di condizione IAM per le impostazioni VPC

È possibile utilizzare chiavi di condizione Lambda specifiche per le impostazioni VPC per fornire controlli di autorizzazione aggiuntivi per le 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: SecurityGroup Ids: consente o nega uno o più gruppi di sicurezza.

Le operazioni CreateFunctione la UpdateFunctionconfigurazione dell'API Lambda supportano 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.

Assicurarsi che gli utenti distribuiscano solo funzioni connesse a VPC

Per garantire che tutti gli utenti distribuiscano solo funzioni connesse a VPC, è possibile 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.

```
{
  "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`.

```
{
  "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`.

```
{
  "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 ElastiCache Amazon VPC

Abilita l'accesso a Internet per le funzioni Lambda connesse a VPC

Per impostazione predefinita, le funzioni Lambda vengono eseguite in un VPC gestito da Lambda 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 Lambda connesse a VPC.

Non ho ancora un VPC

Creazione del VPC

Il flusso di lavoro Create 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 alla 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 Blocco CIDR IPv6, Blocco CIDR IPv6 fornito da Amazon.
5. Configurazione delle sottoreti
    a. In Numero di zone di disponibilità, scegli 2. Consigliamo almeno due AZ per un'elevata 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 Subnet CIDR blocks.

6. Per Gateway NAT, scegli 1 per AZ per migliorare la resilienza.
7. Per il gateway Internet solo Egress, scegli Sì se hai scelto di includere un blocco CIDR IPv6.
8. Per gli endpoint VPC, mantieni l'impostazione predefinita (Gateway S3). Questa opzione non prevede alcun costo. Per ulteriori informazioni, consulta Tipi di endpoint VPC per Amazon S3.
9. Per le 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 tramite il gateway NAT. Il collegamento di una funzione a una sottorete pubblica non le consente 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 AWSLambdaVPCAccessExecutionRole AWS gestita. Le autorizzazioni in questa politica sono necessarie solo per creare interfacce di rete elastiche per la configurazione VPC, non per richiamare la tua funzione. Per applicare le autorizzazioni con privilegi minimi, puoi rimuovere la AWSLambdaVPCAccessExecutionRolepolicy 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 politica AWSLambdaVPCAccessExecutionRole AWS gestita include le autorizzazioni richieste. Per applicare le autorizzazioni con privilegi minimi, puoi rimuovere la AWSLambdaVPCAccessExecutionRolepolicy 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 tramite il gateway NAT. Il collegamento di una funzione a una sottorete pubblica non le consente 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. Selezionare Salva.

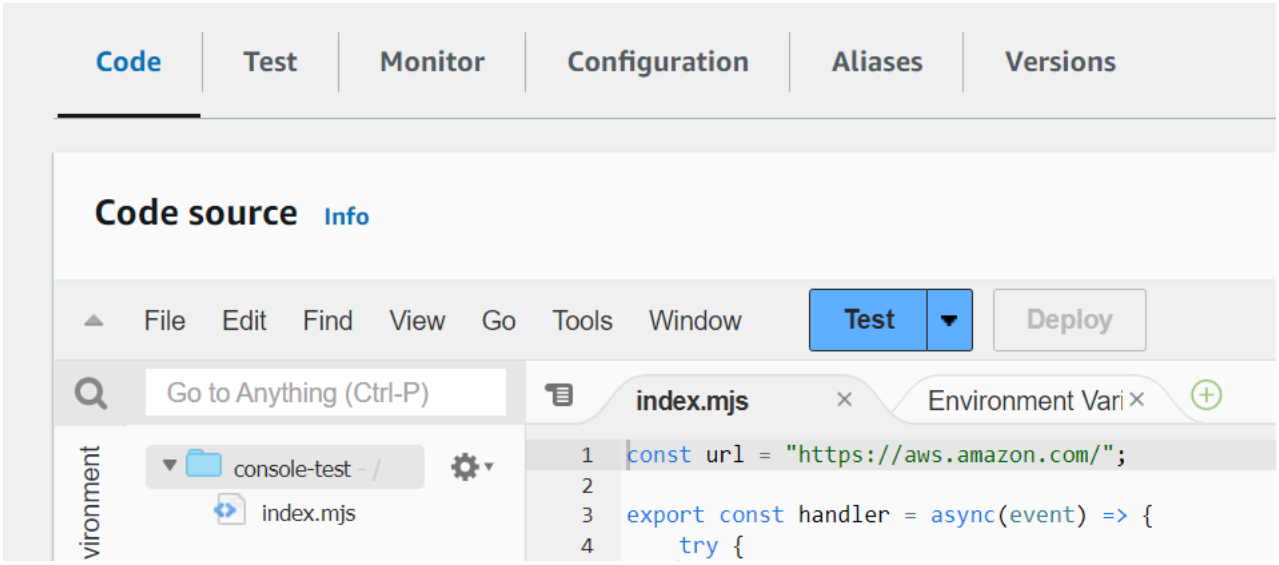
Test della funzione

Usa il seguente codice di esempio per confermare che la tua funzione connessa al VPC possa raggiungere la rete Internet pubblica. In caso di successo, il codice restituisce un `200` codice di stato. In caso di esito negativo, la funzione scade.

Node.js

Questo esempio utilizza`fetch`, che è disponibile nei runtime `nodejs18.x` e nelle versioni successive.

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.

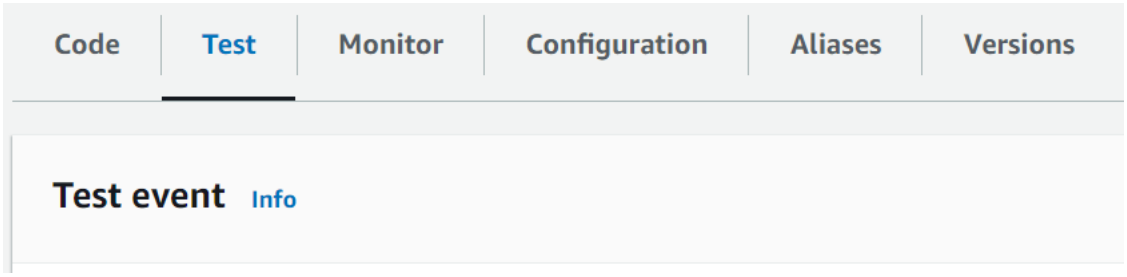


Example — Richiesta HTTP con async/await

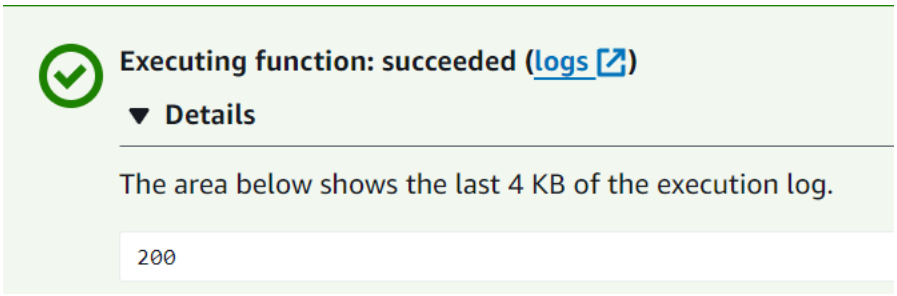
```
const url = "https://aws.amazon.com/";

export const handler = async(event) => {
    try {
        // fetch is available with Node.js 18 and later runtimes
        const res = await fetch(url);
        console.info("status", res.status);
        return res.status;
    }
    catch (e) {
        console.error(e);
```

```
        return 500;
    }
};
```

2. Seleziona Deploy (Implementa).
3. Seleziona la scheda Test.



4. Scegli Test (Esegui test).
5. La funzione restituisce un codice di `200` stato. Ciò significa che la funzione dispone di un accesso a Internet in uscita.

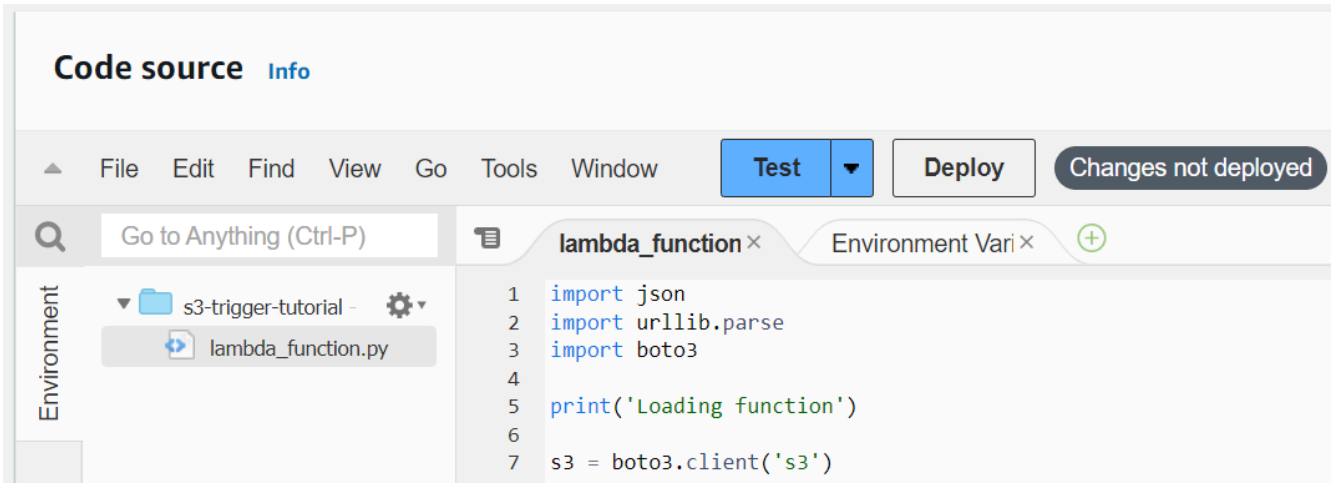


Se la funzione non riesce a raggiungere la rete Internet pubblica, viene visualizzato un messaggio di errore simile al seguente:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

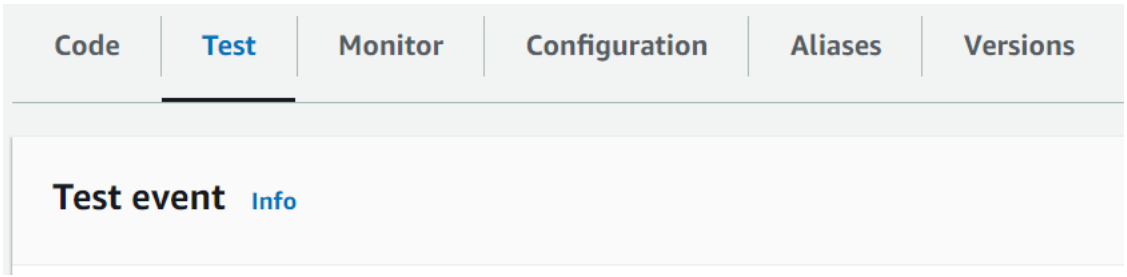
Python

1. Nel riquadro Codice sorgente della console Lambda, incolla il 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.

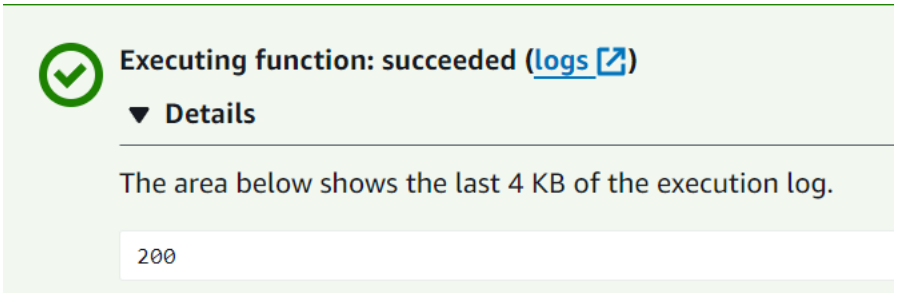



```
import urllib.request

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. Seleziona Deploy (Implementa).
3. Seleziona la scheda Test.



4. Scegli Test (Esegui test).
5. La funzione restituisce un codice di `200` stato. Ciò significa che la funzione dispone di un accesso a Internet in uscita.



Se la funzione non riesce a raggiungere la rete Internet pubblica, viene visualizzato un messaggio di errore simile al seguente:

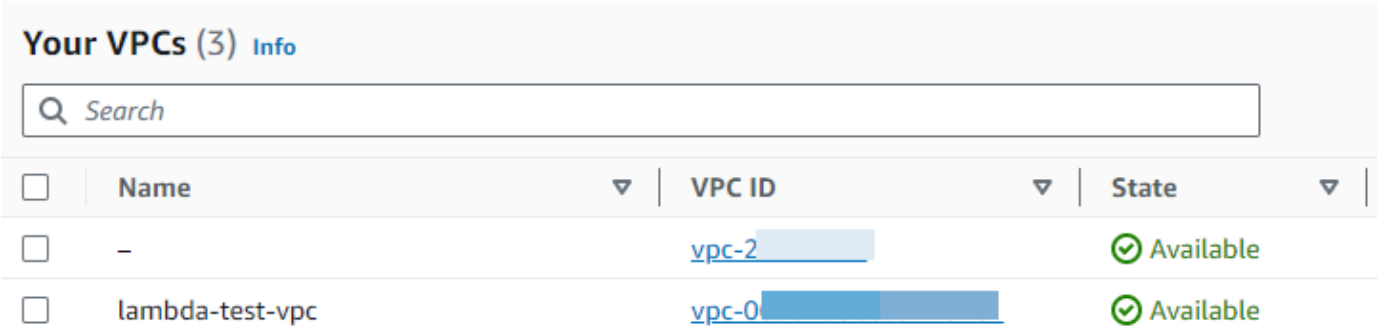
```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

Ho già un VPC

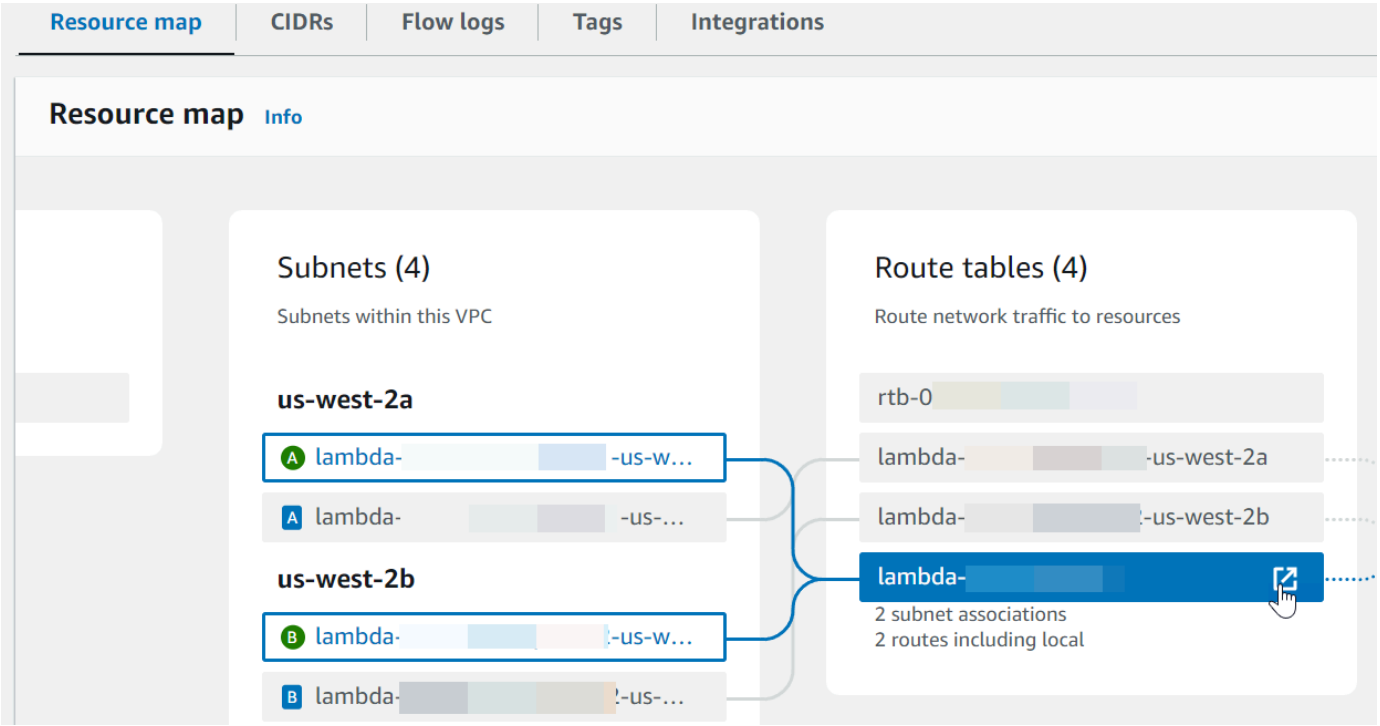
Se disponi già di un VPC ma devi configurare l'accesso pubblico a Internet per una funzione Lambda, segui questi passaggi. Questa procedura presuppone che il VPC disponga di almeno due sottoreti. Se non disponi di due sottoreti, consulta Create a subnet nella Amazon VPC User Guide.

Verifica la configurazione della tabella delle rotte

1. Apri alla console Amazon VPC all'indirizzo https://console.aws.amazon.com/vpc/.
2. Scegli l'ID VPC.



3. Scorri verso il basso fino alla sezione Mappa delle risorse. Nota le mappature della tabella dei percorsi. Apri ogni tabella di routing mappata su una sottorete.

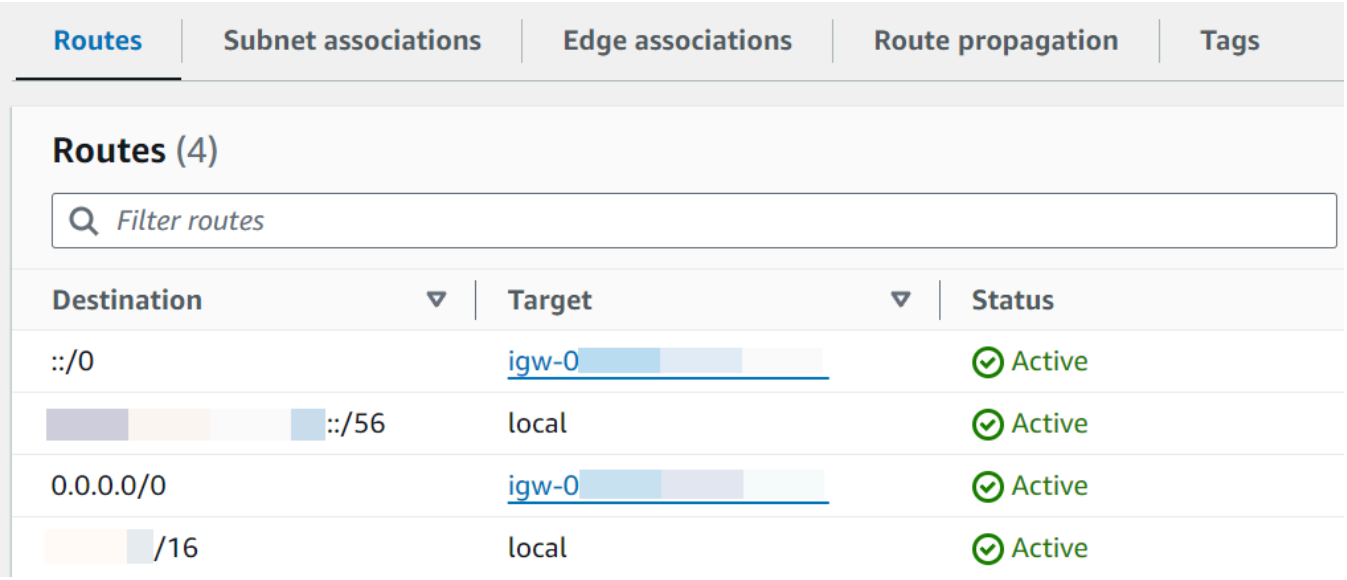


4. Scorri verso il basso fino alla scheda Percorsi. Controlla i percorsi per determinare se una delle seguenti condizioni è vera. Ciascuno di questi requisiti deve essere soddisfatto da una tabella dei percorsi separata.

   - Il traffico collegato a Internet (`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 delle rotte è una sottorete pubblica.

     > **Note**
     >
     > Se la sottorete non ha un blocco CIDR IPv6, vedrete solo la route IPv4 (). `0.0.0.0/0`

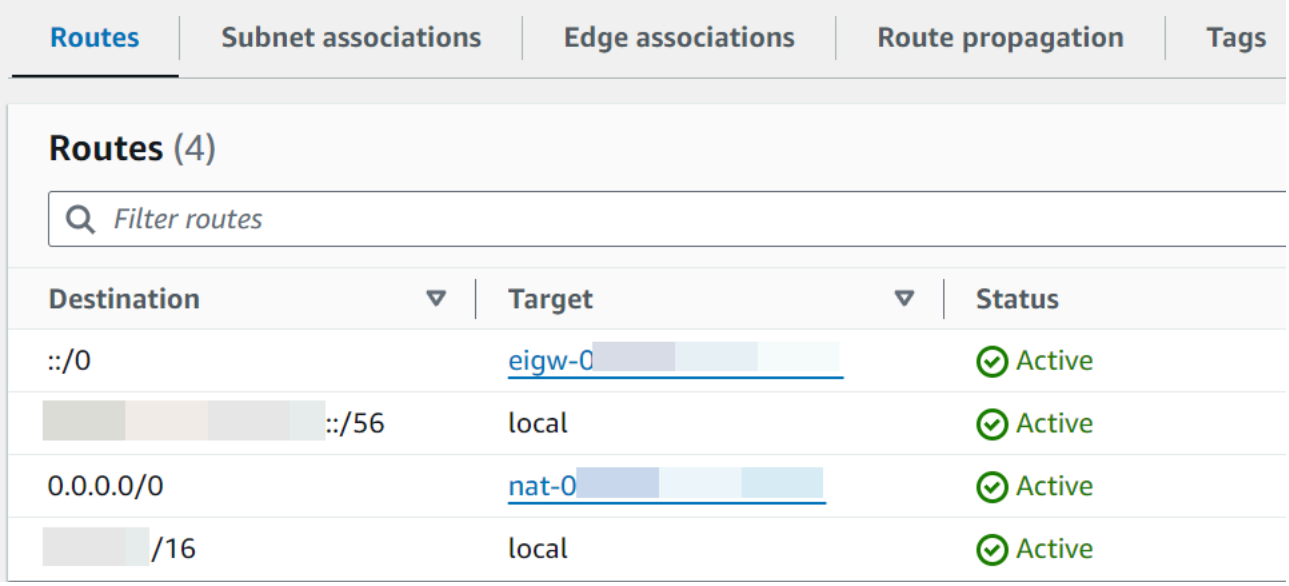
Example tabella di routing di sottorete pubblica



- Il traffico collegato a Internet 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 inoltre indirizzare il traffico IPv6 legato a Internet () verso un gateway Internet di sola uscita (). `::/0` `eigw-xxxxxxxxxx` Se la sottorete non ha un blocco CIDR IPv6, vedrete solo la route IPv4 (). `0.0.0.0/0`

Example tabella di routing di sottorete privata



5. 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 un percorso verso un gateway Internet e una con un percorso verso un gateway NAT, segui questi passaggi per creare le risorse mancanti e le voci della tabella di routing.

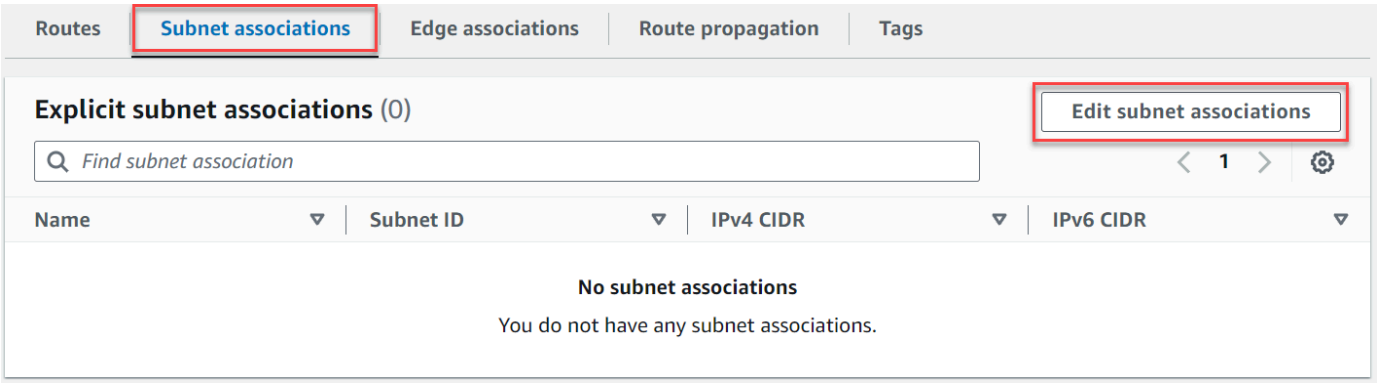
Creazione di una tabella di routing

Segui questi passaggi per creare una tabella di routing e associarla a una sottorete.

Per creare una tabella di routing personalizzata utilizzando la console Amazon VPC

1. Apri la console Amazon VPC all'indirizzo https://console.aws.amazon.com/vpc/.
2. Nel riquadro di navigazione, seleziona Tabelle di routing.
3. Selezionare Create route table (Crea tabella di instradamento).
4. (Facoltativo) In Name (Nome), inserisci un nome per la tabella di instradamento.
5. In VPC, seleziona il VPC.
6. (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.



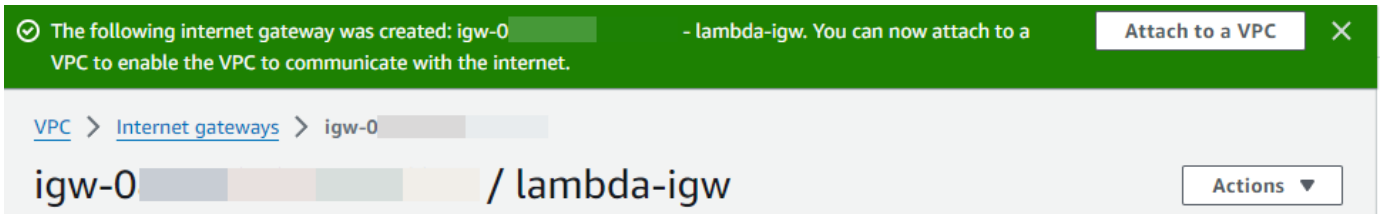
9. Seleziona la casella di controllo per la sottorete da associare alla tabella di instradamento.
10. Scegli Salva associazioni.

Creazione di un Internet Gateway

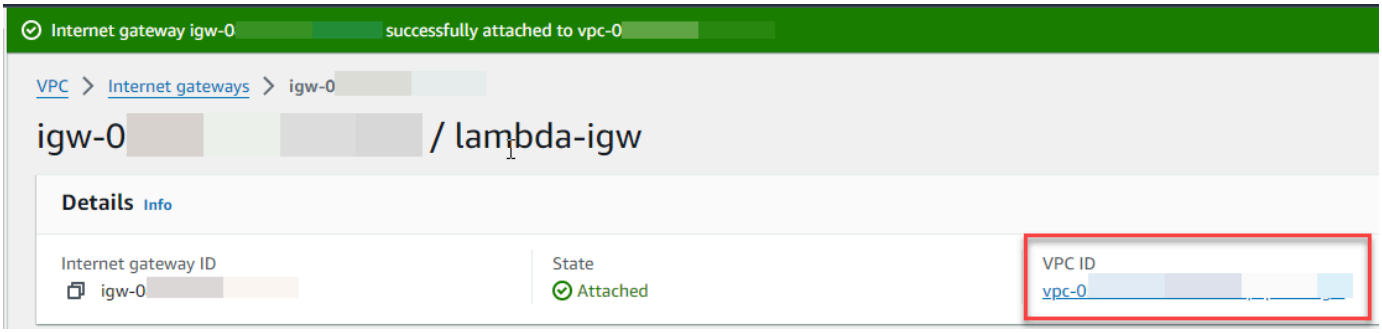
Segui questi passaggi per creare un gateway Internet, collegarlo al tuo VPC e aggiungerlo alla tabella di routing della sottorete pubblica.

Creare un gateway Internet

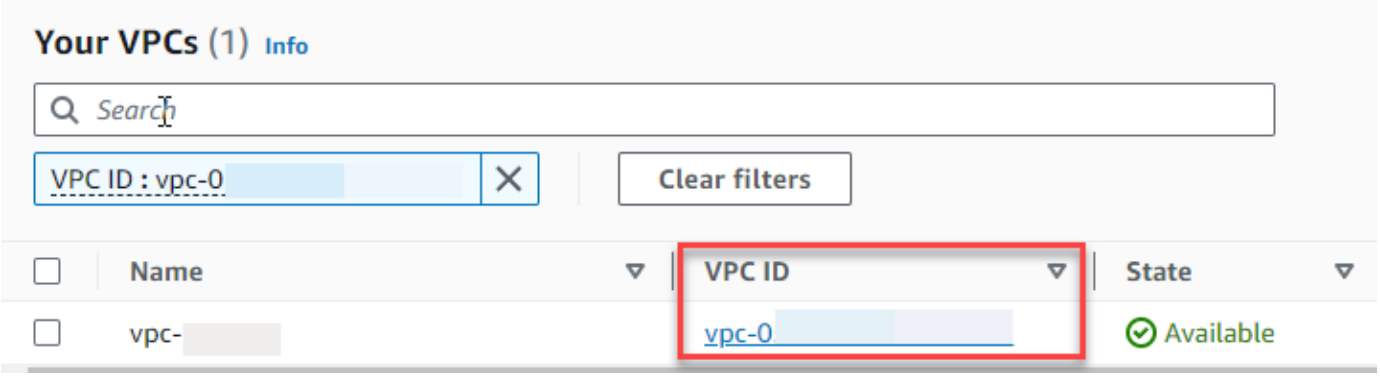
1. Accedere alla console Amazon VPC all'indirizzo https://console.aws.amazon.com/vpc/.
2. Nel pannello di navigazione, scegli Internet gateways (Gateway Internet).
3. Scegliere Crea gateway Internet.
4. (Facoltativo) Inserisci un nome per il gateway Internet.
5. (Facoltativo) Per aggiungere un tag, scegli 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, quindi scegli Collega gateway internet.



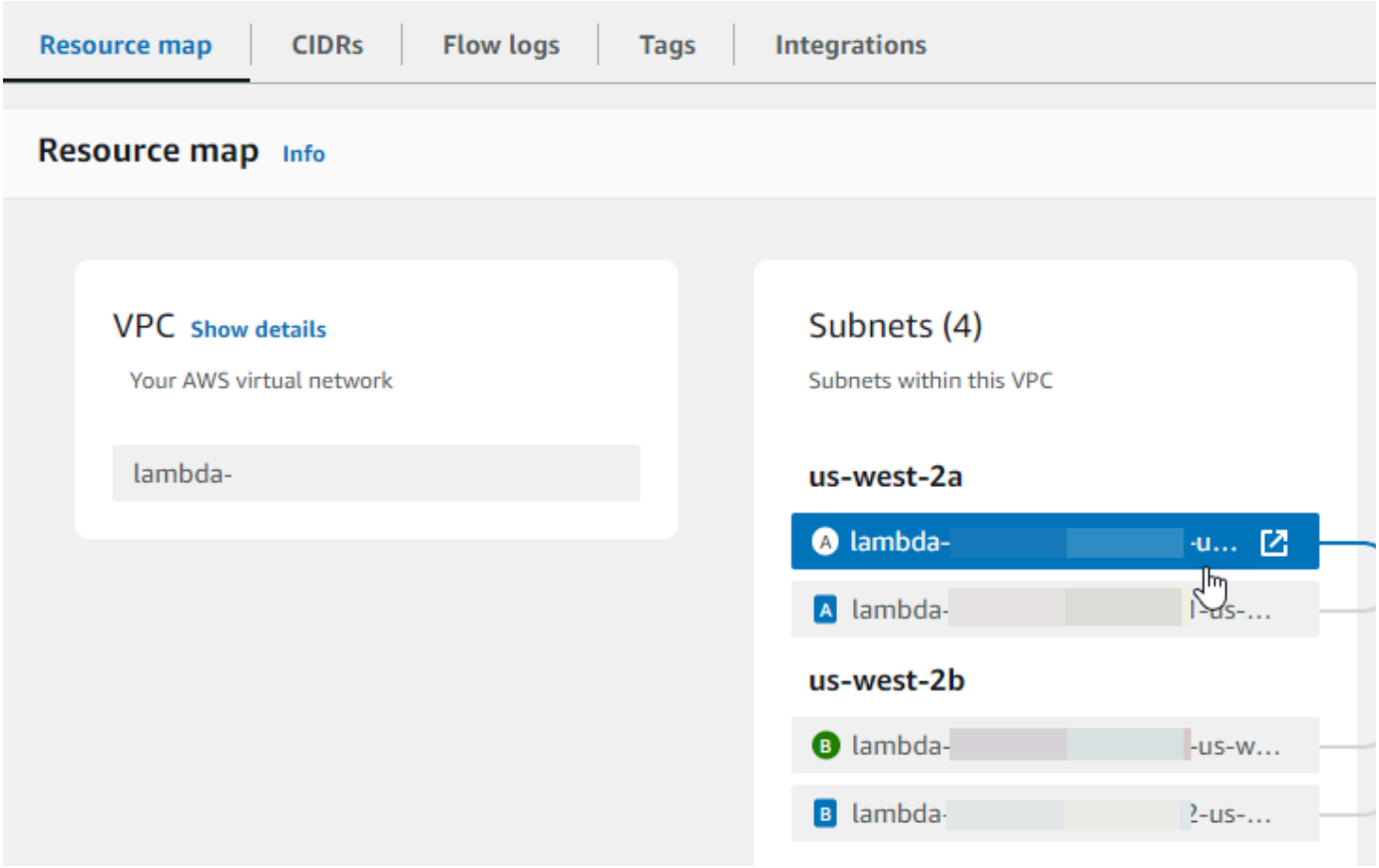
8. Scegli l'ID VPC.



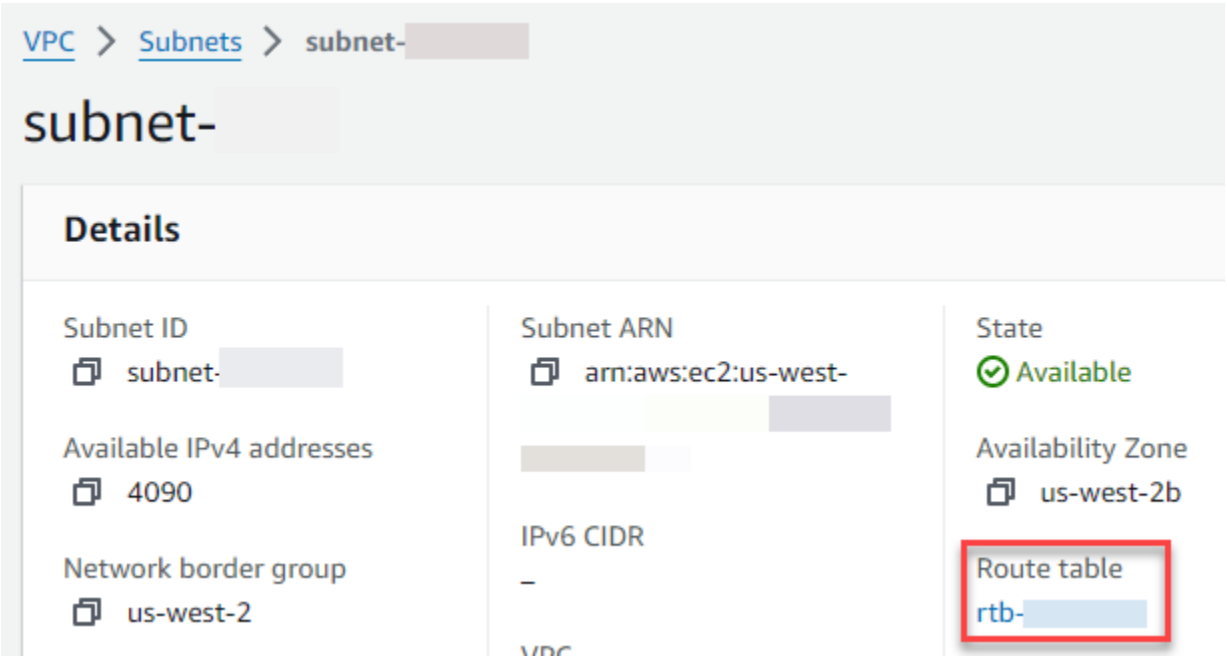
9. Scegli nuovamente l'ID VPC per aprire la pagina dei dettagli del VPC.



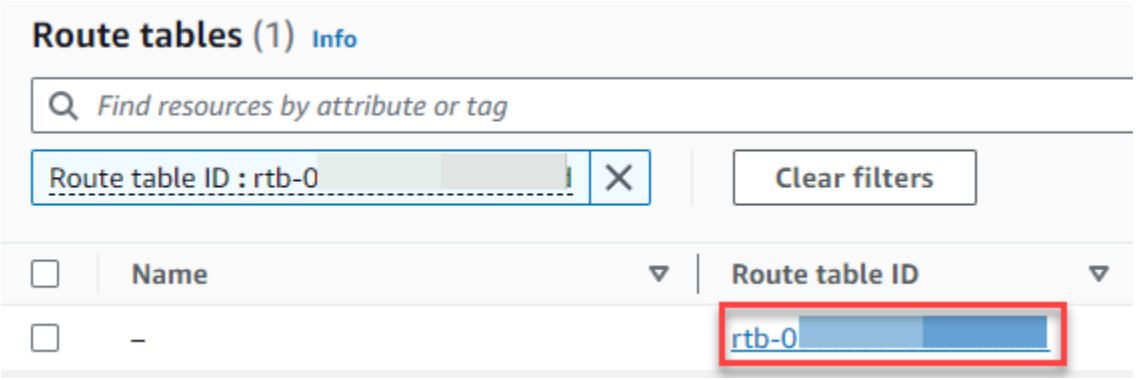
10. Scorri verso il basso fino alla sezione Mappa delle risorse, quindi scegli una sottorete. I dettagli della sottorete vengono visualizzati in una nuova scheda.



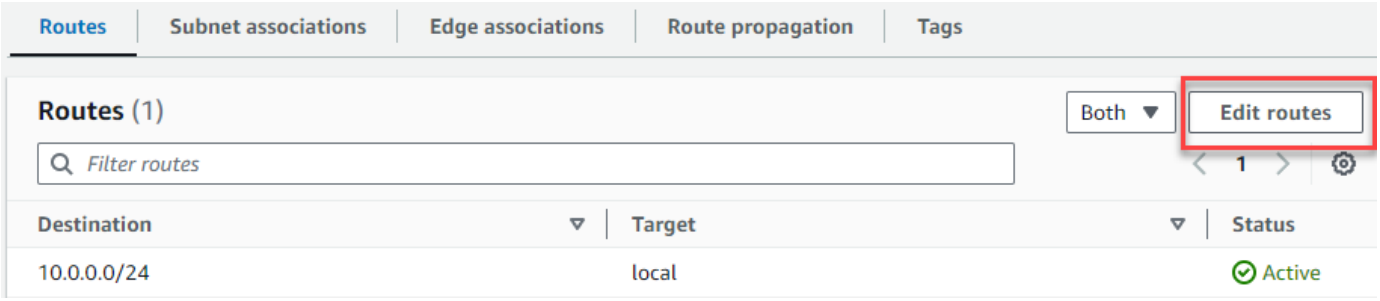
11. Scegli il link nella tabella delle rotte.



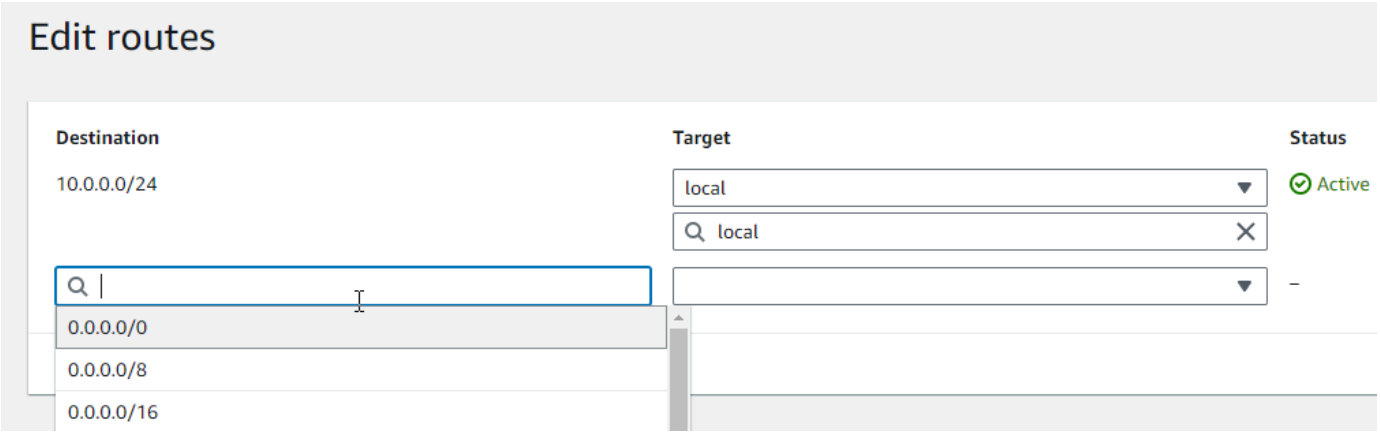
12. Scegli l'ID della tabella delle rotte per aprire la pagina dei dettagli della tabella delle rotte.



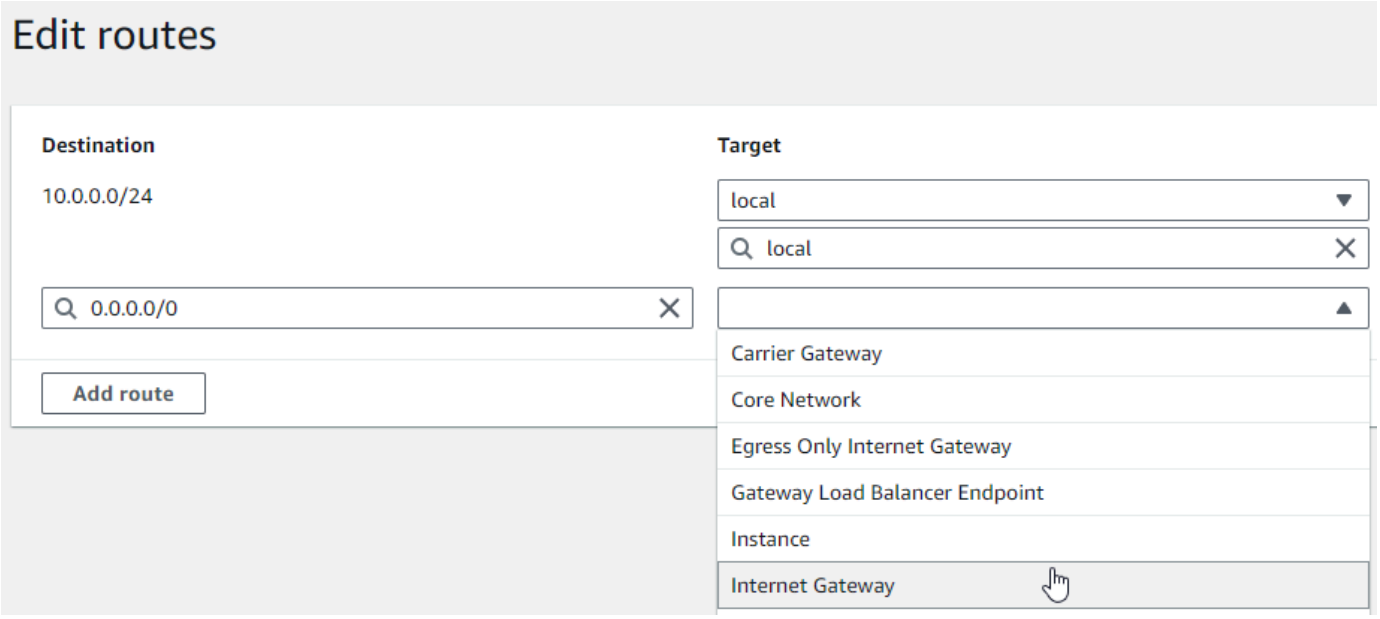
13. In Percorsi, scegli Modifica percorsi.



14. Scegli Aggiungi percorso, quindi inserisci `0.0.0.0/0` nella casella Destinazione.



15. Per Target, seleziona Internet gateway, quindi scegli il gateway Internet creato in precedenza. Se la sottorete ha un blocco CIDR IPv6, devi anche aggiungere un percorso `::/0` per lo stesso gateway Internet.



16. Seleziona Salvataggio delle 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

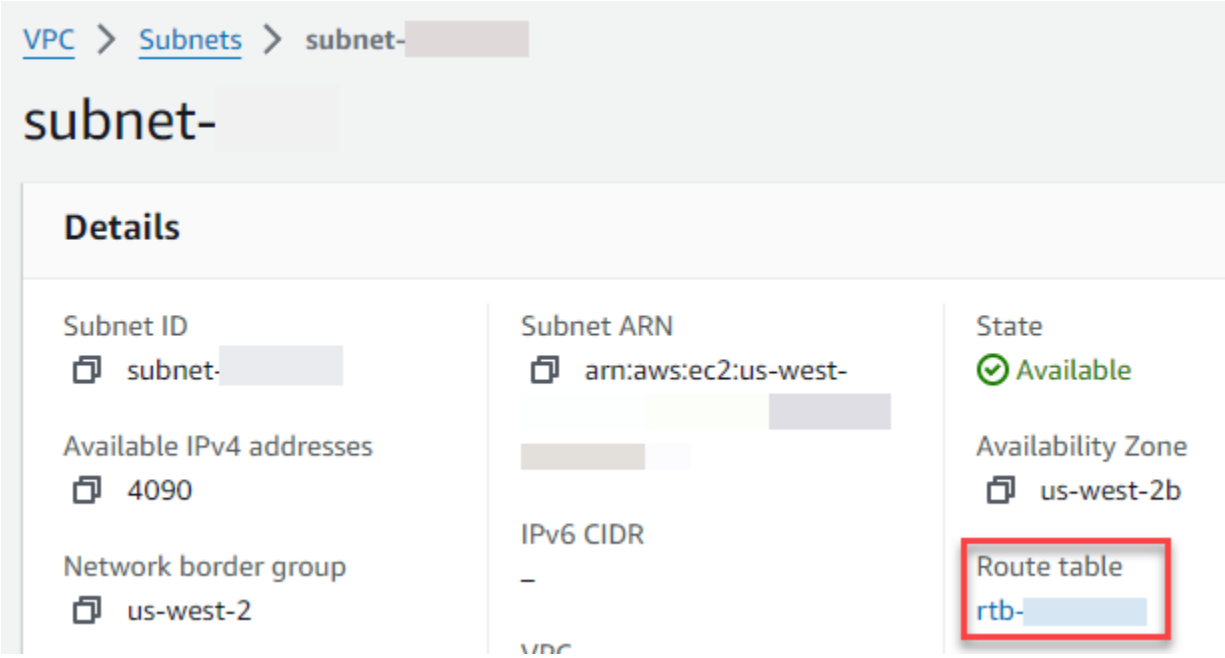
1. Nel riquadro di navigazione, scegli Gateway NAT.
2. Scegli Crea gateway NAT.
3. (Facoltativo) Inserisci un nome per il tuo gateway NAT.
4. Per Subnet, seleziona una sottorete pubblica nel tuo VPC. (Una sottorete pubblica è una sottorete che ha un percorso diretto verso un gateway Internet nella tabella delle rotte.)

> **Note**
>
> I gateway NAT sono associati a una sottorete pubblica, ma la voce della tabella di routing si trova nella sottorete privata.

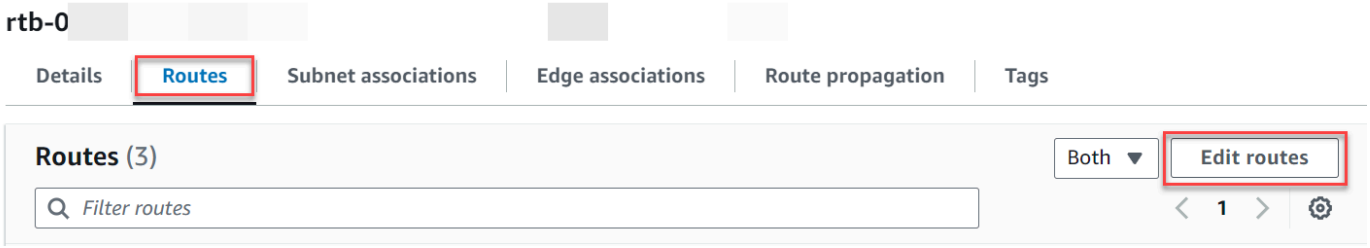
5. Per Elastic IP allocation ID, seleziona un indirizzo IP elastico o scegli Allocate Elastic IP.
6. Scegli Crea gateway NAT.

Per aggiungere una route al gateway NAT nella tabella di routing della sottorete privata

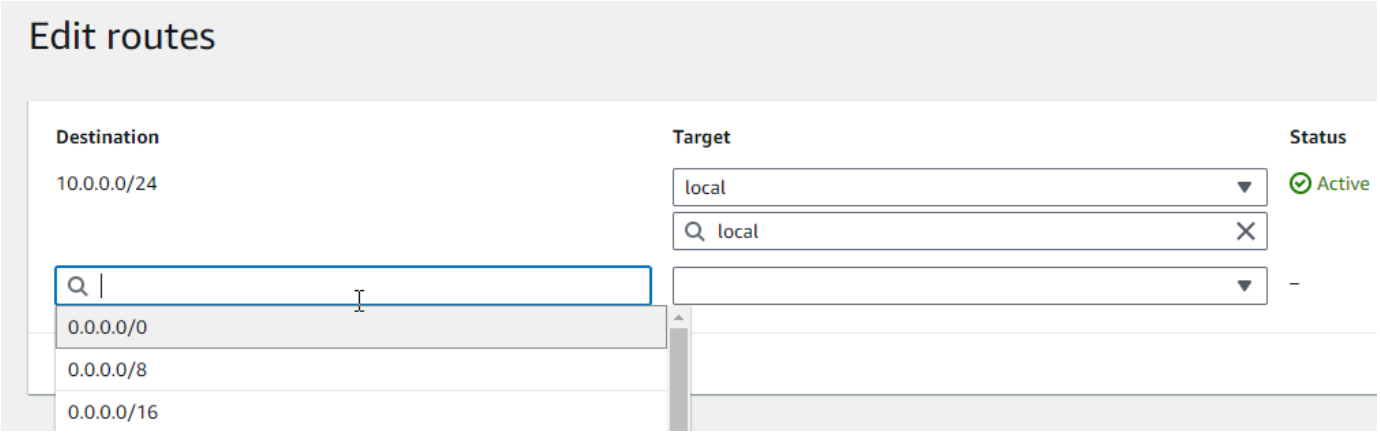
1. Nel pannello di navigazione, scegli Subnets (Sottoreti).
2. Seleziona una sottorete privata nel tuo VPC. (Una sottorete privata è una sottorete che non ha un percorso verso un gateway Internet nella tabella delle rotte.)
3. Scegli il link nella tabella delle rotte.



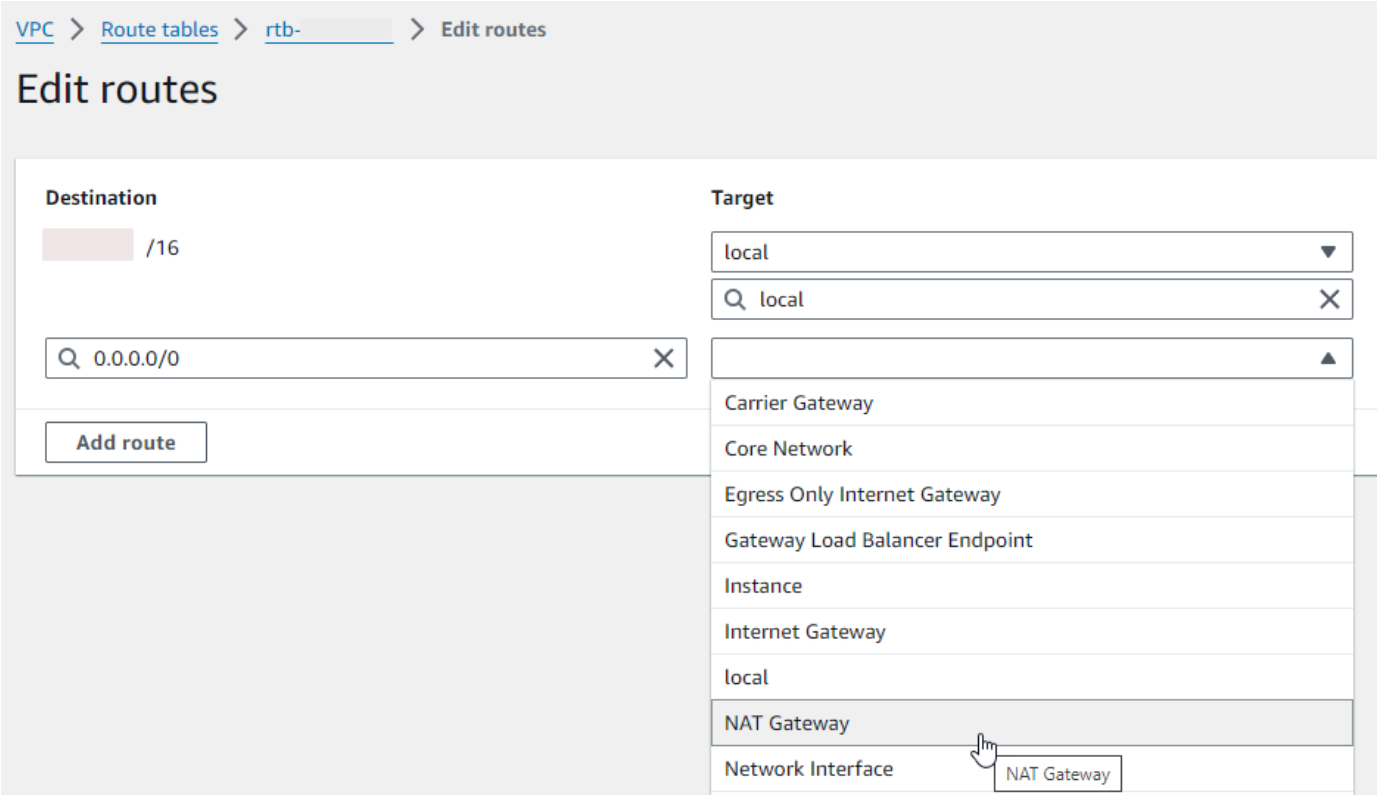
4. Scorri verso il basso e scegli la scheda Percorsi, quindi scegli Modifica percorsi



5. Scegli Aggiungi percorso, quindi inserisci `0.0.0.0/0` nella casella Destinazione.



6. Per Target, seleziona NAT gateway, quindi scegli il gateway NAT che hai creato in precedenza.



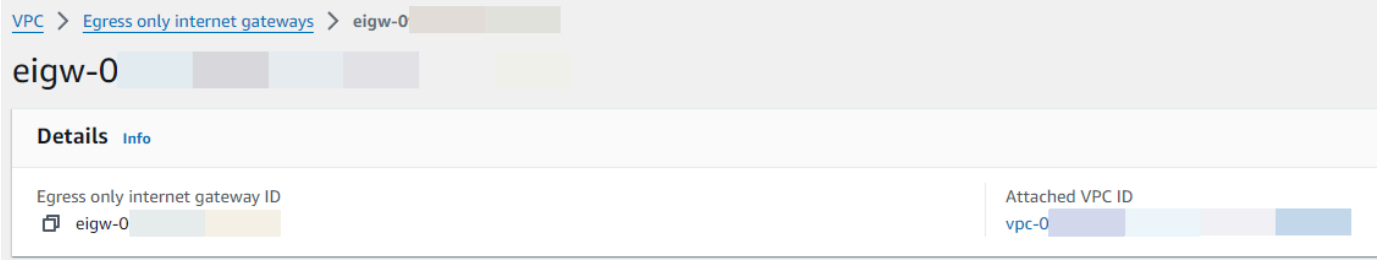
7. Seleziona Salvataggio delle modifiche.

Crea un gateway Internet solo in uscita (solo IPv6)

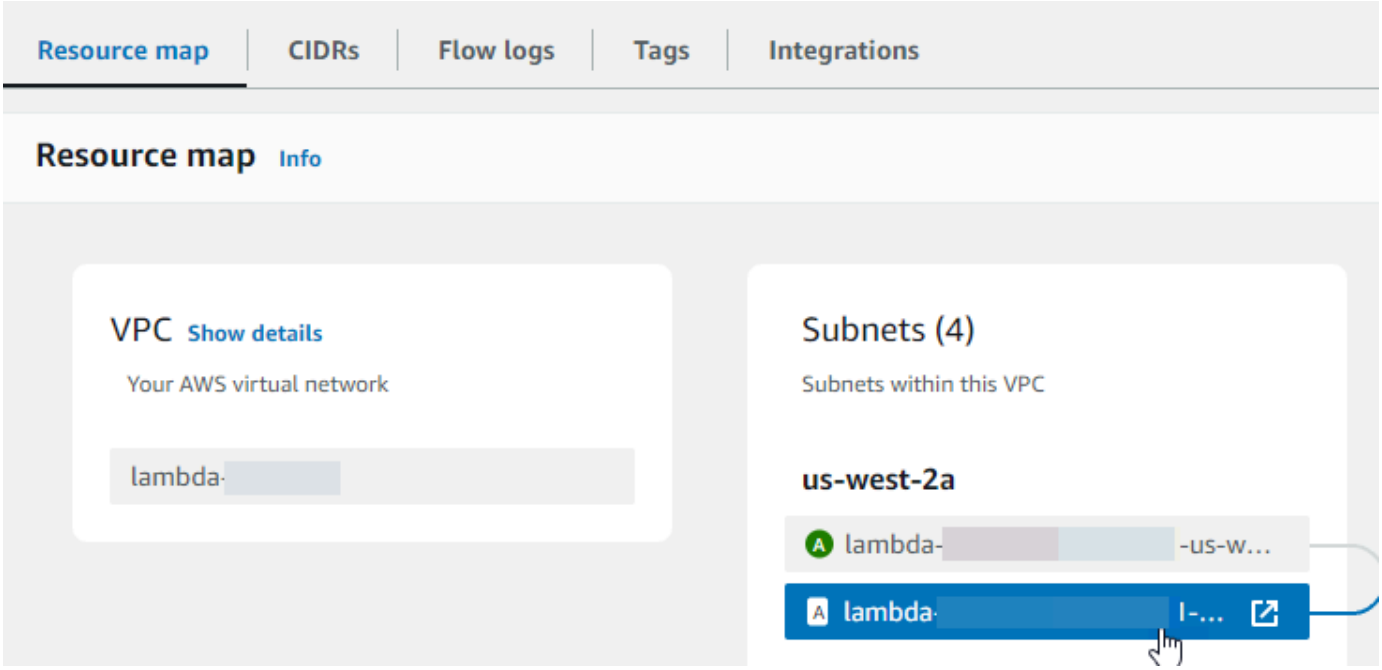
Segui questi passaggi per creare un gateway Internet solo in uscita e aggiungerlo alla tabella di routing della tua sottorete privata.

Per creare un gateway internet egress-only

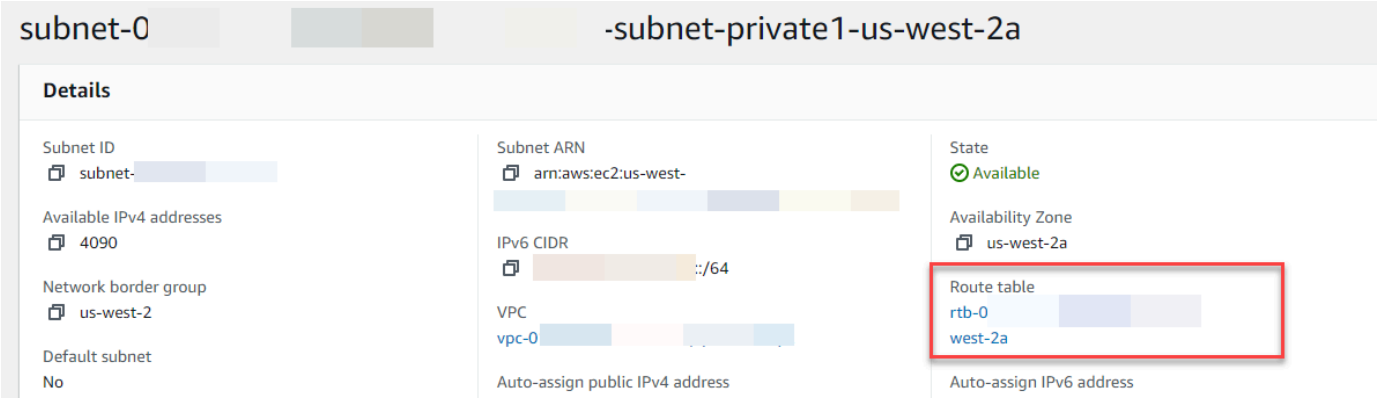
1. Nel riquadro di navigazione, seleziona Gateway Internet solo in uscita.
2. Seleziona Crea gateway Internet solo in uscita.
3. (Facoltativo) Inserisci un nome.
4. Selezionare il VPC nel quale creare l'Internet Gateway egress-only.
5. Seleziona Crea gateway Internet solo in uscita.
6. Scegli il link sotto Allegato VPC ID.



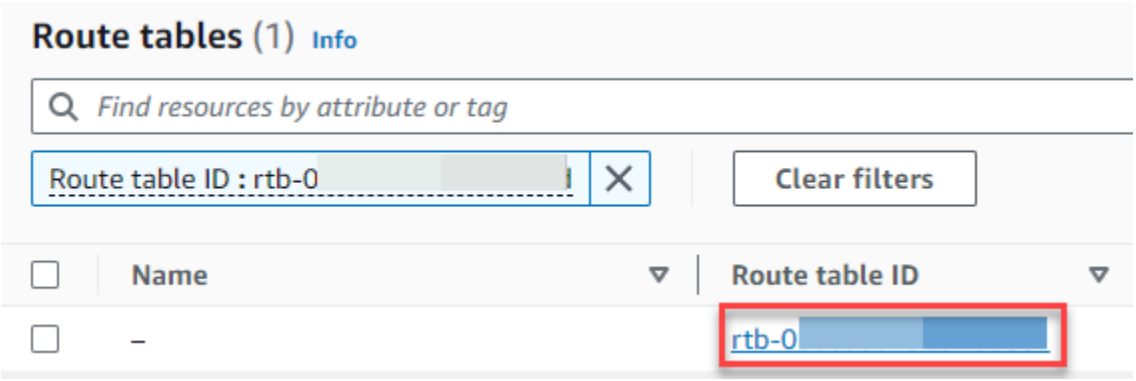
7. Scegli il link sotto VPC ID per aprire la pagina dei dettagli del VPC.
8. Scorri verso il basso fino alla sezione Mappa delle risorse, quindi scegli una sottorete privata. I dettagli della sottorete vengono visualizzati in una nuova scheda.



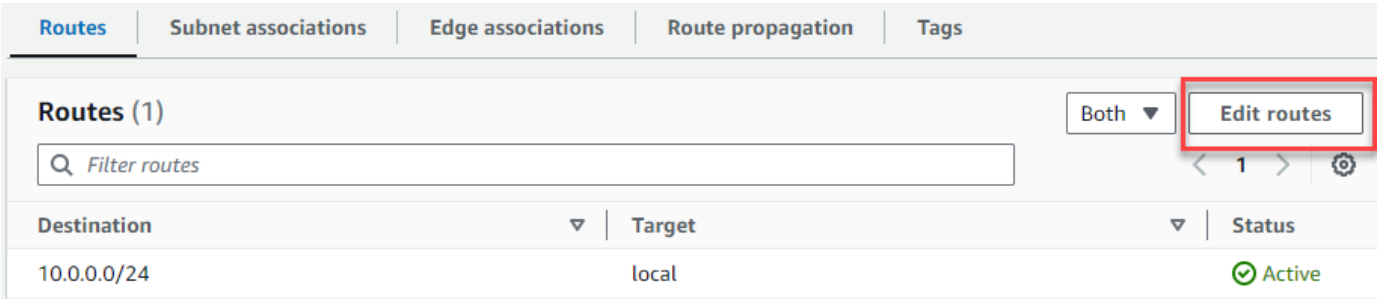
9. Scegli il link nella tabella delle rotte.



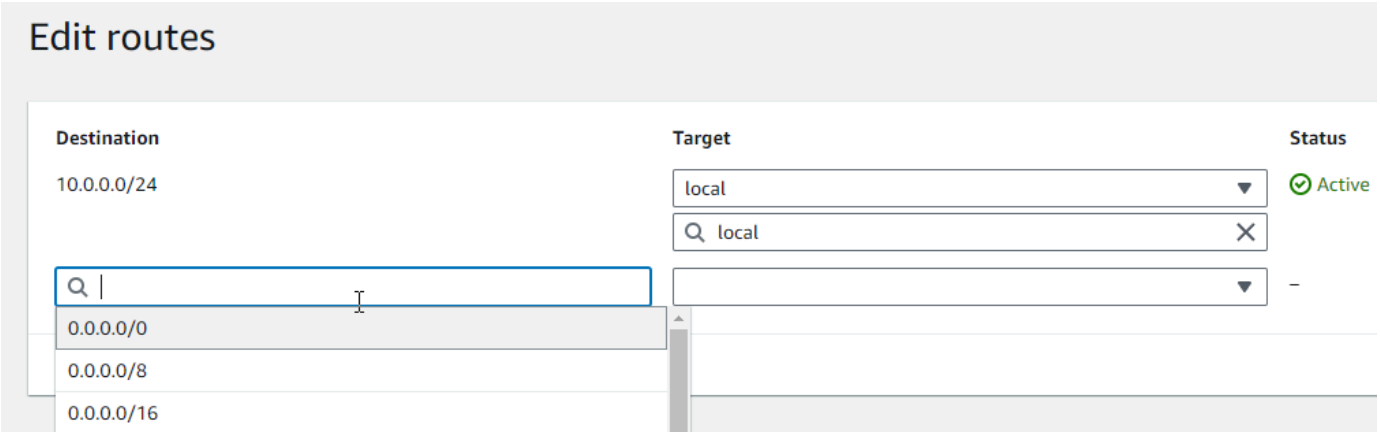
10. Scegli l'ID della tabella delle rotte per aprire la pagina dei dettagli della tabella delle rotte.



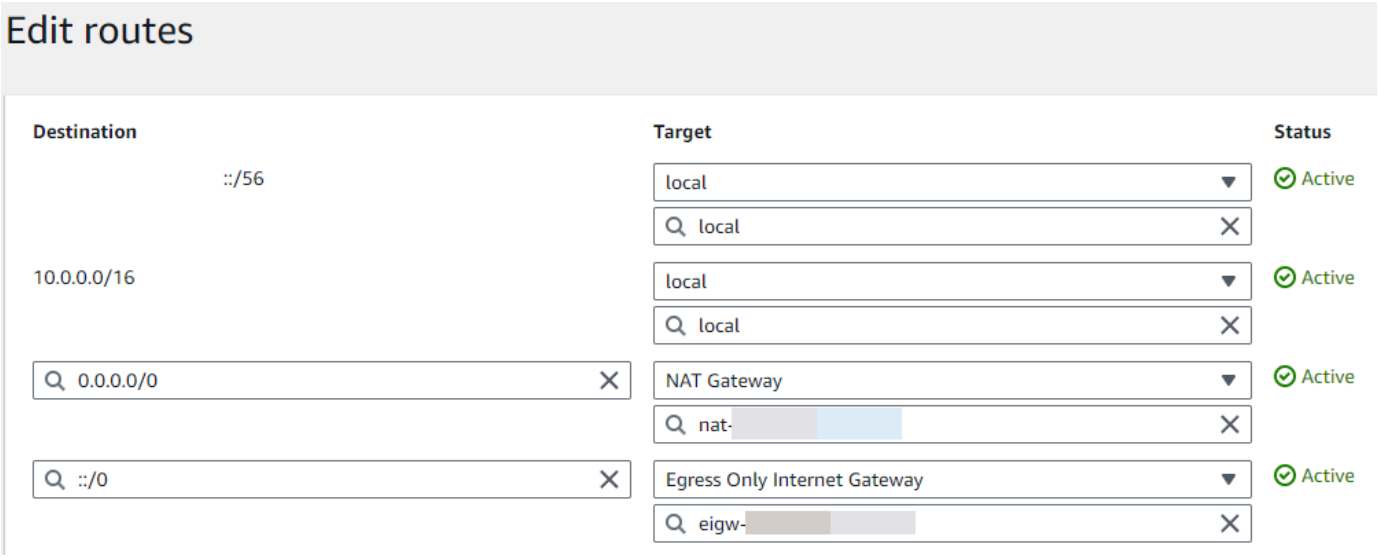
11. In Percorsi, scegli Modifica percorsi.



12. Scegli Aggiungi percorso, quindi inserisci `::/0` nella casella Destinazione.



13. Per Target, seleziona Gateway Internet Only Egress, quindi scegli il gateway creato in precedenza.



14. Seleziona Salvataggio delle 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 tramite il gateway NAT. Il collegamento di una funzione a una sottorete pubblica non le consente 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 AWSLambdaVPCAccessExecutionRole AWS gestita. Le autorizzazioni in questa politica sono necessarie solo per creare interfacce di rete elastiche per la configurazione VPC, non per richiamare la tua funzione. Per applicare le autorizzazioni con privilegi minimi, puoi rimuovere la AWSLambdaVPCAccessExecutionRolepolicy 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 politica AWSLambdaVPCAccessExecutionRole AWS gestita include le autorizzazioni richieste. Per applicare le autorizzazioni con privilegi minimi, puoi rimuovere la AWSLambdaVPCAccessExecutionRolepolicy 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 tramite il gateway NAT. Il collegamento di una funzione a una sottorete pubblica non le consente 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. Selezionare Salva.

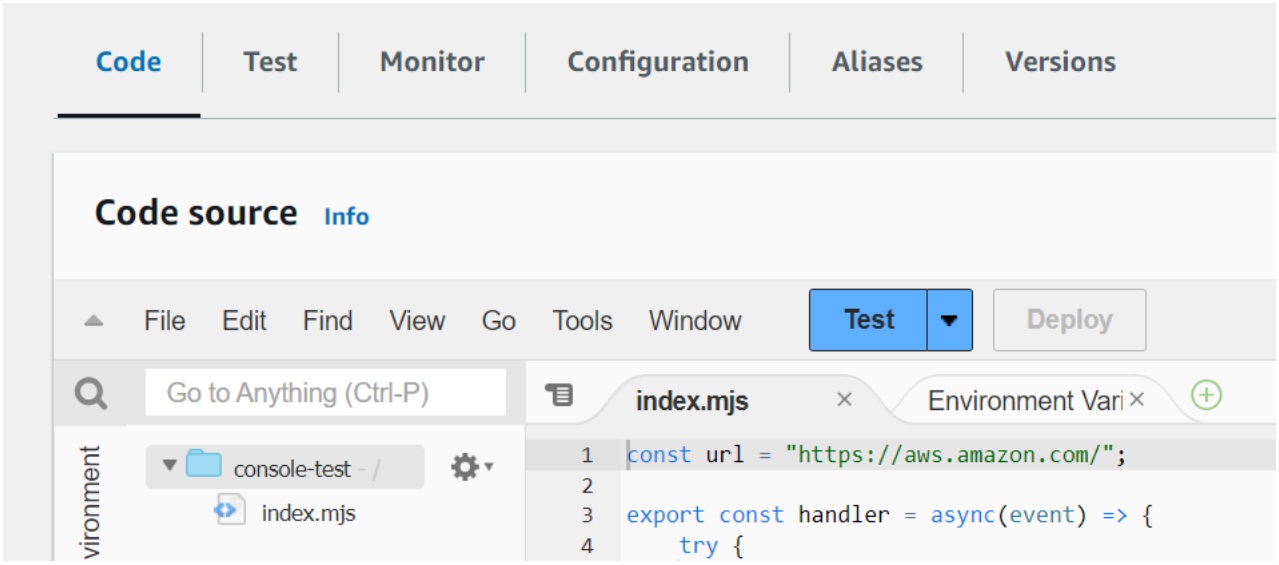
Test della funzione

Usa il seguente codice di esempio per confermare che la tua funzione connessa al VPC possa raggiungere la rete Internet pubblica. In caso di successo, il codice restituisce un `200` codice di stato. In caso di esito negativo, la funzione scade.

Node.js

Questo esempio utilizza`fetch`, che è disponibile nei runtime `nodejs18.x` e nelle versioni successive.

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.

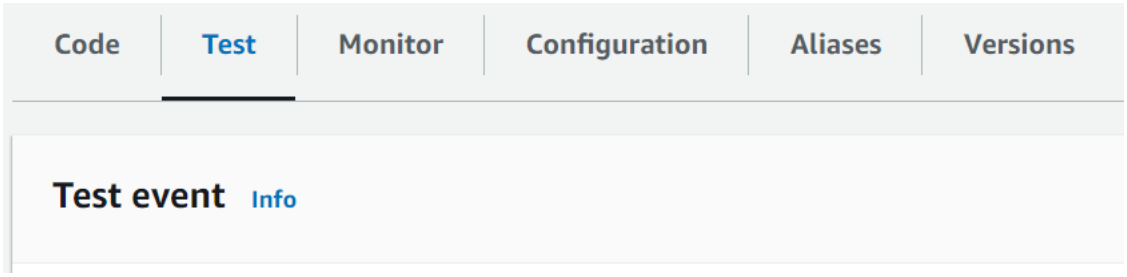


Example — Richiesta HTTP con async/await

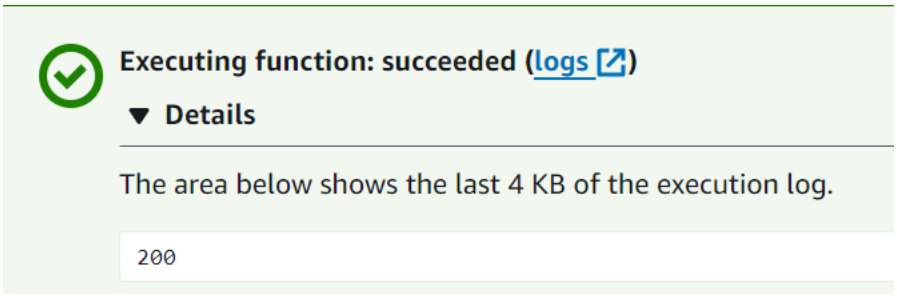
```
const url = "https://aws.amazon.com/";

export const handler = async(event) => {
    try {
        // fetch is available with Node.js 18 and later runtimes
        const res = await fetch(url);
        console.info("status", res.status);
        return res.status;
    }
    catch (e) {
        console.error(e);
```

```
        return 500;
    }
};
```

2. Seleziona Deploy (Implementa).
3. Seleziona la scheda Test.



4. Scegli Test (Esegui test).
5. La funzione restituisce un codice di `200` stato. Ciò significa che la funzione dispone di un accesso a Internet in uscita.

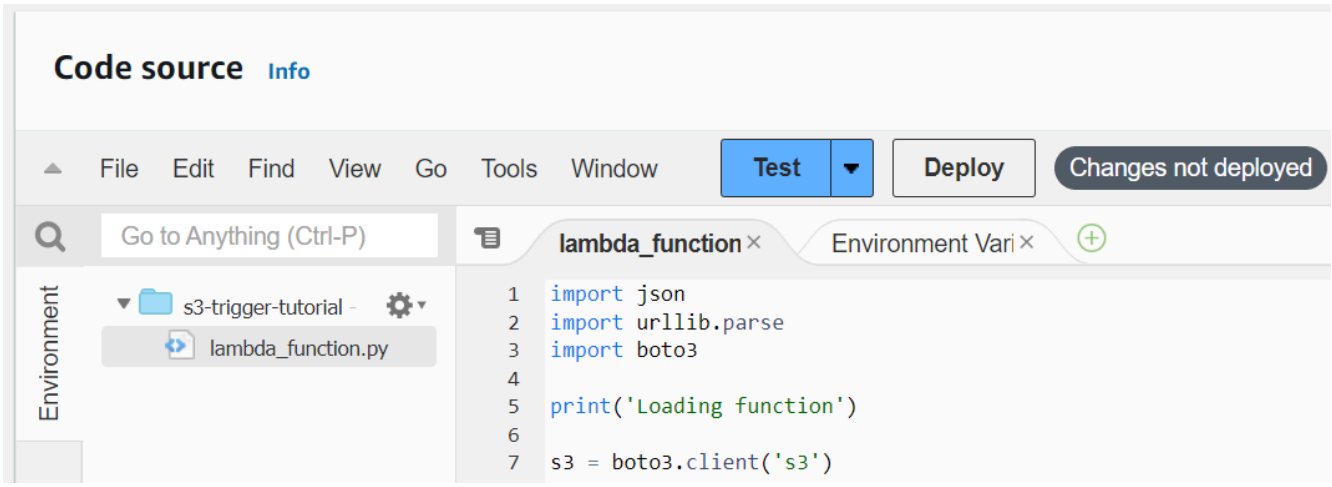


Se la funzione non riesce a raggiungere la rete Internet pubblica, viene visualizzato un messaggio di errore simile al seguente:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

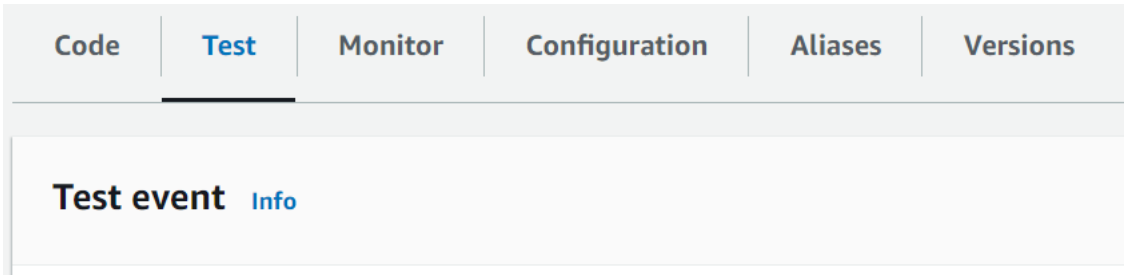
Python

1. Nel riquadro Codice sorgente della console Lambda, incolla il 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.

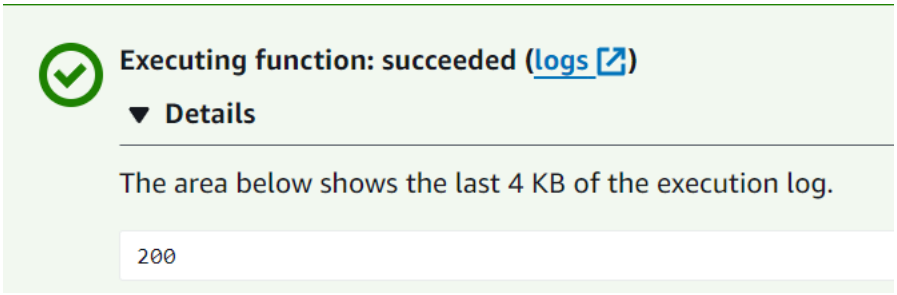



```
import urllib.request

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. Seleziona Deploy (Implementa).
3. Seleziona la scheda Test.



4. Scegli Test (Esegui test).
5. La funzione restituisce un codice di `200` stato. Ciò significa che la funzione dispone di un accesso a Internet in uscita.



Se la funzione non riesce a raggiungere la rete Internet pubblica, viene visualizzato un messaggio di errore simile al seguente:

```
{
  "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. Gli endpoint di interfaccia sono alimentati da AWS PrivateLink, il che consente di accedere in modo privato 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

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 persistenti

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, vedere Riutilizzo delle connessioni con Keep-Alive in Node.js nella Guida per gli sviluppatori di AWS SDK for JavaScript .

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 AWS PrivateLink si applica agli endpoint di interfaccia per Lambda. All' AWS account viene fatturata 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 peer rimane sulla AWS rete e non attraversa la rete Internet pubblica. Una volta eseguito il peering dei vPC, risorse come istanze Amazon Elastic Compute Cloud (Amazon EC2), istanze Amazon Relational Database Service (Amazon RDS) o funzioni Lambda abilitate per VPC in entrambi i VPC possono accedere all'API Lambda tramite gli 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, consulta Creazione di un endpoint dell'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 di AWS .
4. Per Nome servizio, 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 i nomi DNS privati per i tuoi 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 sull'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, consulta Accesso a un servizio tramite un endpoint dell'interfaccia in Guida per l'utente di Amazon VPC.

Per informazioni sulla creazione e configurazione di un endpoint utilizzando AWS CloudFormation, consulta la risorsa AWS: :EC2: :VPCEndpoint nella Guida per l'utente.AWS CloudFormation

Per creare un endpoint di interfaccia per Lambda (AWS CLI)

Utilizzare il comando `create-vpc-endpoint` e specificare l'ID VPC, il tipo di endpoint VPC (interfaccia), il nome del servizio, le sottoreti per utilizzare 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:*"
            ]
      }
   ]
```

```
}
```

Configurazione dell'accesso al file system per le funzioni Lambda

È possibile configurare una funzione per montare un file system Amazon Elastic File System (Amazon EFS) in una directory locale. 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

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

- elasticfilesystem: obiettivi DescribeMount

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 Offrire alle funzioni Lambda l'accesso alle risorse in un Amazon VPC.

Configurazione dell'accesso al file system

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. 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 di dati Amazon EC2 tra VPC nella stessa regione. Per informazioni dettagliate, consulta Prezzi di Lambda.

Per ulteriori informazioni sull'integrazione di Lambda con Amazon EFS, consulta Utilizzo di Amazon EFS con Lambda.

Utilizzo di un file system Amazon EFS in un altro Account AWS per una funzione Lambda

Puoi configurare una funzione per montare un file system Amazon EFS in un altro Account AWS. Prima di montare il file system, accertati che:

- Deve essere configurato il peering VPC e devono essere aggiunti i percorsi appropriati alle tabelle di routing in ogni VPC.
- Il gruppo di sicurezza per il file system Amazon EFS da montare deve essere configurato per consentire l'accesso in entrata dal gruppo di sicurezza associato alla tua funzione Lambda.
- Le sottoreti devono essere create in ogni VPC con ID delle zone di disponibilità (AZ) corrispondenti.
- I nomi host DNS devono essere abilitati in entrambi i VPC.

Affinché la tua funzione Lambda possa accedere a un file system Amazon EFS in un altro Account AWS, tale file system deve avere anche una policy di file system che conceda l'autorizzazione alla tua funzione. Per informazioni sulla creazione di una policy del file system, consulta Creazione di policy del file system nella Guida per l'utente del file system Amazon Elastic.

Di seguito è illustrato un esempio di policy che autorizza le funzioni Lambda nell'autorizzazione di un account specificato per l'esecuzione di tutte le azioni dell'API su un file system.

```
{
    "Version": "2012-10-17",
    "Id": "efs-lambda-policy",
    "Statement": [
        {
            "Sid": "efs-lambda-statement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::{LAMBDA-ACCOUNT-ID}:root"
            },
```

```
            "Action": "*",
            "Resource": "arn:aws:elasticfilesystem:{REGION}:{ACCOUNT-ID}:file-
system/{FILE SYSTEM ID}"
        }
    ]
}
```

> **Note**
>
> La politica di esempio mostrata utilizza il carattere jolly («*») per concedere le autorizzazioni per le funzioni Lambda nel campo Account AWS specificato per eseguire qualsiasi operazione API sul file system. Ciò include l'eliminazione del file system. Per limitare le operazioni che altri Account AWS possono eseguire sul tuo filesystem, specifica le azioni che desideri consentire esplicitamente. Per un elenco delle possibili operazioni dell'API, consulta Azioni, risorse e chiavi di condizione per il file system Amazon Elastic.

Per configurare il montaggio del file system su più account, utilizzate l'operazione (). AWS Command Line Interface AWS CLI`update-function-configuration`

Per montare un file system in un altro Account AWS, eseguite il comando seguente. Usa il nome della funzione e sostituisci il nome della risorsa Amazon (ARN) con l'ARN del punto di accesso Amazon EFS per il file system da montare. `LocalMountPath` è il percorso in cui la funzione può accedere al file system, a partire da `/mnt/`. Accertati che il percorso di montaggio di Lambda corrisponda al percorso del punto di accesso per il file system. Ad esempio, se il punto di accesso è `/efs`, il percorso di montaggio di Lambda deve essere `/mnt/efs`.

```
aws lambda update-function-configuration --function-name MyFunction \
--file-system-configs Arn=arn:aws:elasticfilesystem:us-east-1:222222222222:access-
point/fsap-01234567,LocalMountPath=/mnt/test
```

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.

Sections

Creazione di un alias di funzione (Console)

È possibile creare un alias di funzione utilizzando la console Lambda.

Per creare un alias

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 Configurazione del routing dell'alias.
    e. Seleziona Save (Salva.

Gestione degli alias con l'API Lambda

Per creare un alias usando il AWS Command Line Interface (AWS CLI), usa il comando. `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

Gestione degli alias con e AWS SAMAWS CloudFormation

È possibile creare e gestire alias di funzioni utilizzando AWS Serverless Application Model (AWS SAM) e. AWS CloudFormation

Per vedere come dichiarare un alias di funzione in un AWS SAM modello, consultate la pagina AWS: :Serverless: :Function nella Developer Guide. AWS SAM Per informazioni sulla creazione e configurazione degli alias utilizzando AWS CloudFormation, vedere AWS: :Lambda: :Alias nella Guida per l'utente. AWS CloudFormation

Utilizzo di alias

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 fonti di eventi basate su stream 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`), l'autorizzazione consente di richiamare la funzione `helloworld` utilizzando l'ARN dell'alias (`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 `helloworld` quando Amazon S3 agisce per conto di. `DOC-EXAMPLE-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:::DOC-EXAMPLE-BUCKET --source-account 123456789012
```

Per ulteriori informazioni sull'utilizzo dei nomi delle risorse nelle policy, consulta Ottimizzazione delle sezioni Risorse e Condizioni delle politiche.

Configurazione del routing dell'alias

Utilizza la configurazione di routing in un alias per inviare una parte del traffico a una seconda versione della funzione. Ad esempio, è possibile ridurre il rischio di distribuzione di una nuova

versione configurando l'alias per inviare la maggior parte del traffico alla versione esistente e solo una piccola percentuale di traffico alla nuova versione.

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 varianza 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.

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`.

Per configurare il routing in un alias

> **Note**
>
> Verificare che la funzione abbia almeno due versioni pubblicate. Per creare versioni aggiuntive, seguire le istruzioni in Versioni delle funzioni Lambda.

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 percento per Additional version (Versione aggiuntiva), alla prima versione viene automaticamente assegnato il 90 percento.

g. Selezionare Salva.

Configurazione del routing dell'alias mediante CLI

Utilizza i comandi `update-alias` AWS CLI e `create-alias` 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 percento 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 invocata

Quando configuri i pesi del traffico tra due versioni di funzioni, esistono due modi per determinare la versione della funzione Lambda invocata:

- CloudWatch Registri: Lambda invia automaticamente ad CloudWatch Amazon Logs `START` una voce di registro contenente l'ID di versione richiamato per ogni chiamata di funzione. Di seguito è riportato un esempio:

  ```
  19:44:37 START RequestId: request id Version: $version
  ```

  Per le invocazioni di alias Lambda utilizza la dimensione `Executed Version` per filtrare i dati del parametro dalla versione richiamata. Per ulteriori informazioni, consulta Utilizzo dei parametri delle 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.

Versioni delle funzioni Lambda

Puoi utilizzare le versioni per gestire la distribuzione delle funzioni. Ad esempio, puoi 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.

> **Note**
>
> Per creare una nuova versione della tua funzione, devi prima apportare modifiche alla versione non pubblicata ($LATEST). Queste modifiche possono includere l'aggiornamento del codice o la modifica delle impostazioni di configurazione. Se $LATEST è identico a una versione pubblicata in precedenza, non potrai creare una nuova versione finché non distribuirai le modifiche a $LATEST.

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 sono immutabili. Ciò significa che non puoi 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 al 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 del runtime”.

Sections

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. Scegliere una funzione e quindi scegliere Versions (Versioni).
3. Nella pagina di configurazione delle versioni, scegliere Publish new version (Pubblica nuova versione).
4. (Facoltativo) Immettere una descrizione della versione.
5. Seleziona Publish (Pubblica).

In alternativa, è possibile pubblicare una versione di una funzione utilizzando l'operazione PublishVersionAPI.

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": "nodejs20.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 qualificato, Lambda richiama implicitamente $LATEST.

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.

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.

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 Ottimizzazione delle sezioni Risorse e Condizioni delle politiche.

È possibile semplificare la gestione delle fonti di eventi e delle politiche AWS Identity and Access Management (IAM) utilizzando gli alias delle funzioni. Per ulteriori informazioni, consulta Creare un alias per una funzione Lambda.

Configurazione di una funzione Lambda per lo 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. I payload del flusso di risposta hanno un limite flessibile di 20 MB, a differenza del limite 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.

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.

Lo streaming delle risposte comporta un costo. Per ulteriori informazioni, consulta la sezione Prezzi di AWS Lambda.

Lambda supporta lo streaming delle risposte sui runtime gestiti di Node.js. Per altri linguaggi, puoi utilizzare un runtime personalizzato con un'integrazione dell'API Runtime personalizzata per eseguire lo streaming delle risposte o utilizzare l'adattatore Web Lambda. Puoi trasmettere le risposte tramite gli URL delle funzioni Lambda, l'AWSSDK o l'API Lambda. InvokeWithResponseStream

> **Note**
>
> Quando testi la funzione tramite la console Lambda, vedrai sempre le risposte come memorizzate nel buffer.

Scrittura di funzioni 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:

- Racchiudi la funzione con il decoratore `awslambda.streamifyResponse()` fornito dai runtime nativi di Node.js.
- 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()`.

Example gestore abilitato allo streaming delle risposte

```
const pipeline = require("util").promisify(require("stream").pipeline);
const { Readable } = require('stream');

exports.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()`.

Example Esempio di terminazione di un flusso con pipeline()

```
const pipeline = require("util").promisify(require("stream").pipeline);

exports.handler = awslambda.streamifyResponse(async (event, responseStream, _context)
 => {
  await pipeline(requestStream, responseStream);
});
```

Example Esempio di terminazione di un flusso senza pipeline()

```
exports.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 delle risposte utilizzando gli URL delle funzioni Lambda

> **Note**
>
> È necessario richiamare la funzione utilizzando un URL della funzione per lo streaming delle risposte.

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 20 MB. Tuttavia, è possibile richiedere un aumento della quota.

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`.

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. Selezionare Salva.

Impostazione della modalità di richiamo di un URL della funzione (AWS CLI)

```
aws lambda update-function-url-config --function-name my-function --invoke-mode
 RESPONSE_STREAM
```

Impostazione della modalità di richiamo di un URL della funzione (AWS CloudFormation)

```
MyFunctionUrl:
  Type: AWS::Lambda::Url
```

```
  Properties:
    AuthType: AWS_IAM
    InvokeMode: RESPONSE_STREAM
```

Per ulteriori informazioni sulla configurazione degli URL della funzione, consulta URL della funzione Lambda.

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 della larghezza di banda illimitata varia in base a una serie di fattori, inclusa la velocità di elaborazione della funzione. Normalmente puoi aspettarti una velocità superiore a 2 MB/s per i primi 6 MB di risposta della funzione. Se la funzione esegue lo streaming delle risposte a una destinazione esterna a AWS, la velocità di streaming dipende anche dalla velocità della connessione Internet esterna.

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 degli URL della funzione, consulta Creazione e gestione degli URL della funzione.

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 l'AWS Command Line Interface (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. Scegliere Create role (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.mjs`.

```
import util from 'util';
import stream from 'stream';
const { Readable } = stream;
const pipeline = util.promisify(stream.pipeline);

/* global awslambda */
export const handler = awslambda.streamifyResponse(async (event, responseStream,
 _context) => {
  const requestStream = Readable.from(Buffer.from(JSON.stringify(event)));
  await pipeline(requestStream, responseStream);
});
```

2. Crea un pacchetto di implementazione.

```
zip function.zip index.mjs
```

3. Creare una funzione Lambda con il comando `create-function`. Sostituisci il valore di `--role` con l'ARN del ruolo del passaggio precedente.

```
aws lambda create-function \
  --function-name my-streaming-function \
  --runtime nodejs16.x \
  --zip-file fileb://function.zip \
  --handler index.handler \
  --role arn:aws:iam::123456789012:role/response-streaming-role
```

Creazione di un URL della funzione

1. Aggiungi alla funzione una policy basata sulle risorse per consentire l'accesso alla funzione URL. Sostituisci il valore di `--principal` con il tuo ID. Account AWS

```
aws lambda add-permission \
  --function-name my-streaming-function \
  --action lambda:InvokeFunctionUrl \
  --statement-id 12345 \
  --principal 123456789012 \
  --function-url-auth-type AWS_IAM \
  --statement-id url
```

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
```

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 "<function_url>" --user "<key:token>" --aws-sigv4 "aws:amz:us-
east-1:lambda" --no-buffer
```

La nostra funzione URL utilizza il tipo di autenticazione `IAM_AUTH`. Ciò significa che devi firmare le richieste sia con la chiave di AWS accesso che con la chiave segreta. Nel comando precedente, sostituiscilo `<key:token>` 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.

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 **`delete`** nel campo di immissione testo e scegli Delete (Elimina).

Distribuzione di funzioni Lambda

È possibile distribuire il codice alla funzione Lambda caricando un archivio di file .zip o creando e caricando un'immagine di container.

Argomenti

archivi di file .zip

Un archivio di file .zip include il codice dell'applicazione e le relative dipendenze. Quando si creano funzioni utilizzando la console Lambda o un kit di strumenti, Lambda crea automaticamente un archivio di file .zip del codice.

Quando crei funzioni con l'API Lambda, gli strumenti da riga di comando o gli AWS SDK, devi creare un pacchetto di distribuzione. È inoltre necessario creare un pacchetto di distribuzione se la funzione utilizza un linguaggio compilato o per aggiungere dipendenze alla funzione. Per distribuire il codice della funzione, carica il pacchetto di implementazione da Amazon Simple Storage Service (Amazon S3) o dal computer locale.

Puoi caricare un file.zip come pacchetto di distribuzione utilizzando la console Lambda AWS Command Line Interface ,AWS CLI() o su un bucket Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3).

Autorizzazioni relative ai file del pacchetto di distribuzione

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 eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 755 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta Set, View, Change, or Remove Permissions on an Object nella documentazione di Microsoft Windows.

Immagini di container

È possibile impacchettare il codice e le dipendenze come immagine di container utilizzando strumenti come l'interfaccia della riga di comando Docker (CLI). È quindi possibile caricare l'immagine nel registro di container ospitato su Amazon Elastic Container Registry (Amazon ECR).

Quando si richiama la funzione, Lambda distribuisce l'immagine di container in un ambiente di esecuzione. Lambda inizializza qualsiasi estensione e quindi esegue il codice di inizializzazione della funzione (il codice esterno al gestore principale). Si noti che la durata di inizializzazione della funzione è inclusa nel tempo di esecuzione fatturato.

Lambda esegue quindi la funzione chiamando il punto di ingresso del codice specificato nella configurazione della funzione (le impostazioni dell'immagine del contenitore ENTRYPOINT e CMD).

AWS fornisce un set di immagini di base open source che è possibile utilizzare per creare l'immagine del contenitore per il codice della funzione. È inoltre possibile utilizzare immagini di base alternative da altri registri di contenitori. AWS fornisce anche un client di runtime open source da aggiungere all'immagine di base alternativa per renderla compatibile con il servizio Lambda.

Inoltre, AWS fornisce un emulatore di interfaccia di runtime per testare le funzioni localmente utilizzando strumenti come la CLI Docker.

> **Note**
>
> Devi creare ogni immagine di container in modo che sia compatibile con una delle architetture del set di istruzioni supportate da Lambda. Lambda fornisce immagini di base per ciascuna delle architetture del set di istruzioni; inoltre, Lambda fornisce immagini di base che supportano entrambe le architetture.
> L'immagine creata per la tua funzione deve essere destinata a una sola delle architetture.

Non sono previsti costi aggiuntivi per la creazione di pacchetti e la distribuzione di funzioni come immagini di container. Quando viene richiamata una funzione distribuita come immagine di container,

si paga per le richieste di chiamata e la durata dell'esecuzione. Vanno invece addebitati costi per la memorizzazione delle immagini di container in Amazon ECR. Per ulteriori informazioni, consulta la pagina dei prezzi di Amazon ECR.

Sicurezza delle immagini

Quando Lambda scarica per la prima volta l'immagine del container dalla sorgente originale (Amazon ECR), l'immagine di container viene ottimizzata, crittografata e archiviata utilizzando metodi di crittografia convergente autenticati. Tutte le chiavi necessarie per decrittografare i dati dei clienti sono protette utilizzando chiavi gestite dal cliente. AWS KMS Per tenere traccia e controllare l'utilizzo che fa Lambda delle chiavi gestite del cliente, è possibile visualizzare i log AWS CloudTrail.

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.

> **Note**
>
> Non è possibile modificare il tipo di pacchetto di distribuzione (.zip o immagine contenitore) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine del contenitore per utilizzare un archivio di file.zip. È necessario creare una nuova funzione.

Argomenti

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 Lambda in Node.js con archivi di file .zip

Python

Utilizzo di archivi di file .zip per le funzioni Lambda in Python

Ruby

Utilizzo di archivi di file .zip per le funzioni Lambda in Ruby

Java

Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR

Go

- Distribuisci 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

- Implementa le funzioni PowerShell Lambda con 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).

> **Note**
>
> La console AWS Cloud9 Lambda fornisce un ambiente di sviluppo integrato nel browser. Puoi anche usarle AWS Cloud9 per sviluppare funzioni Lambda nel tuo ambiente. Per ulteriori informazioni, consulta Lavorare con AWS Lambda le funzioni utilizzando la Kit di strumenti AWS guida per l' AWS Cloud9 utente.

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.js, Python 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.
   - Nella casella di testo, inserisci l'URL del link S3 dell'archivio di file .zip, quindi scegli Save (Salva).

Modifica del runtime

Se si aggiorna la configurazione della funzione in modo da utilizzare una nuova versione di runtime, potrebbe essere necessario aggiornare il codice della funzione per renderlo compatibile con la nuova versione. 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.

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. Selezionare 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 può 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. Selezionare 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

AWS CloudFormation

È possibile utilizzare AWS CloudFormation per creare una funzione Lambda che utilizza un archivio di file.zip. Nel AWS CloudFormation modello, la `AWS::Lambda::Function` risorsa specifica la funzione Lambda. Per le descrizioni delle proprietà della `AWS::Lambda::Function` risorsa, AWS::Lambda::Functionconsultate la Guida per l'AWS CloudFormation utente.

Nella risorsa `AWS::Lambda::Function`, impostare le seguenti proprietà per creare una funzione definita come archivio di file .zip:

- AWS::Lambda::Function
  - 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 o Python, puoi fornire il codice sorgente inline della funzione Lambda.
  - Runtime: imposta il valore di runtime.
  - Architettura: imposta il valore dell'architettura su cui `arm64` utilizzare il processore AWS Graviton2. Per impostazione predefinita, il valore dell'architettura è `x86_64`.

Creare una funzione Lambda utilizzando un'immagine del contenitore

Il codice della 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:

- Usare un'immagine AWS di 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.

- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  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 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.

- Utilizzo di un'immagine non di base AWS

  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 caricala in un repository Amazon Elastic Container Registry (Amazon ECR). Quindi, specifica l'URI del repository al momento della creazione della funzione. Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS 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.

Questa pagina spiega i tipi di immagini di base e i requisiti per la creazione di immagini di container compatibili con Lambda.

> **Note**
>
> Non è possibile modificare il tipo di pacchetto di distribuzione (.zip o immagine del contenitore) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine del contenitore per utilizzare un archivio di file.zip. È necessario creare una nuova funzione.

Argomenti

Requisiti

Installa l'AWS Command Line Interface (AWS CLI) versione 2 e la CLI Docker. Sono inoltre necessari i seguenti requisiti:

- L'immagine di container deve implementare l'API di runtime Lambda. 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. Verifica che il codice dell'applicazione non si basi su file che altri utenti Linux non possono eseguire.
- Lambda supporta solo le immagini di container basate su Linux.
- 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.

Usare un'immagine AWS di 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, AL2023 e versioni successive si basano sull'immagine minima del contenitore Amazon Linux 2023. Le immagini di base precedenti utilizzavano 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`.

Le immagini basate su AL2023 utilizzano `microdnf` (symlinked as`dnf`) come gestore di pacchetti anzichéyum, 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 immagini basate su AL2023, 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 la sezione Introduzione al runtime di Amazon Linux 2023 per AWS Lambda sul AWS Compute Blog.

> **Note**
>
> Per eseguire immagini basate su AL2023 localmente, incluso with AWS Serverless Application Model (AWS SAM), devi usare 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 di base Node.js)
- Python
- Java
- Go
- .NET
- Ruby

Utilizzo di un'immagine di AWS base solo per il sistema operativo

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 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	Definizione come obsoleto
al2023	Runtime solo per il sistema operativo	Amazon Linux 2023	Dockerfile per Runtime solo per sistema operativo su GitHub	

Tag	Runtime	Sistema operativo	Dockerfile	Definizione come obsoleto
al2	Runtime solo per il sistema operativo	Amazon Linux 2	Dockerfile per Runtime solo per sistema operativo attivo GitHub	

Galleria pubblica di Amazon Elastic Container Registry: gallery.ecr.aws/lambda/provided

Utilizzo di un'immagine non di base AWS

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.

Client di interfaccia runtime

Se utilizzi un'immagine di base solo per il sistema operativo o un'immagine di base alternativa, devi includere un client dell'interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime deve estendereAPI di runtime Lambda, che gestisce l'interazione tra Lambda e il codice della funzione. AWS fornisce client di interfaccia di runtime open source per le seguenti lingue:

- Node.js
- Python
- Java

- .NET
- Go
- Ruby
- Rust: il client di runtime Rust è un pacchetto sperimentale. È soggetto a modifiche ed è destinato esclusivamente a scopi di valutazione.

Se utilizzi un linguaggio che non dispone di un AWS client di interfaccia di runtime fornito, devi crearne uno personalizzato.

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 `GetRepositoryPolicy` includano e. `SetRepositoryPolicy`

Ad esempio, utilizza la console IAM per creare un ruolo con la seguente policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "ecr:SetRepositoryPolicy",
        "ecr:GetRepositoryPolicy"
      ],
      "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world"
    }
  ]
}
```

Policy del repository Amazon ECR

Per una funzione nello stesso account dell'immagine di container in Amazon ECR, puoi aggiungere le autorizzazioni `ecr:BatchGetImage` e `ecr:GetDownloadUrlForLayer` alla policy del tuo repository Amazon ECR. 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 aggiunge `ecr:BatchGetImage` e `ecr:GetDownloadUrlForLayer` alle autorizzazioni del repository di immagini di container. Lambda può aggiungere queste 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

Un account diverso nella stessa Regione può creare una funzione che utilizza un'immagine container di proprietà del tuo account. Nell'esempio seguente, la policy delle autorizzazioni del repository Amazon ECR richiede le seguenti istruzioni per concedere l'accesso all'account numero 123456789012.

- CrossAccountAutorizzazione: consente all'account 123456789012 di creare e aggiornare funzioni Lambda che utilizzano immagini da questo repository ECR.
- Politica LambdaECR ImageCrossAccountRetrieval: Lambda alla fine imposterà 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

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CrossAccountPermission",
      "Effect": "Allow",
      "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      }
    },
    {
      "Sid": "LambdaECRImageCrossAccountRetrievalPolicy",
      "Effect": "Allow",
      "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Condition": {
        "StringLike": {
          "aws:sourceARN": "arn:aws:lambda:us-east-1:123456789012:function:*"
        }
      }
    }
  ]
}
```

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 di autorizzazione ECR. Questo approccio è in linea con le migliori pratiche di AWS sicurezza che prevedono l'impostazione di autorizzazioni ristrette nelle policy IAM.

Oltre alle autorizzazioni Lambda, l'utente o il ruolo che crea la funzione deve disporre `BatchGetImage` anche delle autorizzazioni e. `GetDownloadUrlForLayer`

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. La funzione passa quindi allo stato `Active`. Mentre lo stato è `Pending`, è possibile richiamare la funzione, ma non è possibile completare altre operazioni. Le chiamate che si verificano durante l'aggiornamento dell'immagine eseguono il codice dell'immagine precedente.

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 del contenitore 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.

Comprensione dei metodi di invocazione della funzione Lambda

Dopo aver distribuito la 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: usa l'SDK per richiamare la tua AWS funzione a livello di codice.
- L'API Invoke: utilizza l'API Lambda Invoke per richiamare direttamente la tua funzione.
- The AWS Command Line Interface (AWS CLI): utilizza il `aws lambda invoke` AWS CLI comando per richiamare direttamente la funzione dalla riga di comando.
- Un endpoint HTTP (S) con URL di funzione: utilizzate gli URL delle funzioni per creare un endpoint HTTP (S) dedicato da 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. Questo si chiama trigger. Per i servizi basati su stream e code, Lambda richiama la funzione con batch di record. Questa operazione è denominata mappatura delle sorgenti 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 `InvocationType` request nell'API Invoke determina il modo in cui Lambda richiama la tua funzione. Un valore di `RequestResponse` indica una chiamata sincrona e un valore di indica una chiamata asincrona. `Event`

Se la chiamata della funzione genera un errore, per le chiamate sincrone, visualizza il messaggio di errore nella risposta e riprova a richiamare manualmente. Per le chiamate asincrone, Lambda gestisce automaticamente i nuovi tentativi e può inviare i record di chiamata a una destinazione.

Invocazione sincrona

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 invocare una funzione in modo sincrono con l'AWS CLI, utilizzare il comando `invoke`.

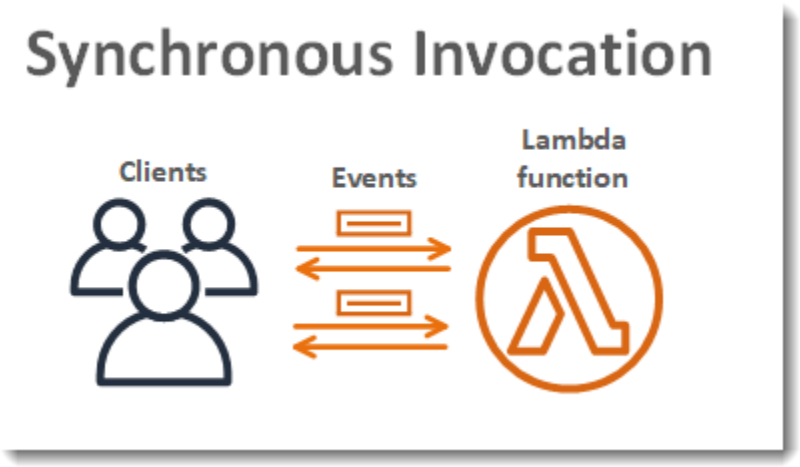
```
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{ "key": "value" }' response.json
```

L'opzione cli-binary-format è necessaria 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 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 Potenzia 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 consente di interagire con i servizi AWS tramite i comandi nella shell a riga di comando. Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- AWS CLI – Configurazione rapida con `aws configure`

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":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "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 necessaria 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 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'AWS CLI e l'SDK AWS, inoltre, riprovano automaticamente sugli errori di timeout del client, di throttling e di servizio. Per ulteriori informazioni, consulta Comprensione del comportamento dei tentativi in Lambda.

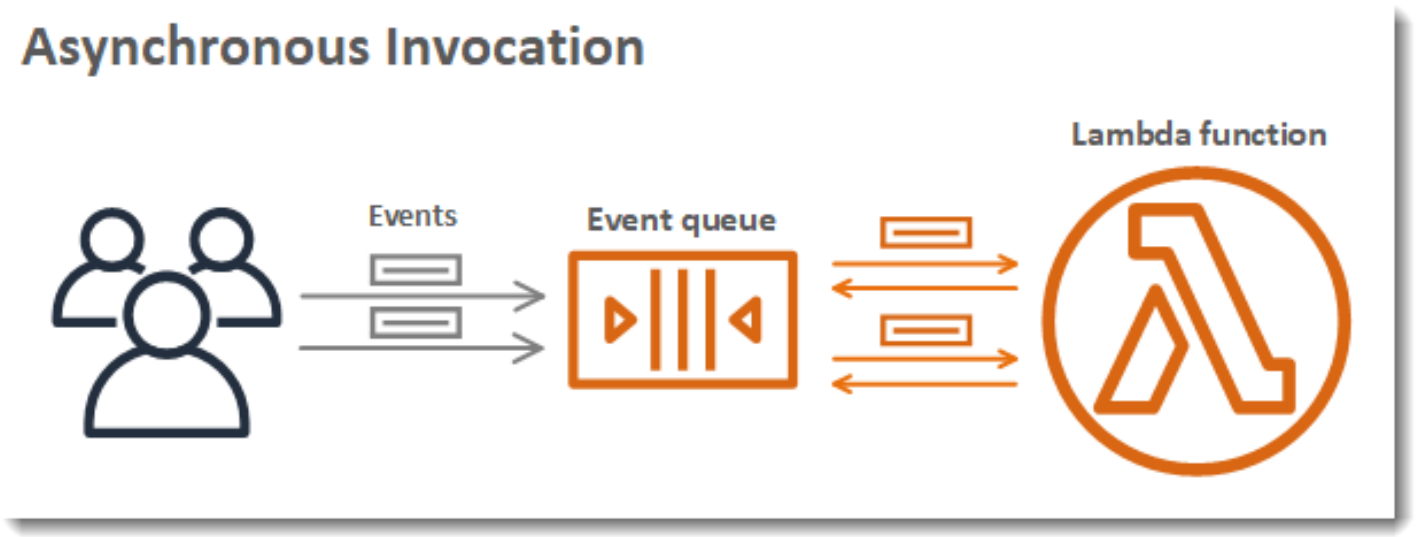
Invocazione asincrona

Molti 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. 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 EventBridge () per concatenare i componenti della tua applicazione. EventBridge

Sections

Come Lambda gestisce le chiamate asincrone

Il seguente diagramma mostra i client che richiamano una funzione Lambda in modo asincrono. Lambda accoda gli eventi prima di inviarli alla funzione.



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 invocare una funzione in modo asincrono, impostare il parametro tipo di invocazione su `Event`.

```
aws lambda invoke \
  --function-name my-function  \
  --invocation-type Event \
  --cli-binary-format raw-in-base64-out \
  --payload '{ "key": "value" }' response.json
```

L'cli-binary-formatopzione è obbligatoria se utilizzi 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.

```
{
    "StatusCode": 202
}
```

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.

Lambda gestisce la coda di eventi asincroni della funzione e tenta di riprovare in caso di errori. Se la funzione restituisce un errore, Lambda tenta di eseguirlo 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 throttling (429) e gli errori di sistema (serie 500), Lambda restituisce l'evento alla coda e tenta di 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.

È anche possibile configurare Lambda in modo che invii un record di invocazione a un altro servizio. Lambda supporta le seguenti destinazioni per l'invocazione asincrona. Tieni presente che le code FIFO SQS e gli argomenti FIFO SNS non sono supportati.

- Amazon SQS: una coda SQS standard.
- Amazon SNS: un argomento SNS standard.
- AWS Lambda: una funzione Lambda.
- Amazon EventBridge: un bus per EventBridge 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 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.

Configurazione della gestione degli errori per la chiamata asincrona

Utilizzare la console Lambda per configurare le impostazioni di gestione degli errori per una funzione, una versione o un alias.

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. Scegliere Save (Salva).

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 evento non riuscita.

Configurazione delle destinazioni per l'invocazione asincrona

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.

> **Note**
>
> Puoi anche conservare i record delle chiamate non riuscite per i seguenti tipi di mappatura delle sorgenti degli eventi: Amazon Kinesis, Amazon DynamoDB, ApacheKafka autogestito e Amazon MSK.

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	Formato JSON specifico della destinazione
Coda Amazon SQS	seghe: SendMessage	Lambda passa il record di chiamata come `Message` alla destinazione.
Argomento Amazon SNS	sns:Publish	Lambda passa il record di chiamata come `Message` alla destinazione.
Funzione Lambda	InvokeFunction	Lambda passa il record di chiamata come payload alla funzione.
EventBridge	eventi: PutEvents	• Lambda passa il record di invocazione come nella `detail` chiamata. PutEvents • Il valore per il campo di eventi `source` è `lambda`. • Il valore per il campo dell'evento `detail-type` è "Risultato della chiamata della funzione Lambda -

Tipo di destinazione	Autorizzazione richiesta	Formato JSON specifico della destinazione
		Successo" o "Risultato della chiamata della funzione Lambda - Errore". • Il campo dell'evento `resource` contiene i nomi della risorsa Amazon (ARN) della funzione e della destinazione. • Per altri campi relativi agli eventi, consulta Amazon EventBridge events.

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. Il record di invocazione contiene dettagli sull'evento, la risposta e il motivo per cui il record è stato inviato.

```
{
    "version": "1.0",
    "timestamp": "2019-11-14T18:16:05.568Z",
    "requestContext": {
        "requestId": "e4b46cbf-b738-xmpl-8880-a18cdf61200e",
        "functionArn": "arn:aws:lambda:us-east-2:123456789012: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"
    }
}
```

I passaggi seguenti descrivono come configurare una destinazione per una funzione utilizzando la console Lambda.

Configurazione di una destinazione per i record di chiamata asincrona

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. Scegliere Save (Salva).

Quando un'invocazione corrisponde alla condizione, Lambda invia un documento JSON con i dettagli sull'invocazione alla destinazione.

Formato JSON specifico della destinazione

- Per Amazon SQS e Amazon SNS (`SnsDestination` e `SqsDestination`), il record di invocazione viene trasferito come `Message` fino alla destinazione.

- Per Lambda (`LambdaDestination`), il record di invocazione viene trasferito come payload alla funzione.
- Per EventBridge (`EventBridgeDestination`), il record di invocazione viene passato come contenuto `detail` nella chiamata. PutEvents Il valore per il campo di eventi `source` è `lambda`. Il valore per il campo dell'evento `detail-type` è Risultato dell'invocazione della funzione Lambda - Successo o Risultato dell'invocazione della funzione Lambda - Errore. Il campo dell'evento `resource` contiene i nomi della risorsa Amazon (ARN) della funzione e della destinazione. Per altri campi relativi agli eventi, consulta Amazon EventBridge events.

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 record di invocazione

```
{
    "version": "1.0",
    "timestamp": "2019-11-14T18:16:05.568Z",
    "requestContext": {
        "requestId": "e4b46cbf-b738-xmpl-8880-a18cdf61200e",
        "functionArn": "arn:aws:lambda:us-east-2:123456789012: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 si attiva il tracciamento X-Ray per una funzione o un servizio che richiama una funzione, Lambda aggiunge un'intestazione X-Ray alla richiesta e passa l'intestazione al servizio di destinazione. Le tracce dei servizi upstream vengono collegate automaticamente alle tracce delle funzioni Lambda a valle e dei servizi di destinazione, creando una end-to-end visualizzazione dell'intera applicazione. Per ulteriori informazioni sul tracciamento, consulta Visualizza le chiamate alla funzione Lambda utilizzando AWS X-Ray.

API di configurazione delle chiamate asincrone

Per gestire le impostazioni di chiamata asincrona con AWS CLI o AWS SDK, utilizza le seguenti operazioni API.

- PutFunctionEventInvokeConfig
- GetFunctionEventInvokeConfig
- UpdateFunctionEventInvokeConfig
- ListFunctionEventInvokeConfigurazioni
- DeleteFunctionEventInvokeConfig

Per configurare l'invocazione asincrona con, usa il comando. AWS CLI`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
```

Verrà visualizzato l'output seguente:

```
{
```

```
    "LastModified": 1573686021.479,
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:error:$LATEST",
    "MaximumRetryAttempts": 0,
    "MaximumEventAgeInSeconds": 3600,
    "DestinationConfig": {
        "OnSuccess": {},
        "OnFailure": {}
    }
}
```

Il comando `put-function-event-invoke-config` sovrascrive qualsiasi configurazione esistente sulla funzione, versione o alias. Per configurare un'opzione senza reimpostarne altre, utilizzare `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 error \
--destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-
east-2:123456789012:destination"}}'
```

Verrà visualizzato l'output seguente:

```
{
    "LastModified": 1573687896.493,
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:error:$LATEST",
    "MaximumRetryAttempts": 0,
    "MaximumEventAgeInSeconds": 3600,
    "DestinationConfig": {
        "OnSuccess": {},
        "OnFailure": {
            "Destination": "arn:aws:sqs:us-east-2:123456789012:destination"
        }
    }
}
```

Code 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, una coda DLQ fa parte della configurazione specifica della versione di una funzione, quindi viene bloccata quando si pubblica una versione. Le destinazioni in caso di errore supportano anche destinazioni aggiuntive e includono dettagli sulla risposta della funzione nel record di invocazione.

Per rielaborare gli eventi in una coda DLQ, è possibile impostarla come fonte degli 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. Se non si dispone di una coda o un argomento, crearne uno. Scegliere il tipo di destinazione che corrisponde al caso d'uso specifico.

- Coda Amazon SQS: una coda conserva gli eventi non riusciti finché non vengono richiamati. Scegli una coda standard Amazon SQS se prevedi che una singola entità, come una funzione Lambda o un CloudWatch allarme, elabori l'evento non riuscito. Per ulteriori informazioni, consulta Utilizzo di Lambda con Amazon SQS.

  Creare una coda nella console di 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. Per esempio, è possibile configurare un argomento in modo tale che invii gli eventi a un indirizzo e-mail, a una funzione Lambda e/o a un endpoint HTTP. Per ulteriori informazioni, consulta Richiamo di funzioni Lambda con le notifiche di Amazon SNS.

  Creare un argomento nella console di Amazon SNS.

Per inviare gli eventi a una coda o argomento, la funzione necessita di autorizzazioni aggiuntive. Aggiungere una policy con le autorizzazioni necessarie per il ruolo di esecuzione di una funzione.

- Amazon SQS — sqs: SendMessage
- Amazon SNS – sns:Publish

Se la coda o l'argomento di destinazione sono crittografati con una chiave gestita dal cliente, il ruolo di esecuzione deve essere anche un utente nella policy basata sulle risorse della chiave.

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.

Per configurare una coda dead-letter

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. Impostare la risorsa DLQ su Amazon SQS o Amazon SNS.
6. Scegliere l'argomento o la coda target.
7. Scegliere Save (Salva).

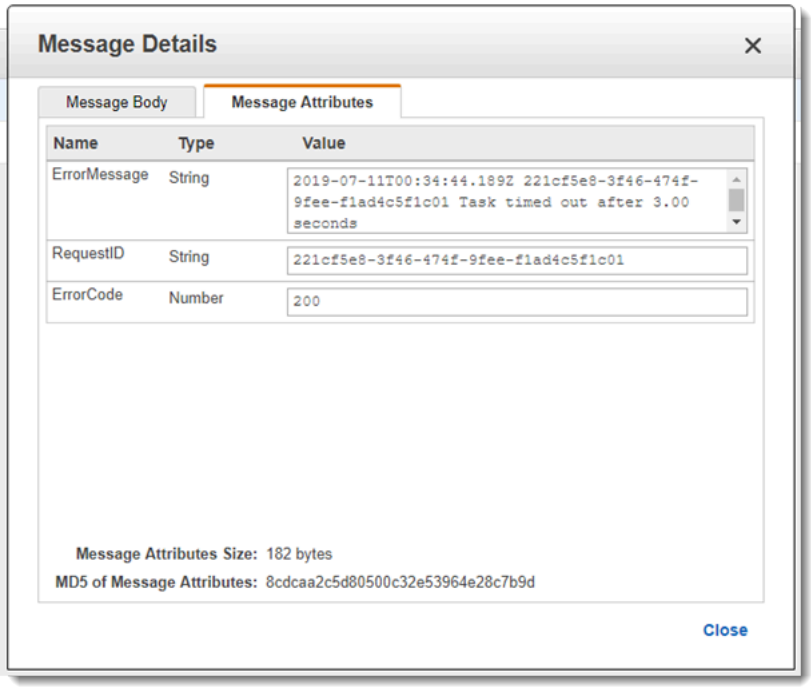
Per configurare una coda di lettere non scritte con, usa il AWS CLI comando. `update-function-configuration`

```
aws lambda update-function-configuration --function-name my-function \
--dead-letter-config TargetArn=arn:aws:sns:us-east-2:123456789012: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 .

Attributi dei messaggi della coda DLQ

- RequestID (String) – L'ID della richiesta di invocazione. Gli ID della richiesta appaiono nei log della funzione. È inoltre possibile utilizzare l'SDK X-Ray per registrare l'ID di richiesta su un attributo nella traccia. Si possono quindi cercare le tracce in base all'ID richiesta nella console X-Ray.
- ErrorCode(Numero) — Il codice di stato HTTP.
- ErrorMessage(String) — Il primo 1 KB del messaggio di errore.



Se Lambda non riesce a inviare un messaggio alla coda delle lettere non scritte, elimina l'evento ed emette la metrica Errors. DeadLetter 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 prossime a 256 KB 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, consultare Utilizzo di Lambda con Amazon SQS.

In che modo Lambda elabora i record provenienti da fonti di eventi basate su stream e code

Una mappatura dell'origine degli eventi è una risorsa Lambda che legge gli elementi dai servizi basati su stream e code e richiama una funzione con batch di record. I seguenti servizi utilizzano le mappature delle sorgenti degli eventi per richiamare le funzioni Lambda:

- Amazon DynamoDB
- Amazon Kinesis
- Amazon MQ
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Apache Kafka gestito dal cliente
- Amazon Simple Queue Service (Amazon SQS)
- Amazon DocumentDB (compatibile con MongoDB) (Amazon DocumentDB)

> **Warning**
>
> Le mappature delle sorgenti degli eventi 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 ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

In che modo le mappature delle sorgenti degli eventi differiscono dai trigger diretti

Alcuni AWS servizi 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 crei 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) 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 viene pubblicato su un argomento SNS. Per ulteriori informazioni, consulta Tutorial: Utilizzo AWS Lambda con Amazon Simple Notification Service.
- Amazon API Gateway: richiama una funzione quando viene effettuata una richiesta API a un endpoint specifico. Per ulteriori informazioni, consulta Richiamo di una funzione Lambda utilizzando un endpoint Amazon API Gateway.

Le mappature delle sorgenti degli eventi sono risorse Lambda create e gestite all'interno del servizio Lambda. Le mappature delle sorgenti degli eventi sono progettate per l'elaborazione di dati o messaggi in streaming ad alto volume dalle code. L'elaborazione dei record da uno stream 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 di batch, è possibile configurare una finestra di batch (MaximumBatchingWindowInSecondi) 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:

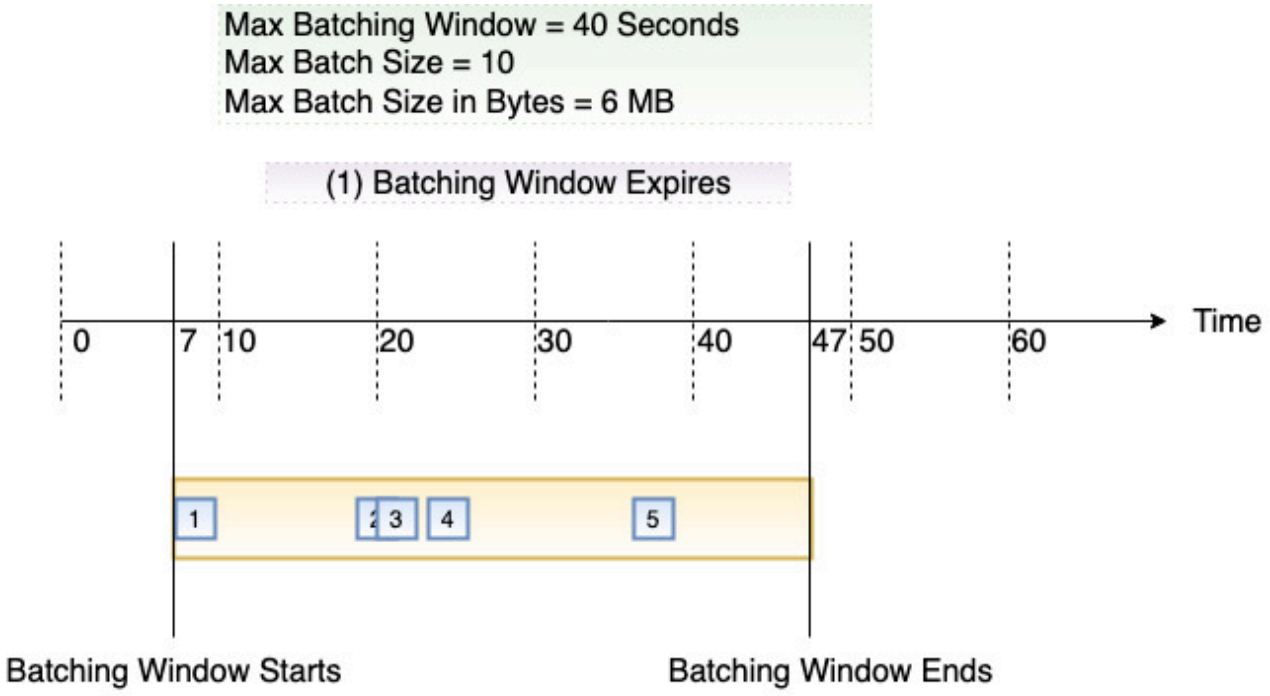
- La finestra di dosaggio raggiunge il valore massimo. Il comportamento predefinito della finestra di batch varia a seconda dell'origine specifica dell'evento.
  - Per le origini eventi Kinesis, DynamoDB e Amazon SQS: la finestra di batch di default è 0 secondi. Ciò significa che Lambda invia i batch alla tua funzione solo quando viene soddisfatta la dimensione del batch o viene raggiunto il limite di dimensione del payload. Per impostare una finestra di batch, configura. `MaximumBatchingWindowInSeconds` È possibile impostare questo parametro su qualsiasi valore compreso tra 0 e 300 secondi con incrementi di 1 secondo. Se si configura una finestra di batch, la finestra successiva inizia non appena viene completata la precedente chiamata 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

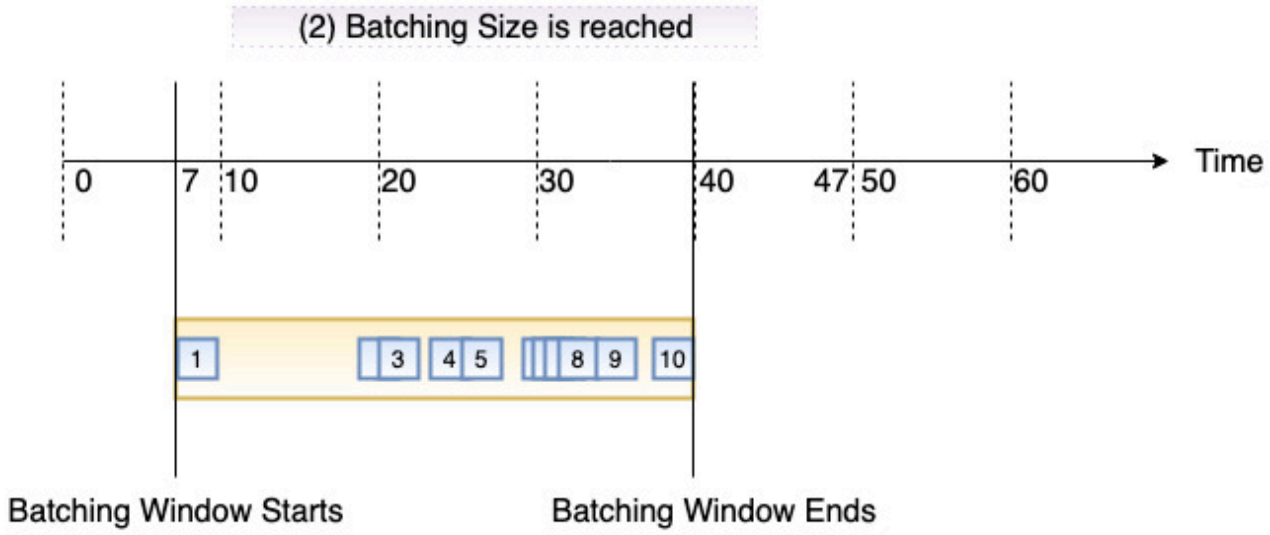
`MaximumBatchingWindowInSeconds` su qualsiasi valore da 0 secondi a 300 secondi con incrementi di secondi. Una finestra di batch inizia non appena arriva il primo registro.

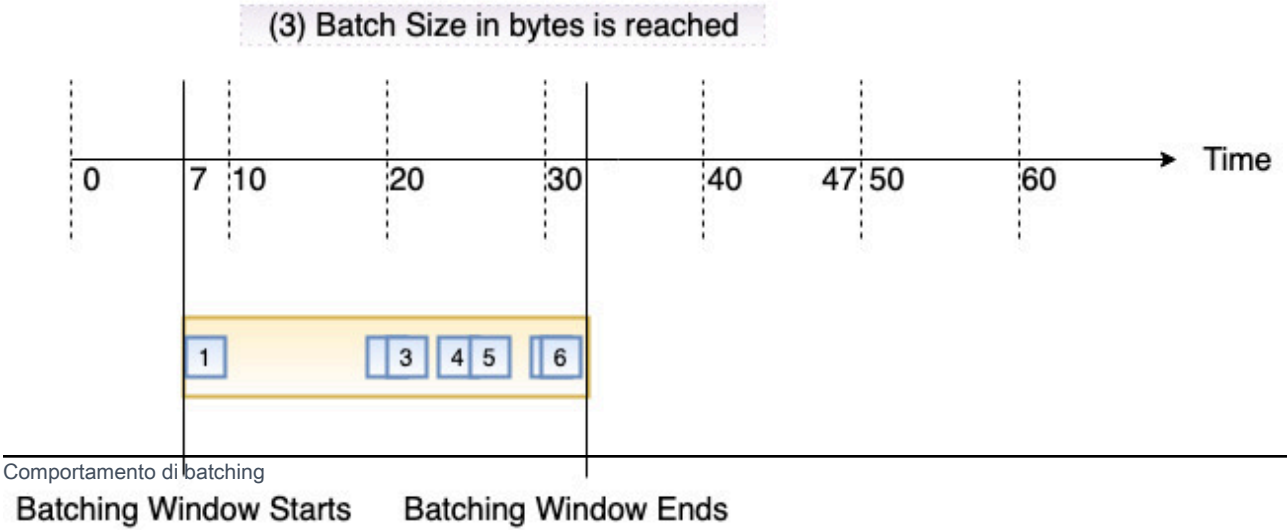
> **Note**
>
> Poiché è possibile modificare solo `MaximumBatchingWindowInSeconds` in incrementi di secondi, non è possibile tornare alla finestra di batch predefinita di 500 ms dopo averla modificata. Per ripristinare la finestra di batch predefinita, è necessario creare una nuova mappatura dell'origine 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 i dettagli su questi valori, consulta le BatchSizespecifiche per il funzionamento dell'`CreateEventSourceMapping`API.
- 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
0
7
10
20
30
40
47
50
60
Time
1
3
4
5
Batching Window Starts
Batching Window Ends


(2) Batching Size is reached
0
7
10
20
30
40
47
50
60
Time
1
3
4
5
8
9
10
Batching Window Starts
Batching Window Ends


(3) Batch Size in bytes is reached
0
7
10
20
30
40
47
50
60
Time
1
3
4
5
6
Batching Window Starts
Batching Window Ends

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 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 un flusso Amazon DynamoDB. Con DynamoDB Streams, è possibile attivare una funzione Lambda per eseguire lavoro aggiuntivo ogni volta che una tabella DynamoDB viene aggiornata.

Lambda legge i record dal flusso 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.

Sections

- CloudWatch Metriche Amazon
- Finestre temporali
- Segnalazione errori articoli batch
- Parametri di configurazione di Flussi Amazon DynamoDB
- Tutorial: Utilizzo AWS Lambda con flussi Amazon DynamoDB
- Codice della funzione di esempio
- Modello AWS SAM per un'applicazione DynamoDB

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"
    }
  ]}
```

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**
>
> Le mappature delle sorgenti degli eventi 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 ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

Configura l'ParallelizationFactorimpostazione per elaborare uno shard di un flusso DynamoDB con più di una chiamata Lambda contemporaneamente. È possibile specificare il numero di batch simultanei di cui Lambda esegue il polling da una partizione da un fattore di parallelizzazione compreso tra da 1 (predefinito) e 10. Quando si aumenta il numero di batch simultanei per shard, Lambda garantisce comunque l'elaborazione in ordine a livello di articolo (chiave di partizione e 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 una partizione 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 dalla stessa partizione 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.

Autorizzazioni del ruolo di esecuzione

La policy AWSLambdaDynamoDBExecutionRole AWS gestita include le autorizzazioni che Lambda deve leggere dal tuo flusso DynamoDB. Aggiungi questa policy gestita al ruolo di esecuzione della tua funzione.

Per inviare i record dei batch non riusciti a una coda SQS standard o a un argomento SNS standard, la funzione necessita di autorizzazioni aggiuntive. Ogni servizio di destinazione richiede un'autorizzazione diversa, come segue:

- Amazon SQS — sqs: SendMessage
- Amazon SNS – sns:Publish

Aggiungi le autorizzazioni e crea la mappatura delle sorgenti degli eventi

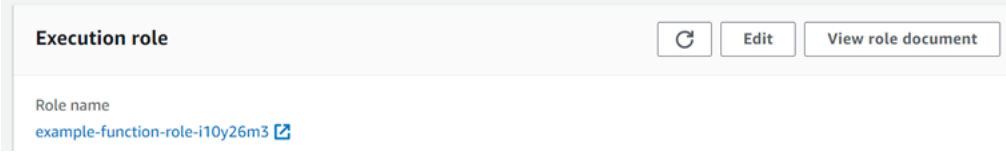
Crea una mappatura dell'origine eventi per indicare a Lambda di inviare i record dal proprio flusso a una funzione Lambda. È possibile creare più mappature dell'origine eventi per elaborare gli stessi dati con più funzioni Lambda o per elaborare elementi da più flussi con una singola funzione.

Per configurare la tua funzione per la lettura da DynamoDB Streams, collega la policy gestita AWSLambdaDynamoDBExecutionRole AWS al tuo ruolo di esecuzione e quindi crea un trigger DynamoDB.

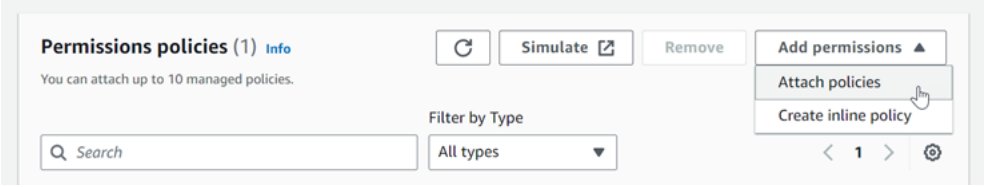
Per aggiungere autorizzazioni e creare un trigger

1. Aprire la pagina Functions (Funzioni) della console Lambda.

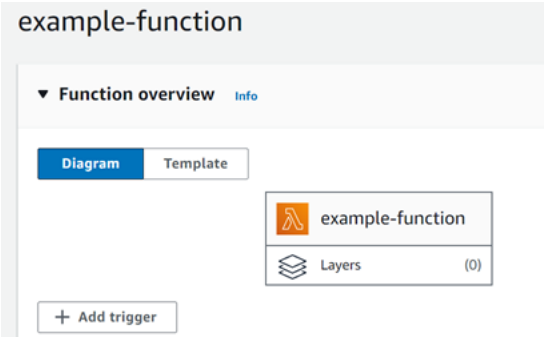
2. Scegliere il nome della funzione.
3. Quindi, seleziona la scheda Configuration (Configurazione) e poi Permissions (Autorizzazioni).
4. In Nome del ruolo, scegli il link al tuo ruolo di esecuzione. Questo link apre il ruolo nella console IAM.



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 il AWS Managed Policy AWSLambdaDynamoDBExecutionRoleReference.
7. Torna alla tua 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 sorgenti di 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.
- Concurrent batches per shard (Batch simultanei per partizione): elabora più batch dalla stessa partizione 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.

Gestione degli errori

La gestione degli errori per le mappature delle sorgenti 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 delle sorgenti degli eventi (secondi). MaximumRecord AgeIn
- Durante la chiamata: se la funzione viene richiamata ma restituisce un errore, Lambda riprova fino alla scadenza dei record, al superamento dell'età massima (MaximumRecordAgeInsecondi) o al raggiungimento della quota di tentativi configurata (Tentments). MaximumRetry Per gli errori di funzione, puoi anche configurare BisectBatchOnFunctionError, che divide un batch non riuscito in due batch più piccoli, 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 sulla partizione 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 con metadati sulla chiamata non riuscita. Puoi configurare qualsiasi argomento Amazon SNS o coda Amazon SQS come destinazione. Il tuo ruolo di esecuzione deve avere le autorizzazioni per la destinazione:

- Per le destinazioni SQS: sqs: SendMessage
- Per le destinazioni SNS: sns:Publish

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. Scegliere Save (Salva).

Puoi anche configurare una destinazione in caso di errore utilizzando (). AWS Command Line Interface AWS CLI Ad esempio, il seguente comando create-event-source-mapping 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 seguente comando update-event-source-mapping 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. Utilizzate 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": ""}}'
```

L'esempio seguente mostra un record di chiamata per un flusso DynamoDB.

Example Record di chiamata

```
{
    "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.

CloudWatch Metriche Amazon

Lambda emette il parametro `IteratorAge` quando la funzione termina l'elaborazione di un batch di record. Il parametro indica a quando risaliva l'ultimo record nel batch al momento del completamento dell'elaborazione. Se la funzione elabora nuovi eventi, è possibile utilizzare la cronologia di iterazione per stimare la latenza tra il momento in cui un record viene aggiunto e il momento in cui la funzione lo elabora.

Una tendenza in aumento nella cronologia di iterazione può indicare problemi con la funzione. Per ulteriori informazioni, consulta Utilizzo dei parametri delle funzioni Lambda.

Finestre temporali

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. Lo stato può essere un massimo di 1 MB per partizione. Se supera quella dimensione, Lambda termina la finestra in anticipo.

Ogni record in un flusso appartiene a una finestra specifica. Lambda elaborerà 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 ribaltabile, specifica la finestra in secondi () TumblingWindowInSeconds. Il seguente comando di esempio AWS Command Line Interface (AWS CLI) crea una mappatura della sorgente degli eventi in streaming con una finestra di rotazione 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 la partizione termina, Lambda considera la finestra chiusa e le partizioni secondarie 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}
```

Segnalazione errori articoli batch

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 successi parziali può contribuire a ridurre il numero di tentativi su un record, anche se non impedisce del tutto la possibilità di tentativi in un record riuscito.

Per attivarlo`ReportBatchItemFailures`, includete il valore enum **`ReportBatchItemFailures`** nell'elenco dei FunctionResponse tipi. Questo elenco indica quali tipi di risposta sono abilitati per la funzione. È possibile configurare questo elenco quando si crea o si aggiorna una mappatura della sorgente degli eventi.

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.

Di seguito sono riportati alcuni esempi di codice di funzione che restituiscono l'elenco degli ID dei messaggi con errori nel batch:

.NET

AWS SDK for .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 degli errori degli elementi batch di DynamoDB con Lambda utilizzando.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.DefaultLambdaJsonSeri

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 altro da sapere. 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 con Lambda utilizzando 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 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 con Lambda utilizzando 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.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 Serializable> {

    @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 (v3) JavaScript

> **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

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBBatchItemFailure, DynamoDBStreamEvent } from "aws-lambda";

export const handler = async (event: DynamoDBStreamEvent):
 Promise<DynamoDBBatchItemFailure[]> => {

    const batchItemsFailures: DynamoDBBatchItemFailure[] = []
    let curRecordSequenceNumber

    for(const record of event.Records) {
        curRecordSequenceNumber = record.dynamodb?.SequenceNumber

        if(curRecordSequenceNumber) {
            batchItemsFailures.push({
                itemIdentifier: curRecordSequenceNumber
            })
        }
    }
```

```
    return batchItemsFailures
}
```

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 degli errori degli elementi batch di DynamoDB con Lambda utilizzando 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\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 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 con Lambda utilizzando 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 altro da sapere. 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 con Lambda utilizzando Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
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 sapere. 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 con Lambda utilizzando Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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
}
```

Parametri di configurazione di Flussi Amazon DynamoDB

Tutti i tipi di sorgenti di eventi Lambda condividono le stesse operazioni CreateEventSourceMappinge quelle dell'UpdateEventSourceMappingAPI. Tuttavia, solo alcuni dei parametri si applicano a DynamoDB Streams.

Parametri di origine eventi che si applicano a DynamoDB Streams

Parametro	Obbligatorio	Predefinito	Note
BatchSize	N	100	Massimo: 10.000.
BisectBatchOnFunctionErrore	N	false	

Parametro	Obbligatorio	Predefinito	Note
DestinationConfig	N		Una destinazione della coda standard di Amazon SQS o di un argomento standard di Amazon SNS per i record scartati.
Abilitato	N	true	
EventSourceArn	Y		L'ARN del flusso dei dati o di un consumer di flusso.
FilterCriteria	N		Filtro eventi Lambda
FunctionName	Y		
FunctionResponseTipologie	N		Per consentire alla funzione di segnalare errori specifici in un batch, includi il valore `ReportBatchItemFailures` in `FunctionResponseTypes`. Per ulteriori informazioni, consulta Segnalazione errori articoli batch.
MaximumBatchingWindowInSecondi	N	0	

Parametro	Obbligatorio	Predefinito	Note
MaximumRecordAgeInSecondi	N	-1	-1 significa infinito: i record non riusciti vengono ritentati fino alla scadenza del record. Il limite di conservazione dei dati per DynamoDB Streams è di 24 ore. Minimo: -1 Massimo: 604.800
MaximumRetryTentativi	N	-1	-1 sta per infinito: i record non riusciti vengono ripetuti fino alla scadenza del record Minimo: 0 Massimo: 10.000.
ParallelizationFactor	N	1	Maximum: 10
StartingPosition	Y		TRIM_HORIZON o LATEST
TumblingWindowInSeconds	N		Minimo: 0 Massimo: 900

Tutorial: Utilizzo AWS Lambda con flussi Amazon DynamoDB

In questo tutorial creerai una funzione Lambda per consumare eventi da un flusso Amazon DynamoDB.

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 l'AWS Command Line Interface (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 del ruolo di esecuzione

Crea il ruolo di esecuzione che autorizza la funzione ad accedere alle risorse. AWS

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`**.

AWSLambdaDynamoDBExecutionRoleDispone delle 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 elabori i tuoi eventi DynamoDB. Il codice della funzione scrive alcuni dei dati degli eventi in entrata in Logs. CloudWatch

.NET

AWS SDK for .NET

> **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.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.DefaultLambdaJsonSeri

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 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 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 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 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 (v3) JavaScript

> **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

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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.

Consumo di un evento DynamoDB con Lambda utilizzando 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\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 di più su. 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 Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

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 di più su. 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 Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

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 di più su. 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 Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

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 nodejs18.x
 \
    --role arn:aws:iam::111122223333:role/lambda-dynamodb-role
```

Test della funzione Lambda

In questo passaggio, si richiama manualmente la funzione Lambda utilizzando il comando `invoke` AWS Lambda CLI e il seguente evento DynamoDB di esempio. Copia quanto segue 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'cli-binary-formatopzione 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.

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 DynamDB 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 lo streaming.

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 delle sorgenti degli eventi, AWS Lambda inizia 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 condivideranno il throughput di lettura per il flusso 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 dallo stream sono AWS Lambda stati letti correttamente e che è stata richiamata

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

Metti alla prova l'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 aggiornamenti allo stream, richiama la funzione Lambda passando i dati degli eventi che trova nello stream.
3. La tua funzione viene eseguita e crea registri in Amazon CloudWatch. Puoi verificare i log riportati nella CloudWatch console Amazon.

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 ai tuoi. 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 **`delete`** 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).

Codice della funzione di esempio

Il codice di esempio è disponibile per i seguenti linguaggi.

Argomenti

- Node.js
- Java 11
- C#
- Python 3
- Go

Node.js

Nell'esempio seguente vengono elaborati i messaggi da DynamoDB e ne viene registrato il contenuto.

Example ProcessDynamoDBStream.js

```
console.log('Loading function');

exports.lambda_handler = function(event, context, callback) {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(function(record) {
        console.log(record.eventID);
        console.log(record.eventName);
        console.log('DynamoDB Record: %j', record.dynamodb);
```

```
    });
    callback(null, "message");
};
```

Comprimere il codice di esempio per creare un pacchetto di distribuzione. Per istruzioni, consultare Distribuisci funzioni Lambda in Node.js con archivi di file .zip.

Java 11

Nell'esempio seguente vengono elaborati i messaggi da DynamoDB e ne viene registrato il contenuto. `handleRequest` è il gestore che viene richiamato da AWS Lambda e che fornisce i dati degli eventi. Il gestore utilizza la classe `DynamodbEvent` predefinita indicata nella libreria `aws-lambda-java-events`.

Example DDB.java EventProcessor

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler2;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;

public class DDBEventProcessor implements
        RequestHandler2<DynamodbEvent, String> {

    public String handleRequest(DynamodbEvent ddbEvent, Context context) {
        for (DynamodbStreamRecord record : ddbEvent.getRecords()){
           System.out.println(record.getEventID());
           System.out.println(record.getEventName());
           System.out.println(record.getDynamodb().toString());

        }
        return "Successfully processed " + ddbEvent.getRecords().size() + " records.";
    }
}
```

Se il gestore termina normalmente senza eccezioni, Lambda considera il batch di record di input come elaborato correttamente e inizia a leggere nuovi record nel flusso. Se il gestore genera un'eccezione, Lambda considera il batch di record di input come non elaborato e richiama nuovamente la funzione con lo stesso batch di record.

Dipendenze

- `aws-lambda-java-core`
- `aws-lambda-java-events`

Crea il codice con le dipendenze della libreria Lambda per creare un pacchetto di implementazione. Per istruzioni, consultare Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR.

C#

Nell'esempio seguente vengono elaborati i messaggi da DynamoDB e ne viene registrato il contenuto. `ProcessDynamoEvent` è il gestore che viene richiamato da AWS Lambda e che fornisce i dati degli eventi. Il gestore utilizza la classe `DynamoDbEvent` predefinita indicata nella libreria `Amazon.Lambda.DynamoDBEvents`.

Example ProcessingDynamoDBStreams.cs

```
using System;
using System.IO;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

using Amazon.Lambda.Serialization.Json;

namespace DynamoDBStreams
{
    public class DdbSample
    {
        private static readonly JsonSerializer _jsonSerializer = new JsonSerializer();

        public void ProcessDynamoEvent(DynamoDBEvent dynamoEvent)
        {
            Console.WriteLine($"Beginning to process {dynamoEvent.Records.Count}
 records...");

            foreach (var record in dynamoEvent.Records)
            {
                Console.WriteLine($"Event ID: {record.EventID}");
                Console.WriteLine($"Event Name: {record.EventName}");

                string streamRecordJson = SerializeObject(record.Dynamodb);
```

```
                Console.WriteLine($"DynamoDB Record:");
                Console.WriteLine(streamRecordJson);
            }

            Console.WriteLine("Stream processing complete.");
        }

        private string SerializeObject(object streamRecord)
        {
            using (var ms = new MemoryStream())
            {
                _jsonSerializer.Serialize(streamRecord, ms);
                return Encoding.UTF8.GetString(ms.ToArray());
            }
        }
    }
}
```

Sostituire `Program.cs` in un progetto .NET Core con l'esempio precedente. Per istruzioni, consultare Crea e implementa le funzioni Lambda C# con gli archivi di file .zip.

Python 3

Nell'esempio seguente vengono elaborati i messaggi da DynamoDB e ne viene registrato il contenuto.

Example ProcessDynamoDBStream.py

```
from __future__ import print_function

def lambda_handler(event, context):
    for record in event['Records']:
        print(record['eventID'])
        print(record['eventName'])
    print('Successfully processed %s records.' % str(len(event['Records'])))
```

Comprimere il codice di esempio per creare un pacchetto di distribuzione. Per istruzioni, consultare Utilizzo di archivi di file .zip per le funzioni Lambda in Python.

Go

Nell'esempio seguente vengono elaborati i messaggi da DynamoDB e ne viene registrato il contenuto.

Example

```
import (
    "strings"

    "github.com/aws/aws-lambda-go/events"
)

func handleRequest(ctx context.Context, e events.DynamoDBEvent) {

    for _, record := range e.Records {
        fmt.Printf("Processing request data for event ID %s, type %s.\n",
 record.EventID, record.EventName)

        // Print new values for attributes of type String
        for name, value := range record.Change.NewImage {
            if value.DataType() == events.DataTypeString {
                fmt.Printf("Attribute name: %s, value: %s\n", name, value.String())
            }
        }
    }
}
```

Creare l'eseguibile con `go build` e creare un pacchetto di distribuzione. Per istruzioni, consultare Distribuisci funzioni Lambda per Go con gli archivi di file .zip.

Modello AWS SAM per un'applicazione DynamoDB

È possibile creare l'applicazione utilizzando AWS SAM. Per ulteriori informazioni sulla creazione di modelli AWS SAM, consulta Nozioni di base sui modelli AWS SAM nella Guida per gli sviluppatori di AWS Serverless Application Model.

Di seguito è riportato un modello AWS SAM di esempio per l'applicazione del tutorial. Copiare il testo seguente in un file .yaml e salvarlo insieme al pacchetto ZIP creato in precedenza. Si noti che i valori dei parametri `Handler` e `Runtime` devono corrispondere a quelli utilizzati nella sezione precedente per creare la funzione.

Example template.yaml

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

```
Resources:
  ProcessDynamoDBStream:
    Type: AWS::Serverless::Function
    Properties:
      Handler: handler
      Runtime: runtime
      Policies: AWSLambdaDynamoDBExecutionRole
      Events:
        Stream:
          Type: DynamoDB
          Properties:
            Stream: !GetAtt DynamoDBTable.StreamArn
            BatchSize: 100
            StartingPosition: TRIM_HORIZON

  DynamoDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: id
          AttributeType: S
      KeySchema:
        - AttributeName: id
          KeyType: HASH
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5
      StreamSpecification:
        StreamViewType: NEW_IMAGE
```

Per informazioni su come creare il pacchetto e distribuire l'applicazione serverless utilizzando i comandi di creazione di pacchetti e distribuzione, consultare Distribuzione delle applicazioni serverless nella Guida per gli sviluppatori di AWS Serverless Application Model.

In che modo Lambda elabora i record di Amazon Kinesis Data Streams

È possibile utilizzare 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 di Kinesis Data Streams (iteratore standard) o a un consumer a throughput dedicato con fan-out migliorato. Per gli iteratori standard, Lambda esegue il polling di ogni partizione nel flusso Kinesis per i record utilizzando il protocollo HTTP. La mappatura dell'origine eventi condivide il throughput di lettura con altri utenti della partizione.

Per informazioni dettagliate sui flussi di dati Kinesis, consulta Lettura dei dati da Amazon Kinesis Data Streams.

> **Note**
>
> Kinesis addebita dei costi per ogni partizione, 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 registri da una singola partizione/flusso dei dati.

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**
>
> Le mappature delle sorgenti degli eventi 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 ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

Configura l'ParallelizationFactorimpostazione per elaborare uno shard di un flusso di dati Kinesis con più di una chiamata Lambda contemporaneamente. È possibile specificare il numero di batch simultanei di cui Lambda esegue il polling da una partizione da un fattore di parallelizzazione compreso tra da 1 (predefinito) e 10. Ad esempio, impostando `ParallelizationFactor` su 2, puoi disporre al massimo di 200 invocazioni Lambda simultanee per elaborare 100

partizioni di dati Kinesis (anche se nella pratica potresti vedere valori differenti per il parametro `ConcurrentExecutions`). Ciò permette di dimensionare verso l'alto il throughput di elaborazione quando il volume dei dati è volatile e l'`IteratorAge` è alta. Quando si aumenta il numero di batch simultanei per shard, Lambda garantisce comunque l'elaborazione in ordine a livello di chiave di partizione.

È possibile utilizzarlo anche `ParallelizationFactor` con l'aggregazione Kinesis. Il comportamento della mappatura delle sorgenti 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": "VGhpcyBpcyBvbmx5IGEgdGVzdC4=",
                "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"
        }
    ]
}
```

Elaborazione dei record di Amazon Kinesis Data Streams con Lambda

Per elaborare i record di Amazon Kinesis Data Streams con Lambda, crea un consumatore per il tuo stream e quindi crea una mappatura delle sorgenti degli eventi Lambda.

Configurazione del flusso di dati e della funzione

La funzione Lambda è un'applicazione consumer per il flusso di dati. Elabora un batch di record alla volta da ciascuna partizione. 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.

- Iteratore standard: Lambda analizza ogni shard del flusso Kinesis alla ricerca di record con una frequenza 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 della partizione.

- Fan-out migliorato: per ridurre al minimo la latenza e massimizzare la velocità di lettura, crea un fruitore di flussi di dati con fan-out migliorato. I consumatori del fan-out avanzato ottengono una connessione dedicata a ciascuna partizione che non ha conseguenze su altre applicazioni che leggono dal flusso. I consumatori del flusso utilizzano HTTP/2 per ridurre la latenza spingendo record da Lambda a long-lived su una connessione di lunga durata e comprimendo le intestazioni della richiesta. 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 tua funzione elabora i record, aggiungi degli shard al tuo flusso di dati. Lambda elabora i record in ogni partizione in ordine. Interrompe l'elaborazione di record aggiuntivi in una partizione 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.

Crea una mappatura dell'origine degli eventi per richiamare una funzione Lambda

Per richiamare la funzione Lambda con i record del flusso di dati, crea una 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 provenienti da più flussi, ogni batch contiene i record di un solo shard o stream.

È possibile configurare le mappature delle sorgenti degli eventi per elaborare i record di un flusso in un altro. Account AWS Per ulteriori informazioni, consulta the section called “Mappature tra account”.

Prima di creare una mappatura dell'origine degli eventi, devi autorizzare la funzione Lambda a leggere da un flusso di dati Kinesis. Lambda necessita delle seguenti autorizzazioni per gestire le risorse relative al flusso di dati Kinesis:

- kinesis: DescribeStream
- kinesis: Riepilogo DescribeStream
- kinesis: GetRecords
- kinesis: iteratore GetShard
- cinesi: ListShards
- cinesi: ListStreams
- cinesi: Shard SubscribeTo

La politica AWS gestita AWSLambdaKinesisExecutionRoleinclude queste autorizzazioni. Aggiungi questa politica gestita alla tua funzione come descritto nella procedura seguente.

AWS Management Console

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, seleziona Autorizzazioni.
3. Nel riquadro Ruolo di esecuzione, in Nome ruolo, scegli il link al ruolo di esecuzione della tua funzione. Questo link apre la pagina relativa a quel ruolo nella console IAM.
4. Nel riquadro Politiche di autorizzazione, scegli Aggiungi autorizzazioni, quindi seleziona Allega politiche.
5. Inserisci **`AWSLambdaKinesisExecutionRole`** nel campo di ricerca.
6. Seleziona la casella di controllo accanto alla politica e 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, aggiungi la `Policies` proprietà come mostrato nell'esempio seguente:

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./my-function/
      Handler: index.handler
      Runtime: nodejs20.x
      Policies:
        - AWSLambdaKinesisExecutionRole
```

Dopo aver configurato le autorizzazioni richieste, create la mappatura della fonte dell'evento.

AWS Management Console

Per creare la mappatura delle sorgenti 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 Trigger, per l'origine, seleziona Kinesis.
4. Seleziona lo stream Kinesis per il quale desideri creare la mappatura delle sorgenti degli eventi e, facoltativamente, un utente del tuo stream.
5. (Facoltativo) modifica la dimensione del batch, la posizione iniziale e la finestra Batch per la mappatura della fonte dell'evento.
6. Scegli Aggiungi.

Quando crei la mappatura delle sorgenti degli eventi dalla console, il tuo ruolo IAM deve disporre delle autorizzazioni kinesis: ListStreams e kinesis: Consumers. ListStream

AWS CLI

Per creare la mappatura delle sorgenti degli eventi Kinesis

- Esegui il seguente comando CLI per creare una mappatura delle sorgenti degli eventi Kinesis. Scegliete la dimensione del batch e la posizione iniziale in base al vostro 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, aggiungete l'`--maximum-batching-window-in-seconds`opzione. Per ulteriori informazioni sull'utilizzo di questo e di altri parametri, vedete create-event-source-mapping nel Command Reference.AWS CLI

AWS SAM

Per creare la mappatura delle sorgenti degli eventi Kinesis

- Nella definizione della funzione, aggiungi la `KinesisEvent` proprietà come mostrato nell'esempio seguente:

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./my-function/
      Handler: index.handler
      Runtime: nodejs20.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 ulteriori informazioni sulla creazione di una mappatura delle sorgenti di eventi per Kinesis Data Streams in AWS SAM, consulta Kinesis nella Developer Guide.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 una mappatura delle sorgenti degli eventi tra più account

Amazon Kinesis Data Streams supporta policy basate sulle risorse. Per questo motivo, puoi elaborare i dati inseriti in un flusso 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 un flusso Kinesis in un Account AWS altro, devi configurare il flusso 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 su più account, consulta Condivisione dell'accesso con AWS Lambda funzioni su più account nella guida per sviluppatori di Amazon Kinesis Streams.

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 utilizzando uno dei metodi descritti nella sezione precedente.

Se scegli di creare la mappatura delle sorgenti degli eventi utilizzando la console Lambda, incolla l'ARN del tuo stream direttamente nel campo di input. Se desideri specificare un consumatore per il tuo stream, incollando l'ARN del consumatore viene compilato automaticamente il campo stream.

Configurazione della risposta batch parziale con Kinesis Data Streams 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 successi parziali può contribuire a ridurre il numero di tentativi su un record, anche se non impedisce del tutto la possibilità di tentativi in un record riuscito.

Per attivarlo`ReportBatchItemFailures`, includi il valore **`ReportBatchItemFailures`** enum nell'elenco dei tipi. FunctionResponse Questo elenco indica quali tipi di risposta sono abilitati per la funzione. È possibile configurare questo elenco quando si crea o si aggiorna una mappatura della sorgente degli eventi.

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.

Di seguito sono riportati alcuni esempi di codice di funzione che restituiscono l'elenco degli ID dei messaggi con errori nel batch:

.NET

AWS SDK for .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.DefaultLambdaJsonSeri

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 relativi agli elementi batch di Kinesis con Lambda utilizzando 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 altro da sapere. 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 di errori relativi agli 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 degli errori relativi agli 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 altro da sapere. 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 relativi agli elementi batch di Kinesis con Lambda utilizzando 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 sapere. 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 Kinesis con Lambda utilizzando 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
}
```

Conserva i record batch scartati per un'origine di eventi Kinesis Data Streams in Lambda

La gestione degli errori per le mappature delle sorgenti 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 delle sorgenti degli eventi (secondi). MaximumRecord AgeIn
- Durante la chiamata: se la funzione viene richiamata ma restituisce un errore, Lambda riprova fino alla scadenza dei record, al superamento dell'età massima (MaximumRecordAgeInsecondi) o al raggiungimento della quota di tentativi configurata (Tentments). MaximumRetry Per gli errori di funzione, puoi anche configurare BisectBatchOnFunctionError, che divide un batch non riuscito in due batch più piccoli, 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 sulla partizione 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 con metadati sulla chiamata non riuscita. Puoi configurare qualsiasi argomento Amazon SNS o coda Amazon SQS come destinazione. Il tuo ruolo di esecuzione deve avere le autorizzazioni per la destinazione:

- Per le destinazioni SQS: sqs: SendMessage
- Per le destinazioni SNS: sns:publish

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. Scegliere Save (Salva).

Puoi anche configurare una destinazione in caso di errore utilizzando (). AWS Command Line Interface AWS CLI Ad esempio, il seguente comando create-event-source-mapping 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 seguente comando update-event-source-mapping 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. Utilizzate 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": ""}}'
```

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 `endSequenceNumber` campi`streamArn`, `shardId``startSequenceNumber`, e per ottenere il record originale completo.

```
{
    "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.

Implementazione dell'elaborazione Kinesis Data Streams con stato 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. Lo stato può essere un massimo di 1 MB per partizione. Se supera quella dimensione, Lambda termina la finestra in anticipo.

Ogni record in un flusso appartiene a una finestra specifica. Lambda elaborerà 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-000000000006",
    "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, specifica la finestra in secondi (). TumblingWindowInSeconds Il seguente comando di esempio AWS Command Line Interface (AWS CLI) crea una mappatura della sorgente degli eventi in streaming con una finestra di rotazione 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 avvieranno la propria

finestra in uno stato nuovo. Quando non vengono aggiunti nuovi record alla finestra corrente, Lambda attende fino a 2 minuti prima di presumere che la finestra sia finita. Questo 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 le mappature delle sorgenti degli eventi di Amazon Kinesis Data Streams

Tutte le mappature delle sorgenti degli eventi Lambda condividono le stesse CreateEventSourceMappingoperazioni e quelle dell'API. UpdateEventSourceMapping Tuttavia, solo alcuni dei parametri si applicano a Kinesis.

Parametri dell'origine eventi applicabili a Kinesis

Parametro	Obbligatorio	Predefinito	Note
BatchSize	N	100	Massimo: 10.000.
BisectBatchOnFunctionErrore	N	false	
DestinationConfig	N		Destinazione della coda Amazon SQS o dell'argomento Amazon SNS per i record scartati. Per ulteriori informazioni, consulta Configurazione delle destinazioni per le chiamate non riuscite.
Enabled (Abilitato)	N	true	
EventSourceArn	Y		L'ARN del flusso dei dati o di un consumer di flusso.
FunctionName	Y		
FunctionResponseTipi	N		Per consentire alla funzione di segnalare errori specifici in un batch, includi il valore `ReportBatchItemFailures` in `FunctionResponseTypes`. Per ulteriori informazioni, consulta Configurazione

Parametro	Obbligatorio	Predefinito	Note
			della risposta batch parziale con Kinesis Data Streams e Lambda.
MaximumBatchingWindowInSecondi	N	0	
MaximumRecordAgeInSecondi	N	-1	-1 significa infinito: Lambda non elimina i record (le impostazioni di conservazione dei dati di Kinesis Data Streams sono ancora valide) Minimo: -1 Massimo: 604.800
MaximumRetryTentativi	N	-1	-1 sta per infinito: i record non riusciti vengono ripetuti fino alla scadenza del record Minimo: -1 Massimo: 10.000.
ParallelizationFactor	N	1	Maximum: 10
StartingPosition	Y		AT_TIMESTAMP, TRIM_HORIZON o LATEST

Parametro	Obbligatorio	Predefinito	Note
StartingPositionTimestamp	N		Valido solo se impostato su StartingPosition AT_TIMESTAMP. Il tempo da cui avviare la lettura, in secondi di tempo Unix.
TumblingWindowInSeconds	N		Minimo: 0 Massimo: 900

Tutorial: Utilizzo di Lambda con Kinesis Data Streams

In questo tutorial, crei una funzione Lambda per consumare eventi da un flusso di dati Amazon Kinesis.

1. L'app personalizzata scrive record nel flusso.
2. AWS Lambda interroga lo 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

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 l'AWS Command Line Interface (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 del ruolo di esecuzione

Crea il ruolo di esecuzione che autorizza la funzione ad accedere alle risorse. AWS

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 AWSLambdaKinesisExecutionRolepolicy dispone delle autorizzazioni necessarie alla funzione per leggere gli elementi da Kinesis e scrivere i log in CloudWatch Logs.

Creazione della funzione

Crea una funzione Lambda che elabora i messaggi Kinesis. Il codice funzione registra l'ID dell'evento e i dati dell'evento del record CloudWatch Kinesis in Logs.

Questo tutorial utilizza il runtime Node.js 18.x, ma è fornito anche un 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 scheda. JavaScript

.NET

AWS SDK for .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.DefaultLambdaJsonSeri

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.

Consumo di un evento Kinesis con Lambda utilizzando 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 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 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.

Consumo di un evento Kinesis con Lambda utilizzando 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.

Consumo di un evento Kinesis con Lambda utilizzando 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 nodejs18.x \
--role arn:aws:iam::111122223333:role/lambda-kinesis-role
```

Test della funzione Lambda

Richiama la funzione Lambda manualmente 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-000000000006:49590338271490256608559692538361571095921575989136588898",
            "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'cli-binary-formatopzione è obbligatoria se utilizzi 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
```

Eseguire 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 un'origine 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 valore `--data` è una stringa che il 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 CloudWatch Logs. 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 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 **`delete`** nel campo di immissione testo e scegli Delete (Elimina).

Per eliminare il flusso Kinesis

1. Accedi AWS Management Console 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 Amazon MQ

> **Note**
>
> Se desideri inviare dati a una destinazione diversa da una funzione Lambda o arricchire i dati prima di inviarli, consulta Amazon Pipes. EventBridge

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.

> **Warning**
>
> Le mappature delle sorgenti degli eventi 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 ulteriori informazioni, consulta 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:

- Concorrenza: le funzioni Lambda che utilizzano uno strumento di mappatura dell'origine degli eventi Amazon MQ hanno un'impostazione di concorrenza massima predefinita. Per ActiveMQ, il servizio Lambda limita il numero di ambienti di esecuzione simultanei a cinque. Per RabbitMQ, il numero di ambienti di esecuzione simultanei è limitato a 1. 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 simultaneità massima predefinita, contatta AWS Support.
- 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 il plug-in SimpleAuthentication ActiveMQ. 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 “Configurazione della rete” più avanti in questo argomento.
- 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 implementazione di broker o cluster a istanza singola per ogni strumento di mappatura dell'origine degli eventi. I broker a istanza singola 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 OpenWire il protocollo /Java Message Service (JMS). 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, vedere OpenWireil 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.

Sezioni

Gruppo di utenti Lambda

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 simultaneo di una determinata funzione utilizzando la `ConcurrentExecutions` metrica in Amazon. 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-
west-2:111122223333:broker:test:b-9bcfa592-423a-4942-879d-eb284b418fc8",
   "messages": [
      {
        "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-
west-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":"QUJDOkFBQUE=",
        "timestamp": 1598827811958,
        "brokerInTime": 1598827811958,
        "brokerOutTime": 1598827811959,
        "properties": {
          "index": "1",
          "doAlarm": "false",
          "myCustomProperty": "value"
        }
      },
```

```
      {
        "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-
west-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-
west-2:111122223333:broker:pizzaBroker:b-9bcfa592-423a-4942-879d-eb284b418fc8",
  "rmqMessagesByQueue": {
    "pizzaQueue::/": [
      {
        "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=="
      }
    ]
  }
}
```

> **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::/`.

Autorizzazioni del ruolo di esecuzione

Per leggere i record da un broker Amazon MQ, la funzione Lambda richiede le seguenti autorizzazioni aggiunte al suo ruolo di esecuzione:

- mq: DescribeBroker
- secretsmanager: Valore GetSecret
- ec2: Interfaccia CreateNetwork
- ec2: Interfaccia DeleteNetwork
- ec2: Interfacce DescribeNetwork
- ec2: Gruppi DescribeSecurity
- ec2: DescribeSubnets
- ec2: DescribeVpcs
- registri: Gruppo CreateLog
- registri: Stream CreateLog
- registri: Eventi PutLog

> **Note**
>
> Quando utilizzi una chiave gestita dal cliente crittografata, aggiungi anche l'autorizzazione `kms:Decrypt`.

Configurazione della rete

Per concedere a Lambda l'accesso completo al tuo broker tramite lo strumento di mappatura dell'origine degli eventi, il broker deve utilizzare un endpoint pubblico (indirizzo IP pubblico) oppure devi fornire l'accesso all'Amazon VPC in cui hai creato il broker.

Per impostazione predefinita, quando crei un broker Amazon MQ, il flag `PubliclyAccessible` è impostato su false. Affinché il tuo broker riceva un indirizzo IP pubblico, devi impostare il flag `PubliclyAccessible` su true.

La migliore pratica per usare Amazon MQ con Lambda consiste nell'utilizzare gli endpoint AWS PrivateLink VPC e consentire alla funzione Lambda di accedere al VPC del broker. Implementa un endpoint per Lambda e, solo per ActiveMQ, un endpoint per (). AWS Security Token Service AWS STS Se il tuo broker utilizza l'autenticazione, implementa anche un endpoint per. AWS Secrets Manager Per ulteriori informazioni, consulta the section called “Uso di endpoint VPC”.

In alternativa, configura un gateway NAT su ogni sottorete pubblica nel VPC contenente il tuo broker Amazon MQ. Per ulteriori informazioni, consulta the section called “Accesso a Internet per le funzioni VPC”.

Quando crei una mappatura dell'origine degli eventi per un broker Amazon MQ, Lambda verifica se le interfacce di rete elastiche (ENI) sono già presenti per le sottoreti e i gruppi di sicurezza del VPC del broker. Se Lambda trova ENI esistenti, tenta di riutilizzarli. Altrimenti, Lambda crea nuovi 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. Questi VPC vengono gestiti automaticamente dal servizio e non sono visibili ai clienti. Puoi anche connettere la tua funzione a un Amazon VPC. In entrambi i casi, la configurazione VPC della funzione non influisce sulla mappatura delle sorgenti degli eventi. Solo la configurazione del VPC dell'origine dell'evento determina il modo in cui Lambda si connette alla fonte dell'evento.

Regole del gruppo di sicurezza VPC

Configura i gruppi di sicurezza per l'Amazon VPC contenente il tuo cluster con le seguenti regole (come minimo):

- Regole in entrata: consenti tutto il traffico sulla porta del broker Kafka per il gruppo di sicurezza specificato per l'origine eventi dall'interno del gruppo di sicurezza. Per impostazione predefinita, ActiveMQ utilizza la porta 61.617 e RabbitMQ utilizza la porta 5.671.

- Regole in uscita: consenti tutto il traffico sulla porta 443 per tutte le destinazioni. Consenti tutto il traffico sulla porta del broker all'interno del proprio gruppo di sicurezza. Per impostazione predefinita, ActiveMQ utilizza la porta 61.617 e RabbitMQ utilizza la porta 5.671.
- Se si utilizzano endpoint VPC anziché gateway NAT, i gruppi di sicurezza associati agli endpoint VPC devono consentire tutto il traffico in entrata sulla porta 443 dai gruppi di sicurezza dell'origine eventi.

Uso di endpoint VPC

Quando utilizzi gli endpoint VPC, le chiamate API per richiamare la tua funzione vengono instradate attraverso questi endpoint utilizzando gli ENI. Il principale del servizio Lambda deve richiamare tutte `lambda:InvokeFunction` le funzioni che utilizzano tali ENI. Inoltre, per ActiveMQ, il responsabile del servizio Lambda deve `sts:AssumeRole` ricorrere a ruoli che utilizzano gli ENI.

Per impostazione predefinita, gli endpoint VPC dispongono di policy IAM aperte. La migliore pratica consiste nel limitare queste policy per consentire solo a soggetti specifici di eseguire le azioni necessarie utilizzando quell'endpoint. Per garantire che la mappatura delle sorgenti degli eventi sia in grado di richiamare la funzione Lambda, la policy degli endpoint VPC deve consentire al principio del servizio Lambda di chiamare e, per ActiveMQ,. `lambda:InvokeFunction` `sts:AssumeRole` Limitare le policy degli endpoint VPC per consentire solo le chiamate API provenienti dall'organizzazione impedisce il corretto funzionamento della mappatura delle sorgenti degli eventi.

L'esempio seguente di policy degli endpoint VPC mostra come concedere l'accesso richiesto per gli endpoint Lambda e per gli endpoint AWS STS Lambda.

Example Policy degli endpoint VPC - endpoint (solo AWS STS ActiveMQ)

```
{
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
```

```
        }
    ]
}
```

Example Politica degli endpoint VPC - Endpoint Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Se il tuo broker Amazon MQ utilizza l'autenticazione, puoi anche limitare la policy degli endpoint VPC per l'endpoint Secrets Manager. Per chiamare l'API Secrets Manager, Lambda utilizza il ruolo della funzione, non il responsabile del servizio Lambda. L'esempio seguente mostra una policy per gli endpoint di Secrets Manager.

Example Politica degli endpoint VPC - Endpoint Secrets Manager

```
{
    "Statement": [
        {
            "Action": "secretsmanager:GetSecretValue",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "customer_function_execution_role_arn"
                ]
            },
            "Resource": "customer_secret_arn"
        }
    ]
}
```

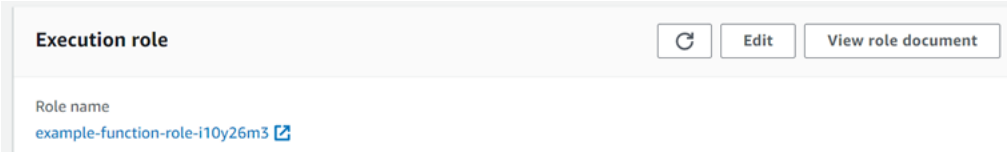
Aggiungi le autorizzazioni e crea la mappatura della fonte dell'evento

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.

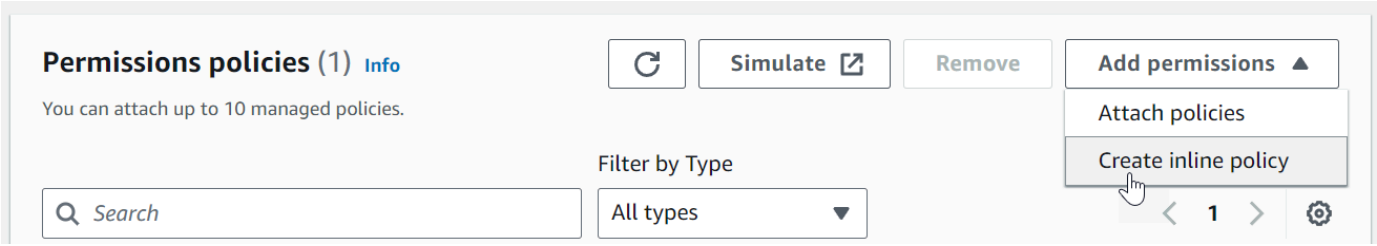
Per configurare la tua funzione per la lettura da Amazon MQ, aggiungi le autorizzazioni richieste e crea un trigger MQ nella console Lambda.

Per aggiungere 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 link apre il ruolo nella console IAM.



5. Scegli Aggiungi autorizzazioni, quindi scegli Crea politica in linea.



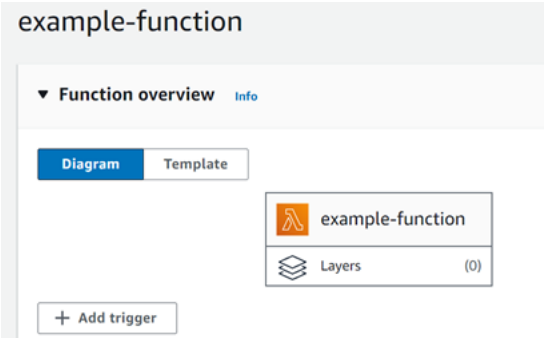
6. Nell'editor delle politiche, scegli JSON. Immetti la seguente policy. La tua funzione necessita di queste autorizzazioni per la lettura da un broker Amazon MQ.

```
{
  "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 crittografata gestita dal cliente, devi aggiungere anche l'`kms:Decrypt`autorizzazione.

7. Seleziona Successivo. Inserisci il nome di una politica, quindi scegli Crea politica.
8. Torna alla tua funzione nella console Lambda. In Panoramica delle funzioni, scegliere Aggiungi trigger.



9. Scegli 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.

Aggiorna la mappatura della fonte dell'evento

Utilizzate il `update-event-source-mapping` comando per aggiornare una mappatura della sorgente degli eventi. Il comando di esempio seguente aggiorna un mapping di origine eventi in modo da avere una dimensione batch pari a 2.

```
aws lambda update-event-source-mapping \
--uuid 91eaeb7e-c976-1234-9451-8709db01f137 \
--batch-size 2
```

Verrà visualizzato l'output seguente:

```
{
    "UUID": "91eaeb7e-c976-1234-9451-8709db01f137",
    "BatchSize": 2,
    "EventSourceArn": "arn:aws:mq:us-east-1:123456789012:broker:ExampleMQBroker:b-
b4d492ef-bdc3-45e3-a781-cd1a3102ecca",
    "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:MQ-Example-
Function",
    "LastModified": 1601928393.531,
    "LastProcessingResult": "No records processed",
    "State": "Updating",
    "StateTransitionReason": "USER_INITIATED"
}
```

Lambda aggiorna queste impostazioni in modo asincrono. L'output non rifletterà le modifiche fino al completamento di questo processo. Per visualizzare lo stato corrente della risorsa, utilizza il comando `get-event-source-mapping`.

```
aws lambda get-event-source-mapping \
--uuid 91eaeb7e-c976-4939-9451-8709db01f137
```

Verrà visualizzato l'output seguente:

```
{
    "UUID": "91eaeb7e-c976-4939-9451-8709db01f137",
    "BatchSize": 2,
    "EventSourceArn": "arn:aws:mq:us-east-1:123456789012:broker:ExampleMQBroker:b-
b4d492ef-bdc3-45e3-a781-cd1a3102ecca",
    "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:MQ-Example-
Function",
    "LastModified": 1601928393.531,
    "LastProcessingResult": "No records processed",
    "State": "Enabled",
    "StateTransitionReason": "USER_INITIATED"
}
```

Errori della mappatura dell'origine eventi

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 politica con i valori predefiniti della pagina Politica di riconsegna sul sito Web di Apache ActiveMQ, impostata su 6. `maximumRedeliveries`

Parametri di configurazione di Amazon MQ e RabbitMQ

Tutti i tipi di sorgenti di eventi Lambda condividono le stesse operazioni CreateEventSourceMappinge quelle dell'UpdateEventSourceMappingAPI. Tuttavia, solo alcuni dei parametri si applicano ad Amazon MQ e RabbitMQ.

Parametri di origine dell'evento applicabili ad Amazon MQ e RabbitMQ

Parametro	Obbligatorio	Predefinito	Note
BatchSize	N	100	Massimo: 10.000
Abilitato	N	true	
FunctionName	Y		
FilterCriteria	N		Filtro eventi Lambda
MaximumBatchingWindowInSecondi	N	500 ms	Comportamento di batching
Queues	N		Il nome della coda di destinazione del broker Amazon MQ da utilizzare.
SourceAccessConfigurazioni	N		Per ActiveMQ, le credenziali BASIC_AUTH. Per RabbitMQ,

Parametro	Obbligatorio	Predefinito	Note
			può contenere sia le credenzia li BASIC_AUTH che le informazioni VIRTUAL_HOST.

Uso di Lambda con Amazon MSK

> **Note**
>
> Se desideri inviare dati a una destinazione diversa da una funzione Lambda o arricchire i dati prima di inviarli, consulta Amazon Pipes. EventBridge

Amazon Managed Streaming for Apache Kafka (Amazon MSK) è un servizio completamente gestito che consente di creare ed eseguire applicazioni che utilizzano Apache Kafka per elaborare i dati in streaming. Amazon MSK semplifica la configurazione, il dimensionamento e la gestione dei cluster che eseguono 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). Amazon MSK supporta più versioni open-source di Kafka.

Amazon MSK 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 Comportamento di batching.

> **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.

Lambda legge i messaggi in sequenza per ogni partizione. Un singolo payload Lambda può contenere messaggi provenienti da più partizioni. 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.

> **Warning**
>
> Le mappature delle sorgenti degli eventi 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 ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

Per un esempio di come configurare Amazon MSK come origine di eventi, consulta Using Amazon MSK come origine di eventi per AWS Lambda sul AWS Compute Blog. Consulta Integrazione di Amazon MSK Lambda nei laboratori Amazon MSK per un tutorial completo.

Argomenti

- Tutorial: utilizzo di una mappatura delle sorgenti di eventi Amazon MSK per richiamare una funzione Lambda
- Esempio di evento
- Autenticazione cluster MSK
- Gestione dell'accesso e delle autorizzazioni API
- Errori di autenticazione e autorizzazione
- Configurazione della rete
- Aggiunta di Amazon MSK come origine eventi
- Creazione di strumenti di mappatura dell'origine degli eventi multi-account
- Destinazioni in caso di errore
- Scalabilità automatica dell'origine eventi Amazon MSK
- Posizioni di partenza di polling e flussi
- CloudWatch Metriche Amazon
- Parametri di configurazione di Amazon MSK

Tutorial: utilizzo di una mappatura delle sorgenti di eventi Amazon MSK per richiamare una funzione Lambda

In questo tutorial, eseguirai 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 una mappatura delle sorgenti 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, potrai configurare una funzione Lambda per elaborare tali eventi automaticamente con il tuo codice Lambda personalizzato.

Cosa puoi fare con questa funzionalità?

Soluzione di esempio: utilizzate una mappatura delle fonti di eventi MSK per fornire risultati in tempo reale ai vostri clienti.

Considerate lo scenario seguente: la vostra 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 di 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 che sviluppi. Avete deciso il seguente approccio di progettazione: le vostre applicazioni client comunicheranno con un backend serverless ospitato in. AWS I client si connetteranno tramite sessioni websocket utilizzando l'API Amazon WebSocket API Gateway.

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 API. Puoi implementare questo componente con AWS Lambda, fornendo la tua logica personalizzata in una funzione Lambda, quindi chiamandolo con una mappatura dell'origine degli eventi AWS Lambda Amazon MSK.

Per ulteriori informazioni sull'implementazione di soluzioni utilizzando l'API Amazon WebSocket API Gateway, 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 seguire la sezione Guida introduttiva all'uso di Amazon MSK nella documentazione di Amazon MSK.

- Un cluster Amazon MSK. Vedi 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 migliora la sicurezza limitando la funzione Lambda all'accesso solo alle risorse Amazon MSK necessarie. Questa opzione è abilitata per impostazione predefinita sui 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 opzione è abilitata per impostazione predefinita sui nuovi cluster Amazon MSK.
- Un argomento di Kafka nel tuo cluster Amazon MSK da utilizzare per questa soluzione. Vedi Creare un argomento in Guida introduttiva a usare 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 Amazon MSK IAM installate. Vedi Creare una macchina client in Guida introduttiva a usare Amazon MSK.

Dopo aver configurato queste risorse, raccogli le seguenti informazioni dal tuo AWS account per confermare che sei pronto a continuare.

- Il nome del tuo cluster Amazon MSK. Puoi 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 tuo cluster Amazon MSK. Puoi trovare queste informazioni nella console Amazon MSK. Nei passaggi seguenti, fai riferimento a questi come ai tuoi *`clusterSecurityGroups`*.

- L'id dell'Amazon VPC contenente il tuo 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 utilizzato nella tua soluzione. Puoi trovare queste informazioni chiamando il tuo cluster Amazon MSK con la `topics` CLI di Kafka dal tuo host di amministrazione Kafka. Per ulteriori informazioni sugli argomenti CLI, vedere Aggiungere e rimuovere argomenti nella documentazione di Kafka.
- Il nome di un gruppo di consumatori per l'argomento Kafka, adatto all'uso con la funzione Lambda. Questo gruppo può essere creato automaticamente da Lambda, quindi non è necessario crearlo con la CLI di Kafka. Se devi gestire i tuoi gruppi di consumatori, per saperne di più sulla CLI dei gruppi di consumatori, consulta Managing Consumer Groups nella documentazione di Kafka.

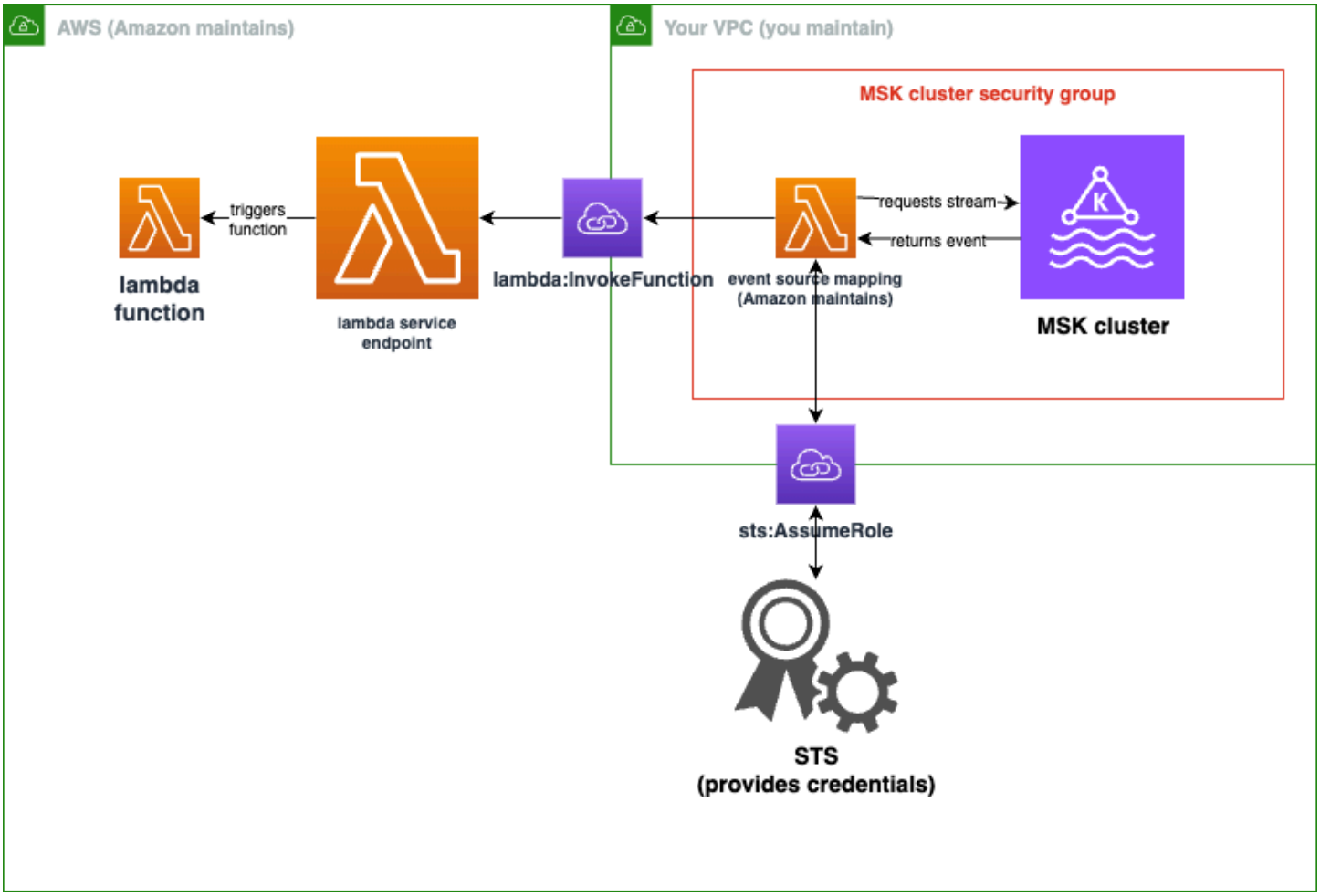
Le seguenti autorizzazioni nel tuo account: AWS

- Autorizzazione a creare e gestire una funzione Lambda.
- Autorizzazione a 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.

Configura la connettività di rete per consentire a Lambda di comunicare con Amazon MSK

AWS PrivateLink Usalo per connettere Lambda e Amazon MSK. Puoi farlo creando endpoint Amazon VPC di interfaccia nella console Amazon VPC. Per ulteriori informazioni sulla configurazione di rete, consulta. the section called “Configurazione della rete”

Quando una mappatura delle sorgenti di eventi 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.



La mappatura delle sorgenti 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 la mappatura delle sorgenti 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 alla mappatura delle sorgenti degli eventi di comunicare con il cluster Amazon MSK.

Puoi configurare i seguenti passaggi utilizzando. AWS Management Console

Per configurare l'interfaccia (endpoint Amazon VPC) per connettere Lambda e Amazon MSK

1. *Crea un gruppo di sicurezza per la tua interfaccia Amazon VPC endpoint, endpoint SecurityGroup, che consenta il traffico TCP in entrata su 443 dal cluster. SecurityGroups* 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 del *cluster SecurityGroups*. Per ogni regola:

- Per Tipo, seleziona HTTPS.
- Per Origine, seleziona uno dei *cluster SecurityGroups*.

2. Crea un endpoint che collega il servizio Lambda 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 servizio, seleziona`com.amazonaws.`*regionName*`.lambda`, dove *RegionName* ospita la funzione Lambda.
   - Per VPC, seleziona Amazon VPC contenente il tuo cluster Amazon MSK.
   - Per i gruppi di sicurezza, seleziona l'*endpoint SecurityGroup*, 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 che il nome Enable DNS rimanga impostato.

3. Crea un endpoint che collega 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, selezionare AWS STS.
   - Per VPC, seleziona Amazon VPC contenente il tuo cluster Amazon MSK.
   - *Per i **gruppi di sicurezza, seleziona endpoint.** SecurityGroup*
   - 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 che il nome Enable DNS rimanga impostato.

4. Per ogni gruppo di sicurezza associato al tuo cluster Amazon MSK, ovvero all'interno del *cluster SecurityGroups*, consenti quanto segue:

   - Consenti tutto il traffico TCP in entrata e in uscita su 9098 verso tutto il *cluster SecurityGroups*, incluso il suo interno.
   - 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 modificare le regole del gruppo di sicurezza, segui le procedure in Aggiungere regole a un gruppo di sicurezza nella documentazione di Amazon EC2.

Aggiungi 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 Port range, fornisci 9098.
  - Per Source, fornisci uno dei *`cluster SecurityGroups`*.
- Per ogni regola in entrata per la porta 443, per Tipo, seleziona HTTPS.

Crea un ruolo IAM per Lambda da leggere dal tuo argomento Amazon MSK

Identifica i requisiti di autenticazione per Lambda da leggere nell'argomento di Amazon MSK, quindi definiscili in una policy. Crea un ruolo, *`lambda AuthRole`*, che autorizzi Lambda a utilizzare tali autorizzazioni. Autorizza le azioni sul tuo cluster Amazon MSK utilizzando azioni `kafka-cluster` IAM. Quindi, autorizza Lambda a eseguire le azioni di Amazon `kafka` MSK e Amazon EC2 necessarie per scoprire e connettersi al tuo cluster Amazon MSK, nonché le CloudWatch azioni in modo che Lambda possa registrare ciò che ha fatto.

Per descrivere i requisiti di autenticazione per Lambda da leggere da Amazon MSK

1. Scrivi un documento di policy IAM (un documento JSON), *`cluster AuthPolicy`*, che consenta a Lambda di leggere il tuo argomento Kafka nel tuo cluster Amazon MSK utilizzando il tuo gruppo di consumatori Kafka. Lambda richiede che durante la lettura sia impostato un gruppo di consumatori Kafka.

   Modifica il seguente modello per allinearlo ai tuoi prerequisiti:

```
{
    "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:region:account-id:cluster/mskClusterName/cluster-
uuid",
                "arn:aws:kafka:region:account-id:topic/mskClusterName/cluster-
uuid/mskTopicName",
                "arn:aws:kafka:region:account-id:group/mskClusterName/cluster-
uuid/mskGroupName"
            ]
        }
    ]
}
```

Per ulteriori informazioni, consultare. the section called “Autenticazione basata su ruoli IAM” Quando scrivi la tua politica:

- Per la *regione* e l'*ID account*, fornisci quelli che ospitano il tuo cluster Amazon MSK.
- Per *msk ClusterName*, fornisci il nome del tuo cluster Amazon MSK.
- Per *cluster-uuid, fornisci l'UUID* nell'ARN per il tuo cluster Amazon MSK.
- Per *msk TopicName*, fornisci il nome del tuo argomento Kafka.
- Per *msk GroupName*, fornisci il nome del tuo gruppo di consumatori Kafka.

2. Identifica Amazon MSK, Amazon EC2 CloudWatch e le autorizzazioni necessarie a Lambda per rilevare e connettere il tuo cluster Amazon MSK e registrare tali eventi.

   La policy `AWSLambdaMSKExecutionRole` gestita definisce in modo permissivo le autorizzazioni richieste. Usalo nei seguenti passaggi.

   In un ambiente di produzione, valuta se `AWSLambdaMSKExecutionRole` limitare la tua politica relativa al ruolo di esecuzione in base al principio del privilegio minimo, quindi scrivi una politica per il tuo ruolo che sostituisca questa politica gestita.

Per i dettagli sul linguaggio di policy IAM, consulta la documentazione IAM.

Ora che hai scritto il tuo documento sulla policy, crea una policy IAM in modo da poterla allegare al tuo ruolo. Puoi farlo utilizzando la console con la seguente procedura.

Per creare una policy IAM dal tuo documento di policy

1. Accedi AWS Management Console 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 il *`cluster AuthPolicy`*.
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 politiche IAM nella documentazione IAM.

Ora che disponi delle politiche IAM appropriate, crea un ruolo e assegnale ad esso. È possibile eseguire questa operazione utilizzando la console con la procedura seguente.

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. Seleziona Successivo.
6. Selezionare le seguenti policy:
   - *`ammasso AuthPolicy`*
   - `AWSLambdaMSKExecutionRole`
7. Seleziona Successivo.
8. Per Nome ruolo, inserisci *`lambda AuthRole`* e quindi scegli Crea ruolo.

Per ulteriori informazioni, consulta the section called “Ruolo di esecuzione (autorizzazioni per le funzioni di accedere ad altre risorse)”.

Crea una funzione Lambda da leggere dal tuo argomento Amazon MSK

Crea una funzione Lambda configurata per utilizzare il tuo ruolo IAM. Puoi creare la tua funzione Lambda utilizzando la console.

Per creare una funzione Lambda utilizzando la configurazione di autenticazione

1. Apri la console Lambda e seleziona Crea funzione dall'intestazione.
2. Scegli Crea da zero.
3. Per il 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 il ruolo di esecuzione predefinito.
6. Seleziona Usa un ruolo esistente.
7. Per Ruolo esistente, seleziona *`lambda AuthRole`*.

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 politiche al tuo ruolo, consulta Aggiungere o rimuovere le autorizzazioni di identità nella documentazione IAM.

Crea una mappatura della sorgente degli eventi sulla tua funzione Lambda

La mappatura delle sorgenti 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 la mappatura della sorgente dell'evento viene impostata automaticamente.

Per creare un trigger Lambda (e una mappatura dell'origine degli eventi)

1. Vai alla pagina di panoramica della tua funzione Lambda.
2. Nella sezione panoramica delle funzioni, scegli Aggiungi trigger in basso a sinistra.
3. Nel menu a discesa Seleziona una fonte, seleziona Amazon MSK.
4. Non impostare l'autenticazione.
5. Per il cluster MSK, seleziona il nome del cluster.
6. Per Dimensione del Batch, inserire 1. Questo passaggio semplifica il test di questa funzionalità e non rappresenta un valore ideale in fase di produzione.

7. Per il nome dell'argomento, inserisci il nome del tuo argomento Kafka.
8. Per l'ID del gruppo di consumatori, fornisci l'ID del tuo gruppo di consumatori Kafka.

Aggiorna la funzione Lambda per leggere i dati di streaming

Lambda fornisce informazioni sugli eventi di Kafka tramite il parametro event method. Per un esempio di struttura di un evento Amazon MSK, consultathe 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:

Node.js

```
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)
      })
    }
}
```

Puoi fornire il codice della funzione alla tua Lambda utilizzando la console.

Per aggiornare il codice della funzione Lambda

1. Vai alla pagina di panoramica della tua funzione Lambda.
2. Scegli la scheda Codice.
3. Inserisci il codice fornito nell'IDE del codice sorgente.
4. Nella barra di navigazione del codice sorgente, scegli Deploy.

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 controllando 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 dalla dimensione del batch per la mappatura dell'origine degli eventi definita nel passaggio precedente, altrimenti Lambda aspetterà che vengano richiamate ulteriori informazioni.
2. Se la funzione viene eseguita, Lambda scrive cosa è successo a. CloudWatch Nella console, vai alla pagina dei dettagli della tua funzione Lambda.
3. Selezionare la scheda Configurazione.
4. Dalla barra laterale, seleziona Strumenti operativi e di monitoraggio.
5. Identifica il gruppo di CloudWatch log in Configurazione della registrazione. Il gruppo di log dovrebbe iniziare con`/aws/lambda`. Scegli il link al gruppo di log.
6. Nella CloudWatch console, controlla gli eventi di registro per gli eventi di registro che Lambda ha inviato al flusso di log. Identifica se ci sono eventi di registro contenenti il messaggio del tuo evento Kafka, come nell'immagine seguente. In tal caso, hai collegato correttamente una funzione Lambda ad Amazon MSK con una mappatura della sorgente degli eventi 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 della partizione Amazon MSK, insieme a una data/ora e a un messaggio con codifica base64.

```
{
   "eventSource":"aws:kafka",
   "eventSourceArn":"arn:aws:kafka:sa-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
                  ]
               }
            ]
         }
      ]
   }
}
```

Autenticazione cluster MSK

Lambda ha bisogno dell'autorizzazione per accedere al cluster Amazon MSK, recuperare registri ed eseguire altri processi. Amazon MSK supporta diverse opzioni per il controllo dell'accesso del client al cluster MSK.

Opzioni di accesso al cluster

- Accesso non autenticato
- Autenticazione SASL/SCRAM
- Autenticazione basata su ruoli IAM
- Autenticazione TLS reciproca
- Configurazione del segreto mTLS
- Come Lambda sceglie un broker bootstrap

Accesso non autenticato

Se nessun client accede al cluster tramite Internet, è possibile utilizzare l'accesso non autenticato.

Autenticazione SASL/SCRAM

Amazon MSK supporta l'autenticazione SASL/SCRAM (Simple Authentication and Security Layer/Salted Challenge Response Authentication Mechanism) con crittografia Transport Layer Security (TLS). Per consentire a Lambda di connettersi al cluster, è necessario archiviare le credenziali di autenticazione (nome utente e password) in un luogo segreto. AWS Secrets Manager

Per ulteriori informazioni sull'uso di Secrets Manager, consulta Autenticazione nome utente e password con AWS Secrets Manager nella Guida per gli sviluppatori di Amazon Managed Streaming for Apache Kafka.

Amazon MSK non supporta l'autenticazione SASL/PLAIN.

Autenticazione basata su ruoli IAM

È possibile utilizzare IAM per autenticare l'identità dei client che si connettono al cluster MSK. Se l'autenticazione IAM è attiva sul tuo cluster MSK e non fornisci un segreto per l'autenticazione, Lambda utilizza automaticamente l'autenticazione IAM. Per creare e implementare policy basate su utenti o ruoli, utilizza l'API o la console IAM. Per ulteriori informazioni, consulta il controllo accessi IAM nella Guida per sviluppatori Amazon Managed Streaming for Apache Kafka.

Per consentire a Lambda di connettersi al cluster MSK, leggere i registri ed eseguire altre operazioni richieste, aggiungere le seguenti autorizzazioni al ruolo di esecuzione della funzione.

```
{
    "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:region:account-id:cluster/cluster-name/cluster-uuid",
                "arn:aws:kafka:region:account-id:topic/cluster-name/cluster-uuid/topic-name",
                "arn:aws:kafka:region:account-id:group/cluster-name/cluster-uuid/consumer-group-id"
            ]
        }
    ]
}
```

È possibile assegnare queste autorizzazioni a un cluster, un argomento e un gruppo specifici. Per ulteriori informazioni, consulta le operazioni Kafka di Amazon MSK nella Guida per sviluppatori Amazon Managed Streaming for Apache Kafka.

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.

Per Amazon MSK, Lambda funge da cliente. È possibile configurare un certificato client (come segreto in Secrets Manager) per autenticare Lambda con i broker nel cluster MSK. Il certificato client deve essere firmato da una CA nell'archivio trust del server. Il cluster MSK invia un certificato server a Lambda per autenticare i broker con Lambda. Il certificato del server deve essere firmato da un'autorità di certificazione (CA) presente nel AWS trust store.

Per istruzioni su come generare un certificato client, consulta Introduzione dell'autenticazione TLS reciproca per Amazon MSK come origine eventi.

Amazon MSK non supporta i certificati server autofirmati poiché tutti i broker di Amazon MSK utilizzano certificati pubblici firmati dalle autorità di certificazione di Amazon Trust Services, su cui Lambda fa affidamento per impostazione predefinita.

Per ulteriori informazioni, su mTLS per Amazon MSK, consulta Autenticazione TLS reciproca nella Guida per sviluppatori Amazon Managed Streaming for Apache Kafka.

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/+1HfgyE2KlmII36dg4IMzNjAFEBZiCRoPimO40s1cRqtFHXoal0QQbIlxk
cmUuiAii9R0=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFgjCCA2qgAwIBAgIQdjNZd6uFf9hbNC5RdfmHrzANBgkqhkiG9w0BAQsFADBb
...
rQoiowbbk5wXCheYSANQIfTZ6weQTgiCHCCbuuMKNVS95FkXm0vqVD/YpXKwA/no
c8PH3PSoAaRwMMgOSA2ALJvbRz8mpg==
-----END CERTIFICATE-----",
 "privateKey": "-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFKzBVBgkqhkiG9w0BBQ0wSDAnBgkqhkiG9w0BBQwwGgQUiAFcK5hT/X7Kjmgp
...
QrSekqF+kWzmB6nAfSzgO9IaoAaytLvNgGTckWeUkWn/V0Ck+LdGUXzAC4RxZnoQ
zp2mwJn2NYB7AZ7+imp0azDZb+8YG2aUCiyqb6PnnA==
-----END ENCRYPTED PRIVATE KEY-----"
}
```

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 mTLS 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 fornito per 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.

Gestione dell'accesso e delle autorizzazioni API

Oltre ad accedere al cluster Amazon MSK, la funzione necessita di autorizzazioni per eseguire varie operazioni dell'API Amazon MSK. Aggiungi queste autorizzazioni al ruolo di esecuzione della funzione. Se gli utenti hanno bisogno di accedere a una qualsiasi delle operazioni dell'API Amazon MSK, aggiungi le autorizzazioni richieste alla policy di identità per l'utente o il ruolo.

Puoi aggiungere manualmente ciascuna delle seguenti autorizzazioni al tuo ruolo di esecuzione. In alternativa, puoi allegare la policy AWS gestita AWSLambdaMSKExecutionRoleal tuo ruolo di esecuzione. La policy `AWSLambdaMSKExecutionRole` contiene tutte le azioni API e le autorizzazioni VPC richieste elencate di seguito.

Autorizzazioni del ruolo di esecuzione della funzione Lambda necessarie

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:

- log: Gruppo CreateLog
- registri: Stream CreateLog
- registri: Eventi PutLog

Affinché Lambda possa accedere al cluster Amazon MSK per tuo conto, la funzione Lambda deve disporre delle seguenti autorizzazioni per il ruolo di esecuzione:

- kafka: DescribeCluster
- caffè: V2 DescribeCluster
- kafka: Broker GetBootstrap

- kafka: DescribeVpc Connessione: richiesta solo per le mappature delle sorgenti degli eventi tra account.
- kafka: ListVpc Connessioni: non richiesto nel ruolo di esecuzione, ma richiesto per un principale IAM che sta creando una mappatura delle sorgenti degli eventi tra account.

Devi solo aggiungere uno dei seguenti: `kafka:DescribeCluster` o `kafka:DescribeClusterV2`. Sui cluster MSK con provisioning funzionano entrambe le autorizzazioni. Per i cluster MSK serverless è necessario utilizzare `kafka:DescribeClusterV2`.

> **Note**
>
> Lambda alla fine prevede di rimuovere l'autorizzazione `kafka:DescribeCluster` dalla policy associata gestita da `AWSLambdaMSKExecutionRole`. Se utilizzi questa policy, sarebbe opportuno migrare tutte le applicazioni tramite `kafka:DescribeCluster` in modo da utilizzare `kafka:DescribeClusterV2` al suo posto.

Autorizzazioni VPC

Se solo gli utenti all'interno di un VPC possono accedere al cluster Amazon MSK, la funzione Lambda deve disporre dell'autorizzazione ad 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. Queste autorizzazioni sono incluse nella politica AWSLambdaMSKExecutionRole AWS gestita.

- ec2: Interfaccia CreateNetwork
- ec2: Interfacce DescribeNetwork
- ec2: DescribeVpcs
- ec2: Interfaccia DeleteNetwork
- ec2: DescribeSubnets
- ec2: Gruppi DescribeSecurity

Autorizzazioni facoltative per la funzione Lambda

La funzione Lambda potrebbe richiedere autorizzazioni per:

- Accedi al tuo segreto SCRAM, se utilizzi l'autenticazione SASL/SCRAM.

- Descrivere il segreto di Secrets Manager.
- Accedi alla tua AWS Key Management Service (AWS KMS) chiave gestita dal cliente.
- 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 Amazon MSK, la tua funzione Lambda potrebbe aver bisogno dell'autorizzazione per accedere al tuo segreto SCRAM (se usi l'autenticazione SASL/SCRAM) o al segreto di Secrets Manager per decrittografare la chiave gestita dal cliente. AWS KMS Per accedere a queste risorse, il ruolo di esecuzione della funzione deve disporre delle seguenti autorizzazioni:

- kafka: Segreti ListScram
- secretsmanager: Valore GetSecret
- kms:Decrypt

Aggiunta di autorizzazioni al ruolo di esecuzione

Segui questi passaggi per aggiungere la policy AWS gestita AWSLambdaMSKExecutionRoleal tuo ruolo di esecuzione utilizzando la console IAM.

Per aggiungere una policy AWS gestita

1. Aprire la pagina Policies (Policy) nella console IAM.
2. Nella casella di ricerca inserisci il nome della policy (`AWSLambdaMSKExecutionRole`).
3. Seleziona la policy dall'elenco, quindi scegli Policy actions (Azioni delle policy), Attach (Allega).
4. Alla pagina Attach policy (Allega policy), seleziona il ruolo di esecuzione dall'elenco, quindi scegli Attach policy (Allega policy).

Concessione di accesso agli utenti con una policy IAM

Per impostazione predefinita, gli utenti e i ruoli non sono autorizzati a eseguire le operazioni API Amazon MSK. Per concedere l'accesso agli utenti dell'organizzazione o dell'account, è possibile aggiungere una policy basata sull'identità. Per ulteriori informazioni, consulta Esempi di policy basate sull'identità Amazon MSK nella Guida per gli sviluppatori di Amazon Managed Streaming for Apache Kafka.

Errori di autenticazione e autorizzazione

Se manca una delle autorizzazioni necessarie per consumare i dati dal cluster Amazon MSK, Lambda visualizza uno dei seguenti messaggi di errore nella mappatura delle origini degli eventi in Risultato. LastProcessing

Messaggi di errore

- Il cluster non è riuscito ad autorizzare Lambda
- Autenticazione SASL non riuscita
- Il server non è riuscito ad autenticare Lambda
- Il certificato o la chiave privata forniti non sono validi

Il cluster non è riuscito ad autorizzare Lambda

Per SASL/SCRAM o mTLS, questo errore indica che l'utente fornito non dispone di tutte le seguenti autorizzazioni della lista di controllo accessi Kafka (ACL) richieste:

- DescribeConfigs Cluster
- Descrivi il gruppo
- Leggi il gruppo
- Descrivi l'argomento
- Leggi l'argomento

Per il controllo accessi IAM, al ruolo di esecuzione della funzione manca una o più autorizzazioni necessarie per accedere al gruppo o all'argomento. Rivedi l'elenco delle autorizzazioni richieste in the section called “Autenticazione basata su ruoli IAM”.

Quando si creano ACL Kafka o una policy IAM con le autorizzazioni del cluster Kafka richieste, è necessario specificare 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.

Autenticazione SASL non riuscita

Per SASL/SCRAM, questo errore indica che il nome utente e la password forniti non sono validi.

Per il controllo accessi IAM, al ruolo di esecuzione manca l'autorizzazione `kafka-cluster:Connect` per il cluster MSK. 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 del servizio Amazon MSK. 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`.

Il server non è riuscito ad autenticare Lambda

Questo errore indica che i broker Amazon MSK Kafka non sono riusciti ad autenticarsi con 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 non sono configurati per l'utilizzo di mTLS.
- Un certificato client non è attendibile per i broker.

Il certificato o la chiave privata forniti non sono validi

Questo errore indica che il consumatore Amazon MSK non ha potuto utilizzare il certificato o la chiave privata forniti. Assicurati che il certificato e la chiave utilizzino il formato PEM e che la crittografia della chiave privata utilizzi un algoritmo PBES1.

Configurazione della rete

Affinché Lambda utilizzi il cluster Kafka come origine di eventi, Lambda deve accedere all'Amazon VPC in cui risiede il cluster. Ti consigliamo di implementare endpoint AWS PrivateLink VPC per Lambda per accedere al tuo VPC. Distribuisci endpoint per Lambda e (). AWS Security Token Service AWS STS Se il broker utilizza l'autenticazione, implementa anche un endpoint VPC per Secrets Manager. Se hai configurato una destinazione in caso di errore, implementa anche un endpoint VPC per il servizio di destinazione.

In alternativa, verifica che il VPC associato al cluster Kafka includa un gateway NAT per sottorete pubblica. Per ulteriori informazioni, consulta the section called “Accesso a Internet per le funzioni VPC”.

Se utilizzi endpoint VPC, devi anche configurarli in modo da abilitare i nomi DNS privati.

Quando si crea una mappatura dell'origine degli eventi per un cluster MSK, Lambda verifica se le interfacce di rete elastiche (ENI) sono già presenti per le sottoreti e i gruppi di sicurezza del VPC del cluster. Se Lambda trova ENI esistenti, tenta di riutilizzarli. Altrimenti, Lambda crea nuovi 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. Questi VPC vengono gestiti automaticamente dal servizio e non sono visibili ai clienti. Puoi anche collegare la tua funzione a un Amazon VPC. In entrambi i casi, la configurazione VPC della funzione non influisce sulla mappatura delle sorgenti degli eventi. Solo la configurazione del VPC dell'origine dell'evento determina il modo in cui Lambda si connette alla fonte dell'evento.

La configurazione di Amazon VPC è individuabile tramite l'API Amazon MSK. Non è necessario configurarlo durante l'installazione utilizzando il comando create-event-source-mapping.

Per ulteriori informazioni sulla configurazione della rete, consulta Configurazione AWS Lambda con un cluster Apache Kafka all'interno di un VPC sul blog di Compute. AWS

Regole del gruppo di sicurezza VPC

Configura i gruppi di sicurezza per l'Amazon VPC contenente il tuo cluster con le seguenti regole (come minimo):

- Regole in entrata: consenti tutto il traffico sulla porta del broker Amazon MSK (9092 per testo normale, 9094 per TLS, 9096 per SASL, 9098 per IAM) per i gruppi di sicurezza specificati per l'origine eventi.
- Regole in uscita: consenti tutto il traffico sulla porta 443 per tutte le destinazioni. Consenti tutto il traffico sulla porta del broker Amazon MSK (9092 per testo normale, 9094 per TLS, 9096 per SASL, 9098 per IAM) per i gruppi di sicurezza specificati per l'origine eventi.
- Se si utilizzano endpoint VPC anziché gateway NAT, i gruppi di sicurezza associati agli endpoint VPC devono consentire tutto il traffico in entrata sulla porta 443 dai gruppi di sicurezza dell'origine eventi.

Uso di endpoint VPC

Quando utilizzi gli endpoint VPC, le chiamate API per richiamare la tua funzione vengono instradate attraverso questi endpoint utilizzando gli ENI. Il responsabile del servizio Lambda deve chiamare `sts:AssumeRole` e `lambda:InvokeFunction` attivare tutti i ruoli e le funzioni che utilizzano tali ENI.

Per impostazione predefinita, gli endpoint VPC dispongono di policy IAM aperte. La migliore pratica consiste nel limitare queste policy per consentire solo a soggetti specifici di eseguire le azioni necessarie utilizzando quell'endpoint. Per garantire che la mappatura delle sorgenti degli eventi sia in grado di richiamare la funzione Lambda, la policy degli endpoint VPC deve consentire al principio del servizio Lambda di chiamare e. `sts:AssumeRole` `lambda:InvokeFunction` La limitazione delle policy degli endpoint VPC per consentire solo le chiamate API provenienti dall'organizzazione impedisce il corretto funzionamento della mappatura delle sorgenti degli eventi.

I seguenti esempi di policy degli endpoint VPC mostrano come concedere l'accesso richiesto al principale del servizio Lambda per gli endpoint Lambda e Lambda. AWS STS

Example Politica degli endpoint VPC - endpoint AWS STS

```
{
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Example Politica degli endpoint VPC - Endpoint Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
```

```
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Se il tuo broker Kafka utilizza l'autenticazione, puoi anche limitare la policy degli endpoint VPC per l'endpoint Secrets Manager. Per chiamare l'API Secrets Manager, Lambda utilizza il ruolo della funzione, non il responsabile del servizio Lambda. L'esempio seguente mostra una policy per gli endpoint di Secrets Manager.

Example Politica degli endpoint VPC - Endpoint Secrets Manager

```
{
    "Statement": [
        {
            "Action": "secretsmanager:GetSecretValue",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "customer_function_execution_role_arn"
                ]
            },
            "Resource": "customer_secret_arn"
        }
    ]
}
```

Se hai configurato una destinazione in caso di errore, Lambda utilizza anche il ruolo della tua funzione per chiamare una delle `s3:PutObject` due `sqs:sendMessage` o utilizzare gli `sns:Publish` ENI gestiti da Lambda.

Aggiunta di Amazon MSK come origine eventi

Per creare una mappatura dell'origine eventi, aggiungi il cluster Amazon MSK come trigger della funzione Lambda utilizzando la console Lambda, un SDK AWS o AWS Command Line Interface (AWS CLI). Tieni presente che quando aggiungi Amazon MSK come trigger, Lambda assume le impostazioni VPC del cluster Amazon MSK, non le impostazioni VPC della funzione Lambda.

Questa sezione descrive come creare una mappatura dell'origine eventi utilizzando la console Lambda e AWS CLI.

Prerequisiti

- Un cluster Amazon MSK e un argomento Kafka. Per ulteriori informazioni, consulta Nozioni di base per l'uso di Amazon MSK nella Guida per gli sviluppatori di Amazon Managed Streaming for Apache Kafka.
- Un ruolo di esecuzione con autorizzazione ad accedere alle AWS risorse utilizzate dal cluster MSK.

ID gruppo di consumer personalizzabile

Quando configuri Kafka come origine eventi, 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.

Se specifichi l'ID gruppo di consumer e sono presenti altri sondaggi attivi all'interno di quel gruppo di consumer, Kafka distribuisce i messaggi a tutti i consumer. In altre parole, Lambda non riceve tutti i messaggi relativi all'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 `StartingPosition` 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 `StartingPosition` specificata.

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.

Aggiunta di un trigger Amazon MSK (console)

Segui questi passaggi per aggiungere il cluster Amazon MSK e un argomento Kafka come trigger per la funzione Lambda.

Per aggiungere un trigger Amazon MSK alla funzione Lambda (console)

1. Aprire la pagina Functions (Funzioni) della console Lambda.

2. Scegliere il nome della funzione Lambda.
3. In Panoramica delle funzioni, scegliere Aggiungi trigger.
4. In Trigger configuration (Configurazione trigger), effettua le operazioni seguenti:

    a. Seleziona il tipo di trigger MSK.
    b. Per MSK cluster (Cluster MSK) seleziona il cluster.
    c. Per Batch Size (Dimensione batch), immettere il numero massimo di messaggi da recuperare in un singolo batch.
    d. Per Batch window (Finestra batch), immetti il tempo massimo in secondi per la raccolta dei registri da parte di Lambda prima di richiamare la funzione.
    e. Per Topic name (Nome argomento) immetti un nome per l'argomento Kafka.
    f. (Facoltativo) Per Consumer group ID (ID gruppo di consumer), inserisci l'ID di un gruppo di consumer Kafka a cui aderire.
    g. (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.
    h. (Facoltativo) Per Authentication (Autenticazione), scegli la chiave segreta per l'autenticazione con i broker nel cluster MSK.
    i. Per creare il trigger in uno stato disabilitato per il test (scelta consigliata), deselezionare Enable trigger (Abilita trigger). Oppure, per attivare immediatamente il trigger, selezionareAbilita trigger.
5. Per creare il trigger, scegli Add (Aggiungi).

Aggiunta di un trigger Amazon MSK (AWS CLI)

Usa i seguenti AWS CLI comandi di esempio per creare e visualizzare un trigger Amazon MSK per la tua funzione Lambda.

Creare un trigger utilizzando il AWS CLI

Example — Creazione di uno strumento di mappatura dell'origine degli eventi per cluster che utilizzano l'autenticazione IAM

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` La posizione iniziale dell'argomento è impostata su `LATEST`. Quando il cluster

utilizza l'autenticazione basata sui ruoli IAM, non è necessario un oggetto di configurazione. SourceAccess Esempio:

```
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
```

Example — Creazione di uno strumento di mappatura dell'origine degli eventi per cluster che utilizzano l'autenticazione SASL/SCRAM

Se il cluster utilizza l'autenticazione SASL/SCRAM, è necessario includere un oggetto di SourceAccessconfigurazione che specifichi e un ARN segreto di `SASL_SCRAM_512_AUTH` Secrets Manager.

```
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"}]'
```

Example — Creazione di uno strumento di mappatura dell'origine degli eventi per cluster che utilizzano l'autenticazione mTLS

Se il cluster utilizza l'autenticazione MTLS, è necessario includere un oggetto di SourceAccessconfigurazione che specifichi `CLIENT_CERTIFICATE_TLS_AUTH` e un ARN segreto di Secrets Manager.

```
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"}]'
```

Per ulteriori informazioni, consulta la documentazione di riferimento dell'CreateEventSourceMappingAPI.

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 6d9bce8e-836b-442c-8070-74e77903c815
```

Creazione di strumenti di mappatura dell'origine degli eventi multi-account

È possibile utilizzare la connettività privata multi-VPC per connettere una funzione Lambda a un cluster MSK assegnato in un altro Account AWS. Utilizza la connettività multi-VPC AWS PrivateLink, che mantiene tutto il traffico all'interno della AWS rete.

> **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'CreateEventSourceMappingoperazione varia anche 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 basata sui ruoli IAM, non è necessario un oggetto di configurazione. SourceAccess 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 — Creazione di uno strumento di mappatura dell'origine degli eventi multi-account per cluster che utilizzano l'autenticazione SASL/SCRAM

Se il cluster utilizza l'autenticazione SASL/SCRAM, è necessario includere un oggetto di SourceAccessconfigurazione che specifichi e un ARN segreto di `SASL_SCRAM_512_AUTH` Secrets Manager.

Esistono due modi per utilizzare i segreti per lo strumento di mappatura dell'origine degli eventi Amazon MSK multi-account con l'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 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 Autorizzazioni ai AWS Secrets Manager segreti per gli utenti di un account diverso.

```
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 oggetto di SourceAccessconfigurazione 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"}]'
```

Destinazioni in caso di errore

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 con metadati sulla chiamata non riuscita. Puoi configurare qualsiasi argomento Amazon SNS, coda Amazon SQS o bucket S3 come destinazione. Il tuo ruolo di esecuzione deve avere le autorizzazioni per la destinazione:

- Per le destinazioni SQS: sqs: SendMessage
- Per le destinazioni SNS: sns:publish
- Per le destinazioni dei bucket S3: s3: e s3: PutObject ListBuckets

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 richiesta la chiave kms:. GenerateData 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 per consentire kms: Key. GenerateData
- Se hai abilitato la crittografia con la tua chiave KMS per la destinazione SQS, sono obbligatori KMS:Decrypt e kms: Key. GenerateData 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 affidabile il ruolo di esecuzione per consentire KMS:Decrypt, kms: Key, kms: e kms:. GenerateData DescribeKey ReEncrypt
- Se hai abilitato la crittografia con la tua chiave KMS per la destinazione SNS, sono obbligatori KMS:Decrypt e kms: Key. GenerateData 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: Key, kms: e kms:. GenerateData DescribeKey ReEncrypt

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. Scegliere Save (Salva).

Puoi anche configurare una destinazione AWS CLI in caso di errore utilizzando. Ad esempio, il seguente comando create-event-source-mapping 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 seguente comando update-event-source-mapping 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": ""}}'
```

Record di invocazione di esempio SNS e SQS

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 la partizione di Kafka, separati da un trattino. Ad esempio, per la chiave `"Topic-0"`, `Topic` è l'argomento di Kafka e `0` è la partizione. 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.

Scalabilità automatica dell'origine eventi Amazon MSK

Quando si crea inizialmente una fonte evento Amazon MSK, Lambda assegna un consumatore per elaborare tutte le partizioni dell'argomento Kafka. Ogni consumatore ha più processori in esecuzione in parallelo per gestire carichi di lavoro più elevati. Inoltre, Lambda aumenta o diminuisce automaticamente il numero di consumatori in base al carico di lavoro. Per preservare l'ordinamento dei messaggi in ogni partizione, il numero massimo di consumatori è un consumatore per ogni partizione dell'argomento.

Ogni minuto, Lambda valuta il ritardo dell'offset del consumatore di tutte le partizioni dell'argomento. Se il ritardo è troppo alto, la partizione sta ricevendo messaggi più velocemente di quanto Lambda possa elaborarli. Se necessario, Lambda aggiunge o rimuove i consumer dall'argomento. Il processo di dimensionamento di aggiunta o rimozione dei consumatori avviene entro tre minuti dalla valutazione.

Se la funzione Lambda di destinazione è limitata, Lambda riduce il numero di consumer. Questa operazione riduce il carico di lavoro sulla funzione riducendo il numero di messaggi che i consumer possono recuperare e inviare alla funzione.

Per monitorare il throughput dell'argomento Kafka, visualizza il parametro del ritardo dell'offset che Lambda emette mentre la funzione elabora i registri.

Per controllare quante chiamate di funzioni si verificano in parallelo, è inoltre possibile monitorare i parametri di concorrenza per la funzione.

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`.

CloudWatch Metriche Amazon

Lambda emette il parametro `OffsetLag` mentre la funzione elabora i registri. Il valore di questo parametro è la differenza di offset tra l'ultimo registro scritto nell'argomento dell'origine eventi Kafka e l'ultimo registro elaborato da Lambda. Puoi utilizzare `OffsetLag` per stimare la latenza tra il momento in cui un registro viene aggiunto e il momento in cui il gruppo di consumer lo elabora.

Una tendenza in aumento in `OffsetLag` può indicare problemi con i sondaggi nel gruppo di consumer della funzione. Per ulteriori informazioni, consulta Utilizzo dei parametri delle funzioni Lambda.

Parametri di configurazione di Amazon MSK

Tutti i tipi di sorgenti di eventi Lambda condividono le stesse operazioni CreateEventSourceMappinge quelle dell'UpdateEventSourceMappingAPI. Tuttavia, solo alcuni dei parametri si applicano ad Amazon MSK.

Parametri dell'origine eventi applicabili ad Amazon MSK

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
Abilitato	N	Abilitato	
EventSourceArn	Y		Può essere impostato solo su Create
FunctionName	Y		
FilterCriteria	N		Filtro eventi Lambda
MaximumBatchingWindowInSecondi	N	500 ms	Comportamento di batching
SourceAccessConfigurazioni	N	Nessuna credenziale	Credenziali di autenticazione SASL/SCRAM o CLIENT_CERTIFICATE_TLS_AUTH (MutualTLS) per la tua origine eventi
StartingPosition	Y		AT_TIMESTAMP, TRIM_HORIZON o LATEST Può essere impostato solo su Create

Parametro	Obbligatorio	Predefinito	Note
StartingPositionTimestamp	N		Obbligatorio se StartingPosition è impostato su AT_TIMESTAMP
Argomenti	Y		Nome argomento Kafka Può essere impostato solo su Create

Utilizzo di Lambda con Apache Kafka gestito dal cliente

> **Note**
>
> Se desideri inviare dati a una destinazione diversa da una funzione Lambda o arricchire i dati prima di inviarli, consulta Amazon Pipes. EventBridge

Lambda supporta Apache Kafka come origine eventi. Apache Kafka è una piattaforma di flussi di eventi open source che supporta carichi di lavoro come pipeline di dati e analisi dei dati di streaming.

Puoi utilizzare il servizio Kafka AWS gestito Amazon Managed Streaming for Apache Kafka (Amazon MSK) o un cluster Kafka autogestito. Per informazioni dettagliate sull'uso di Lambda con Amazon MSK, consultare Uso di Lambda con Amazon MSK.

Questo argomento descrive come utilizzare Lambda con un cluster Kafka autogestito. In AWS terminologia, un cluster autogestito include cluster Kafka non ospitati.AWS Ad esempio, è possibile ospitare il proprio cluster Kafka con un provider cloud come Confluent Cloud.

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.)

> **Warning**
>
> Le mappature delle sorgenti degli eventi 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 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
- Autenticazione cluster Kafka
- Gestione dell'accesso e delle autorizzazioni API
- Errori di autenticazione e autorizzazione
- Configurazione della rete
- Aggiunta di un cluster Kafka come origine eventi
- Destinazioni in caso di errore
- Aggiunta di un cluster Kafka come origine eventi
- Posizioni di partenza di polling e flussi
- Scalabilità automatica dell'origine eventi Kafka
- Errori della mappatura dell'origine eventi
- CloudWatch Metriche Amazon
- Parametri di configurazione Apache Kafka gestiti dal cliente

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 della partizione 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
                  ]
               }
            ]
         }
      ]
   }
}
```

Autenticazione cluster Kafka

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.

Accesso VPC

Se accedono ai broker Kafka soltanto gli utenti Kafka all'interno del VPC, bisogna configurare la fonte evento Kafka per l'accesso ad Amazon Virtual Private Cloud (Amazon VPC).

Autenticazione SASL/SCRAM

Lambda supporta l'autenticazione Simple Authentication and Security Layer/Salted Challenge Response Authentication Mechanism (SASL/SCRAM) con crittografia Transport Layer Security (TLS) (`SASL_SSL`). Lambda invia le credenziali crittografate per l'autenticazione con il cluster. Lambda non supporta SASL/SCRAM con testo in chiaro (`SASL_PLAINTEXT`). Per ulteriori informazioni sull'autenticazione SASL/SCRAM, consultare RFC 5802.

Lambda supporta anche l'autenticazione SASL/PLAIN. 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'uso di Secrets Manager, consultare Tutorial: Creare e recuperare un segreto nella Guida per l'utente AWS Secrets Manager .

> **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 CA/autofirmato privato. Il certificato emesso da una CA pubblica deve essere firmato da un'autorità di certificazione (CA) presente nel trust store di Lambda. Per un certificato CA/autofirmato privato, 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 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, includere la password per chiave privata nel segreto.

```
{"privateKeyPassword":"testpassword",
"certificate":"-----BEGIN CERTIFICATE-----
MIIE5DCCAsygAwIBAgIRAPJdwaFaNRrytHBto0j5BA0wDQYJKoZIhvcNAQELBQAw
...
j0Lh4/+1HfgyE2KlmII36dg4IMzNjAFEBZiCRoPimO40s1cRqtFHXoal0QQbIlxk
cmUuiAii9R0=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFgjCCA2qgAwIBAgIQdjNZd6uFf9hbNC5RdfmHrzANBgkqhkiG9w0BAQsFADBb
...
rQoiowbbk5wXCheYSANQIfTZ6weQTgiCHCCbuuMKNVS95FkXm0vqVD/YpXKwA/no
c8PH3PSoAaRwMMgOSA2ALJvbRz8mpg==
-----END CERTIFICATE-----",
"privateKey":"-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFKzBVBgkqhkiG9w0BBQ0wSDAnBgkqhkiG9w0BBQwwGgQUiAFcK5hT/X7Kjmgp
...
QrSekqF+kWzmB6nAfSzgO9IaoAaytLvNgGTckWeUkWn/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. È possibile 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
EDAOBgNVBAgTB0FyaXpvbmExEzARBgNVBAcTClNjb3R0c2RhbGUxJTAjBgNVBAoT
HFN0YXJmaWVsZCBUZWNobm9sb2dpZXMsIEluYy4xOzA5BgNVBAMTMlN0YXJmaWVs
ZCBTZXJ2aWNlcyBSb290IENlcnRpZmljYXRlIEF1dG...
-----END CERTIFICATE-----"
}
```

Gestione dell'accesso e delle autorizzazioni API

Oltre ad accedere al cluster Kafka autogestito, la funzione Lambda necessita di autorizzazioni per eseguire varie operazioni 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 AWS Identity and Access Management ruolo (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:

- log: Gruppo CreateLog
- registri: Stream CreateLog
- registri: Eventi PutLog

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:

- secretsmanager: Valore GetSecret
- 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: Interfaccia CreateNetwork
- ec2: Interfacce DescribeNetwork
- ec2: DescribeVpcs
- ec2: Interfaccia DeleteNetwork
- ec2: DescribeSubnets
- ec2: Gruppi DescribeSecurity

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 e quindi collegare la policy al ruolo di esecuzione. Questo esempio mostra come creare una policy che consente a Lambda di accedere alle risorse Amazon VPC.

```
{
        "Version":"2012-10-17",
        "Statement":[
           {
              "Effect":"Allow",
              "Action":[
                 "ec2:CreateNetworkInterface",
                 "ec2:DescribeNetworkInterfaces",
```

```
                "ec2:DescribeVpcs",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource":"*"
        }
    ]
}
```

Per informazioni sulla creazione di un documento di policy JSON nella console IAM, consultare Creazione di policy nella scheda JSON nella Guida per l'utente di IAM.

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.

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 in Risultato. LastProcessing

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 SASL/SCRAM o mTLS, questo errore indica che l'utente fornito non dispone di tutte le seguenti autorizzazioni della lista di controllo accessi Kafka (ACL) richieste:

- DescribeConfigs Cluster

- Descrivi il gruppo
- Leggi il gruppo
- Descrivi l'argomento
- Leggi l'argomento

Quando si crea ACL Kafka con le autorizzazioni `kafka-cluster` richieste, è necessario specificare 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.

Autenticazione SASL non riuscita

Per SASL/SCRAM o SASL/PLAIN, questo errore indica che le credenziali di accesso fornite non sono valide.

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:

- I broker Kafka utilizzano certificati autofirmati o una CA privata, ma non hanno fornito il certificato CA root del server.
- Il certificato CA root del server non corrisponde alla CA root 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 della chiave privata utilizzi un algoritmo PBES1.

Configurazione della rete

Affinché Lambda utilizzi il cluster Kafka come origine di eventi, Lambda deve accedere all'Amazon VPC in cui risiede il cluster. Ti consigliamo di implementare gli endpoint AWS PrivateLink VPC per Lambda per accedere al tuo VPC. Distribuisci endpoint per Lambda e (). AWS Security Token Service AWS STS Se il broker utilizza l'autenticazione, implementa anche un endpoint VPC per Secrets Manager. Se hai configurato una destinazione in caso di errore, implementa anche un endpoint VPC per il servizio di destinazione.

In alternativa, verifica che il VPC associato al cluster Kafka includa un gateway NAT per sottorete pubblica. Per ulteriori informazioni, consulta the section called “Accesso a Internet per le funzioni VPC”.

Se utilizzi endpoint VPC, devi anche configurarli in modo da abilitare i nomi DNS privati.

Quando crei una mappatura dell'origine degli eventi per un cluster Apache Kafka autogestito, Lambda verifica se le interfacce di rete elastiche (ENI) sono già presenti per le sottoreti e i gruppi di sicurezza del VPC del cluster. Se Lambda trova ENI esistenti, tenta di riutilizzarli. Altrimenti, Lambda crea nuovi 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. Questi VPC vengono gestiti automaticamente dal servizio e non sono visibili ai clienti. Puoi anche connettere la tua funzione a un Amazon VPC. In entrambi i casi, la configurazione VPC della funzione non influisce sulla mappatura delle sorgenti degli eventi. Solo la configurazione del VPC dell'origine dell'evento determina il modo in cui Lambda si connette alla fonte dell'evento.

Per ulteriori informazioni sulla configurazione della rete, consulta Configurazione AWS Lambda con un cluster Apache Kafka all'interno di un VPC sul blog di Compute. AWS

Regole del gruppo di sicurezza VPC

Configura i gruppi di sicurezza per l'Amazon VPC contenente il tuo cluster con le seguenti regole (come minimo):

- Regole in entrata – Consenti tutto il traffico sulla porta del broker Kafka per i gruppi di sicurezza specificati per l'origine eventi. Kafka utilizza la porta 9092 per impostazione predefinita.
- Regole in uscita: consenti tutto il traffico sulla porta 443 per tutte le destinazioni. Consenti tutto il traffico sulla porta del broker Kafka per i gruppi di sicurezza specificati per l'origine eventi. Kafka utilizza la porta 9092 per impostazione predefinita.
- Se si utilizzano endpoint VPC anziché gateway NAT, i gruppi di sicurezza associati agli endpoint VPC devono consentire tutto il traffico in entrata sulla porta 443 dai gruppi di sicurezza dell'origine eventi.

Uso di endpoint VPC

Quando utilizzi gli endpoint VPC, le chiamate API per richiamare la tua funzione vengono instradate attraverso questi endpoint utilizzando gli ENI. Il responsabile del servizio Lambda deve chiamare `sts:AssumeRole` e `lambda:InvokeFunction` attivare tutti i ruoli e le funzioni che utilizzano tali ENI.

Per impostazione predefinita, gli endpoint VPC dispongono di policy IAM aperte. La migliore pratica consiste nel limitare queste policy per consentire solo a soggetti specifici di eseguire le azioni necessarie utilizzando quell'endpoint. Per garantire che la mappatura delle sorgenti degli eventi sia in grado di richiamare la funzione Lambda, la policy degli endpoint VPC deve consentire al principio del servizio Lambda di chiamare e. `sts:AssumeRole` `lambda:InvokeFunction` La limitazione delle policy degli endpoint VPC per consentire solo le chiamate API provenienti dall'organizzazione impedisce il corretto funzionamento della mappatura delle sorgenti degli eventi.

Il seguente esempio di policy degli endpoint VPC mostra come concedere l'accesso richiesto al principale del servizio Lambda per gli endpoint Lambda e Lambda. AWS STS

Example Politica degli endpoint VPC - endpoint AWS STS

```
{
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
```

```
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Example Politica degli endpoint VPC - Endpoint Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Se il tuo broker Kafka utilizza l'autenticazione, puoi anche limitare la policy degli endpoint VPC per l'endpoint Secrets Manager. Per chiamare l'API Secrets Manager, Lambda utilizza il ruolo della funzione, non il responsabile del servizio Lambda. L'esempio seguente mostra una policy per gli endpoint di Secrets Manager.

Example Politica degli endpoint VPC - Endpoint Secrets Manager

```
{
    "Statement": [
        {
            "Action": "secretsmanager:GetSecretValue",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "customer_function_execution_role_arn"
```

```
                ]
            },
            "Resource": "customer_secret_arn"
        }
    ]
}
```

Se hai configurato una destinazione in caso di errore, Lambda utilizza anche il ruolo della tua funzione per chiamare una delle `s3:PutObject` due `sqs:sendMessage` o utilizzare gli `sns:Publish` ENI gestiti da Lambda.

Aggiunta di un cluster Kafka come origine eventi

Per creare una mappatura dell'origine eventi, aggiungi il cluster Kafka come trigger della funzione Lambda utilizzando la console Lambda, un SDK AWS o AWS Command Line Interface (AWS CLI).

Questa sezione descrive come creare una mappatura dell'origine eventi utilizzando la console Lambda e AWS CLI.

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.

ID gruppo di consumer personalizzabile

Quando configuri Kafka come origine eventi, 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.

Se specifichi l'ID gruppo di consumer e sono presenti altri sondaggi attivi all'interno di quel gruppo di consumer, Kafka distribuisce i messaggi a tutti i consumer. In altre parole, Lambda non riceve tutti i messaggi relativi all'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 `StartingPosition` 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 `StartingPosition` specificata.

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 del gruppo di consumer specificato, non sarà più possibile aggiornare questo valore.

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. (Facoltativo) In Dimensioni batch, inserisci il numero massimo di record da ricevere in un singolo batch.
    e. Per Finestra batch, immetti il tempo massimo in secondi per la raccolta dei registri da parte di Lambda prima di richiamare la funzione.
    f. (Facoltativo) Per ID gruppo di consumer, inserisci l'ID di un gruppo di consumer Kafka a cui aderire.
    g. (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.
    h. (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.

i. (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 broker Kafka utilizza l'autenticazione SASL/PLAIN, scegli BASIC_AUTH.
- Se il broker utilizza l'autenticazione SASL/SCRAM, scegli uno dei protocolli SASL_SCRAM.
- Se stai configurando l'autenticazione mTLS, scegli il protocollo CLIENT_CERTIFICATE_TLS_AUTH.

ii. Per l'autenticazione SASL/SCRAM o mTLS, scegli la chiave segreta di Secrets Manager che contiene le credenziali per il cluster Kafka.

j. (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/SCRAM o SASL/PLAIN e all'autenticazione MTLS.

k. Per creare il trigger in uno stato disabilitato per il test (scelta consigliata), deselezionare Abilita trigger. Oppure, per attivare immediatamente il trigger, selezionareAbilita 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 broker Kafka tramite Internet, è necessario specificare il segreto di Secrets Manager 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
```

Destinazioni in caso di errore

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 con metadati sulla chiamata non riuscita. Puoi configurare qualsiasi argomento Amazon SNS, coda Amazon SQS o bucket S3 come destinazione. Il tuo ruolo di esecuzione deve avere le autorizzazioni per la destinazione:

- Per le destinazioni SQS: sqs: SendMessage
- Per le destinazioni SNS: sns:Publish

- Per le destinazioni dei bucket S3: s3: e s3: PutObject ListBuckets

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 richiesta la chiave kms:. GenerateData 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 per consentire kms: Key. GenerateData
- Se hai abilitato la crittografia con la tua chiave KMS per la destinazione SQS, sono obbligatori KMS:Decrypt e kms: Key. GenerateData 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 affidabile il ruolo di esecuzione per consentire KMS:Decrypt, kms: Key, kms: e kms:. GenerateData DescribeKey ReEncrypt
- Se hai abilitato la crittografia con la tua chiave KMS per la destinazione SNS, sono obbligatori KMS:Decrypt e kms: Key. GenerateData 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: Key, kms: e kms:. GenerateData DescribeKey ReEncrypt

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. Scegliere Save (Salva).

Puoi anche configurare una destinazione AWS CLI in caso di errore utilizzando. Ad esempio, il seguente comando create-event-source-mapping 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 seguente comando update-event-source-mapping 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": ""}}'
```

Record di invocazione di esempio SNS e SQS

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 la partizione di Kafka, separati da un trattino. Ad esempio, per la chiave `"Topic-0"`, `Topic` è l'argomento di Kafka e `0` è la partizione. 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.

Aggiunta di un cluster Kafka come origine eventi

Quando aggiungi il cluster Apache Kafka 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 CreateEventSourceMappingrichiesta, in base a ciò che specifichi. `StartingPosition` Dopo che l'elaborazione è avvenuta con successo, l'argomento Kafka viene salvato nel cluster Kafka.

Se specifichi `StartingPosition` come `LATEST`, Lambda inizia a leggere a partire dall'ultimo messaggio in ogni partizione appartenente all'argomento. Poiché ci può essere un certo ritardo dopo la configurazione del trigger prima che Lambda inizi a leggere i messaggi, Lambda non legge alcun messaggio prodotto durante questo periodo.

Lambda elabora i registri da una o più partizioni dell'argomento Kafka specificate e invia un payload JSON alla funzione. Quando sono disponibili più record, Lambda continua a elaborare i record in batch, in base al `BatchSize` valore specificato in una CreateEventSourceMappingrichiesta, finché la funzione non raggiunge l'argomento.

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. È possibile inviare i record che non superano tutti i tentativi di riprova a una destinazione in caso di 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. Questa limitazione garantisce che lo strumento di mappatura dell'origine degli eventi possa gestire correttamente errori di funzioni e nuovi tentativi.

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`.

Scalabilità automatica dell'origine eventi Kafka

Quando si crea inizialmente un'origine eventi Apache Kafka, Lambda assegna un consumatore per elaborare tutte le partizioni dell'argomento Kafka. Ogni consumatore ha più processori in esecuzione in parallelo per gestire carichi di lavoro più elevati. Inoltre, Lambda aumenta o diminuisce automaticamente il numero di consumatori in base al carico di lavoro. Per preservare l'ordinamento dei messaggi in ogni partizione, il numero massimo di consumatori è un consumatore per ogni partizione dell'argomento.

Ogni minuto, Lambda valuta il ritardo dell'offset del consumatore di tutte le partizioni dell'argomento. Se il ritardo è troppo alto, la partizione sta ricevendo messaggi più velocemente di quanto Lambda possa elaborarli. Se necessario, Lambda aggiunge o rimuove i consumer dall'argomento. Il processo di dimensionamento di aggiunta o rimozione dei consumatori avviene entro tre minuti dalla valutazione.

Se la funzione Lambda di destinazione è sovraccarica, Lambda riduce il numero di consumer. Questa operazione riduce il carico di lavoro sulla funzione riducendo il numero di messaggi che i consumer possono recuperare e inviare alla funzione.

Per monitorare il throughput del proprio argomento Kafka, è possibile visualizzare i parametri dei consumer Apache Kafka, come `consumer_lag` e `consumer_offset`. Per controllare quante invocazioni di funzioni si verificano in parallelo, è inoltre possibile monitorare i parametri di concorrenza per la funzione.

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 una partizione 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.

CloudWatch Metriche Amazon

Lambda emette il parametro `OffsetLag` mentre la funzione elabora i registri. Il valore di questo parametro è la differenza di offset tra l'ultimo registro scritto nell'argomento dell'origine eventi Kafka e l'ultimo registro elaborato da Lambda. Puoi utilizzare `OffsetLag` per stimare la latenza tra il momento in cui un registro viene aggiunto e il momento in cui il gruppo di consumer lo elabora.

Una tendenza in aumento in `OffsetLag` può indicare problemi con i sondaggi nel gruppo di consumer della funzione. Per ulteriori informazioni, consulta Utilizzo dei parametri delle funzioni Lambda.

Parametri di configurazione Apache Kafka gestiti dal cliente

Tutti i tipi di sorgenti di eventi Lambda condividono le stesse operazioni CreateEventSourceMappinge quelle dell'UpdateEventSourceMappingAPI. Tuttavia, solo alcuni dei parametri si applicano ad Apache Kafka.

Parametri di origine eventi che si applicano ad Apache Kafka autogestito

Parametro	Obbligatorio	Predefinito	Note
BatchSize	N	100	Massimo: 10.000
Abilitato	N	Abilitato	
FunctionName	Y		
FilterCriteria	N		Filtro eventi Lambda
MaximumBatchingWindowInSecondi	N	500 ms	Comportamento di batching
SelfManagedEventSource	Y		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
SourceAccessConfigurazioni	N	Nessuna credenziale	Informazioni sul VPC o credenziali di autenticazione per il cluster

Parametro	Obbligatorio	Predefinito	Note
			Per SASL_PLAIN, impostare su BASIC_AUTH
StartingPosition	Y		AT_TIMESTAMP, TRIM_HORIZON o LATEST Può essere impostato solo su Create
StartingPositionTimestamp	N		Obbligatorio se StartingPosition è impostato su AT_TIMESTAMP
Argomenti	Y		Nome argomento Può essere impostato solo su Create

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 Pipes. EventBridge

È possibile utilizzare una funzione Lambda per elaborare i messaggi in una coda Amazon Simple Queue Service (Amazon SQS). Lambda supporta sia le code standard che le codeFIFO (First-in, First-Out) per le mappature delle sorgenti degli eventi.

Comprensione del comportamento di polling e batch per le mappature delle sorgenti degli eventi Amazon SQS

Con le mappature delle sorgenti 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 riceve un batch, i messaggi rimangono in coda ma sono nascosti per tutta la durata del timeout di visibilità della coda. Se la funzione elabora correttamente tutti i messaggi del batch, Lambda elimina i messaggi dalla coda. 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 dopo la scadenza del timeout di visibilità. Per questo motivo, il codice della funzione deve riuscire a elaborare lo stesso messaggio più volte, senza intoppi indesiderati.

> **Warning**
>
> Le mappature delle sorgenti degli eventi 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 ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

Per impedire a Lambda di elaborare un messaggio più volte, puoi configurare la mappatura dell'origine degli eventi per includere gli errori degli elementi batch nella risposta della funzione oppure puoi utilizzare l'DeleteMessageAPI 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 le mappature delle sorgenti degli eventi SQS, vedere. the section called “Creazione di una mappatura delle sorgenti degli eventi SQS”

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": "AIDAIENQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082649185"
            },
            "messageAttributes": {},
            "md5OfBody": "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": "AIDAIENQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082650649"
            },
            "messageAttributes": {},
            "md5OfBody": "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 numero limitato di record, puoi configurare l'origine dell'evento in modo che memorizzi i record nel buffer per un massimo di 5 minuti configurando una finestra batch. Prima di richiamare la funzione, Lambda continua a eseguire il polling dei messaggi dalla coda standard fino alla scadenza

della finestra del batch, al raggiungimento della quota di dimensione del payload di chiamata o al raggiungimento della dimensione massima del batch configurata.

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": "AIDAIO23YVJENQZJOL4VO",
                "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 una mappatura delle sorgenti di eventi Amazon SQS

Per elaborare i messaggi Amazon SQS con Lambda, configura la coda con le impostazioni appropriate, quindi crea una mappatura dell'origine degli eventi Lambda.

Configurazione di una coda da utilizzare con Lambda

Se non disponi già di una coda Amazon SQS esistente, creane una che funga da origine di eventi per la tua funzione Lambda. Quindi configura la coda in modo che la funzione Lambda abbia tempo sufficiente per elaborare ogni batch di eventi.

Per consentire alla funzione di elaborare ogni batch di record, imposta il timeout di visibilità della coda di origine su almeno sei volte il timeout di configurazione della tua funzione. Il tempo supplementare consente a Lambda di riprovare se la funzione viene limitata durante l'elaborazione di un batch precedente.

Per impostazione predefinita, se Lambda rileva un errore in qualsiasi momento durante l'elaborazione di un batch, tutti i messaggi in quel batch tornano in coda. Dopo il timeout di visibilità, i messaggi diventano nuovamente visibili a Lambda. È possibile configurare la mappatura delle sorgenti degli eventi in modo da utilizzare risposte batch parziali per riportare in coda solo i messaggi non riusciti. Inoltre, se la tua funzione non riesce a elaborare un messaggio più volte, Amazon SQS può inviarlo a una coda di lettere non scritte. Ti consigliamo di impostare la politica di redrive della coda di origine `maxReceiveCount` su almeno 5. Ciò offre a Lambda alcune possibilità di riprovare prima di inviare i messaggi non riusciti direttamente alla coda delle lettere morte.

Configurazione delle autorizzazioni per il ruolo di esecuzione Lambda

La policy AWSLambdaSQSQueueExecutionRole AWS gestita include le autorizzazioni di cui Lambda ha bisogno 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

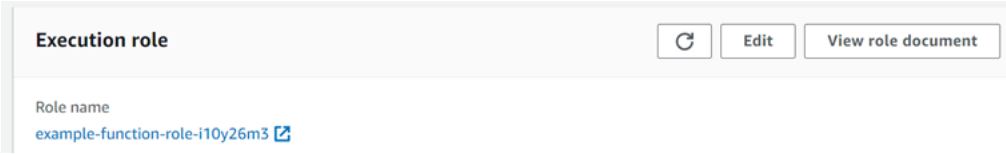
Creazione di una mappatura delle sorgenti degli eventi SQS

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.

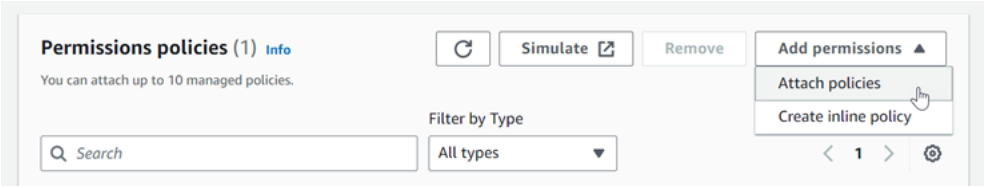
Per configurare la tua funzione per la lettura da Amazon SQS, collega la policy AWSLambdaSQSQueueExecutionRole AWS gestita al tuo ruolo di esecuzione. Quindi, crea una mappatura della sorgente degli eventi SQS dalla console utilizzando i seguenti passaggi.

Per aggiungere autorizzazioni e creare un trigger

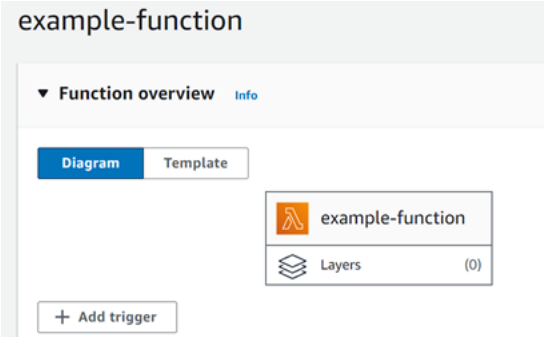
1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Quindi, seleziona la scheda Configuration (Configurazione) e poi Permissions (Autorizzazioni).
4. In Nome del ruolo, scegli il link al tuo ruolo di esecuzione. Questo link apre il ruolo nella console 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 di cui la funzione ha bisogno per leggere da una coda Amazon SQS. Per ulteriori informazioni su questa politica, consulta il AWS Managed Policy AWSLambdaSQSQueueExecutionRoleReference.
7. Torna alla tua 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 sorgenti di eventi Amazon SQS:

Coda SQS

La coda Amazon SQS da cui leggere i record.

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 concedere tempo sufficiente per elaborare un intero batch di articoli. 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 record del batch alla funzione in un'unica chiamata, purché la dimensione totale degli eventi non superi la quota di dimensione del payload di invocazione per la chiamata 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 delle ReceiveMessageAPI nell'Amazon Simple 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 utilizzi una finestra batch superiore a 0 secondi, devi 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 sono ancora disponibili dei messaggi, Lambda aggiunge un massimo di altre 300 istanze della funzione al minuto, fino a un massimo di 1.000 istanze di funzione. Per informazioni su simultaneità e dimensionamento delle funzioni, consulta Dimensionamento della funzione Lambda.

Per elaborare più messaggi, puoi ottimizzare la funzione Lambda per una maggiore velocità di trasmissione effettiva. Per ulteriori informazioni, consulta Comprendere la AWS Lambda scalabilità con le code standard di Amazon SQS.

Simultaneità massima

Il numero massimo di funzioni simultanee che l'origine eventi può richiamare. Per ulteriori informazioni, consulta Configurazione della simultaneità massima per le origini eventi di Amazon SQS.

Criteri di filtro

Aggiungi i criteri di filtro per controllare gli eventi che Lambda invia alla funzione per l'elaborazione. Per ulteriori informazioni, consulta Filtro eventi Lambda.

Configurazione del comportamento di ridimensionamento per le mappature delle sorgenti degli eventi SQS

Per le code standard, Lambda utilizza il polling lungo per eseguire il polling di una coda finché non diventa attiva. Quando sono disponibili dei messaggi, Lambda inizia a elaborare cinque batch alla volta con cinque chiamate simultanee della funzione. Se sono ancora disponibili dei messaggi, Lambda aumenta il numero di processi che stanno leggendo i batch fino a 300 istanze aggiuntive al minuto. Il numero massimo di batch che è possibile elaborare contemporaneamente con una mappatura fonte evento è 1.000.

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.

Configurazione della simultaneità massima per le origini eventi di Amazon SQS

È possibile utilizzare l'impostazione di massima concorrenza per controllare il comportamento di ridimensionamento delle sorgenti di eventi SQS. 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.

Se la simultaneità massima non è impostata, Lambda può dimensionare l'origine di eventi Amazon SQS fino alla quota di simultaneità totale del tuo account che, per impostazione predefinita, è 1.000.

> **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. Selezionare Salva.

Configura la concorrenza massima utilizzando () AWS Command Line InterfaceAWS 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'UpdateEventSourceMappingazione CreateEventSourceMappingo con un ScalingConfigoggetto.

Gestione degli errori per un'origine di eventi SQS in Lambda

Per gestire gli errori relativi a un'origine di eventi SQS, Lambda utilizza automaticamente una strategia di riprova con una strategia di backoff. Puoi anche personalizzare il comportamento di gestione degli errori configurando la mappatura delle sorgenti 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 interromperà l'elaborazione e riproverà la chiamata. Nel frattempo, Lambda si ritira gradualmente, riducendo la quantità di concorrenza allocata alla mappatura delle sorgenti degli eventi Amazon SQS. Una volta scaduto il timeout di visibilità della coda, il messaggio riapparirà nuovamente 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 parziali in batch, specifica l'azione FunctionResponseTipi durante `ReportBatchItemFailures` la configurazione della mappatura delle sorgenti degli eventi. Ciò consente alla funzione di restituire un completamento parziale, riducendo così il numero di tentativi non necessari sui registri.

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 della sorgente dell'evento, esegui il comando list-event-source-mappings. AWS CLI

   ```
   aws lambda update-event-source-mapping \
   --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
   --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

AWS SDK for .NET

> **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.

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 degli errori degli elementi batch SQS con Lambda utilizzando Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "encoding/json"
 "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 /* Your message processing condition here */ {
   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 altro da sapere. 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>();
        String messageId = "";
        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
            try {
                //process your message
                messageId = message.getMessageId();
            } catch (Exception e) {
                //Add failed message identifier to the batchItemFailures
 list
                batchItemFailures.add(new
 SQSBatchResponse.BatchItemFailure(messageId));
            }
```

```
        }
        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 utilizzando Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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 utilizzando Lambda. 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 degli errori degli elementi batch 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 altro da sapere. 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:
                # process message
            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
```

Rust

SDK per Rust

> **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 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 tornano in coda, vedi Come posso risolvere i problemi della funzione Lambda SQS? ReportBatchItemFailures nel Knowledge Center. AWS

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 riporta correttamente gli errori degli articoli in batch, puoi monitorare i parametri `NumberOfMessagesDeleted` e `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.

Parametri Lambda per le mappature delle sorgenti degli eventi Amazon SQS

Tutti i tipi di sorgenti di eventi Lambda condividono le stesse operazioni CreateEventSourceMappinge quelle dell'UpdateEventSourceMappingAPI. Tuttavia, solo alcuni dei parametri si applicano ad Amazon SQS.

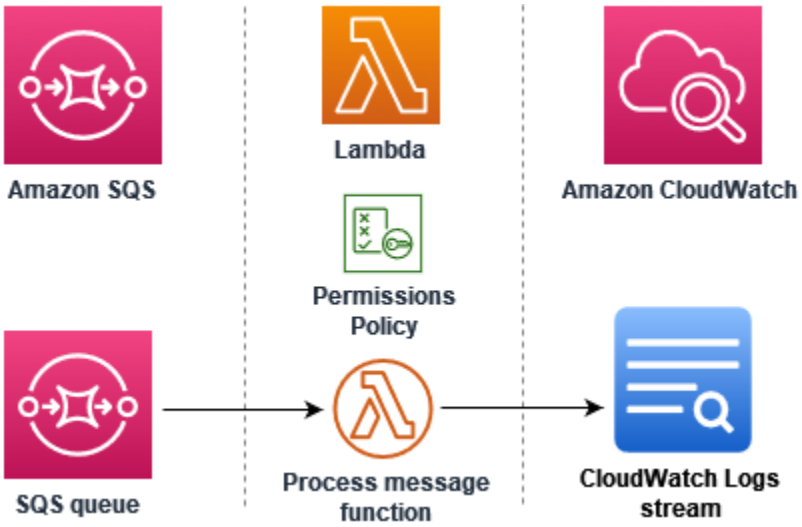
Parametri dell'origine evento applicabili ad Amazon SQS

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	
EventSourceArn	Y		L'ARN del flusso dei dati o di un consumatore di flusso
FunctionName	Y		
FilterCriteria	N		Filtro eventi Lambda
FunctionResponseTipologie	N		Per consentire alla funzione di segnalare errori specifici in un batch, includi il valore `ReportBatchItemFailures` in `FunctionResponseTypes` . Per ulteriori informazioni, consulta

Parametro	Obbligatorio	Predefinito	Note
			Implementazione di risposte batch parziali.
MaximumBatchingWindowInSecondi	N	0	
ScalingConfig	N		Configurazione della simultaneità massima per le origini eventi di Amazon SQS

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 registri. 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 in Logs. CloudWatch

Prerequisiti

Registrati per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come procedura consigliata in materia di sicurezza, assegnate l'accesso amministrativo a un utente e utilizzate solo l'utente root per eseguire attività che richiedono l'accesso da parte dell'utente root.

AWS ti invia un'e-mail di conferma dopo il completamento della procedura di registrazione. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .

2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .

2. In IAM Identity Center, concedi l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con le impostazioni predefinite IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accedi come utente con accesso amministrativo

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso ad altri utenti

1. In IAM Identity Center, crea un set di autorizzazioni che segua la migliore pratica di applicazione delle autorizzazioni con privilegi minimi.

   Per istruzioni, consulta Creare un set di autorizzazioni nella Guida per l'utente.AWS IAM Identity Center

2. Assegna gli utenti a un gruppo, quindi assegna l'accesso Single Sign-On al gruppo.

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente.AWS IAM Identity Center

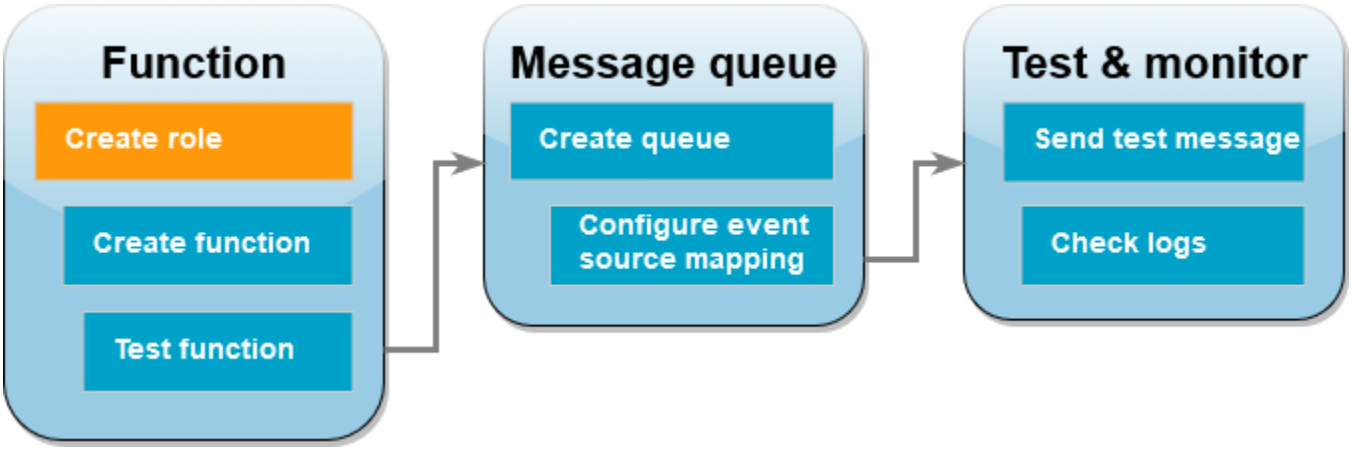
Installa il AWS Command Line Interface

Se non l'hai ancora installato AWS Command Line Interface, segui i passaggi indicati in Installazione o aggiornamento della versione più recente di 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 versione integrata su Windows 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 ad AWS accedere a servizi e risorse. Per consentire alla tua funzione di leggere articoli da Amazon SQS, allega la politica di AWSLambdaSQSQueueExecutionRoleautorizzazione.

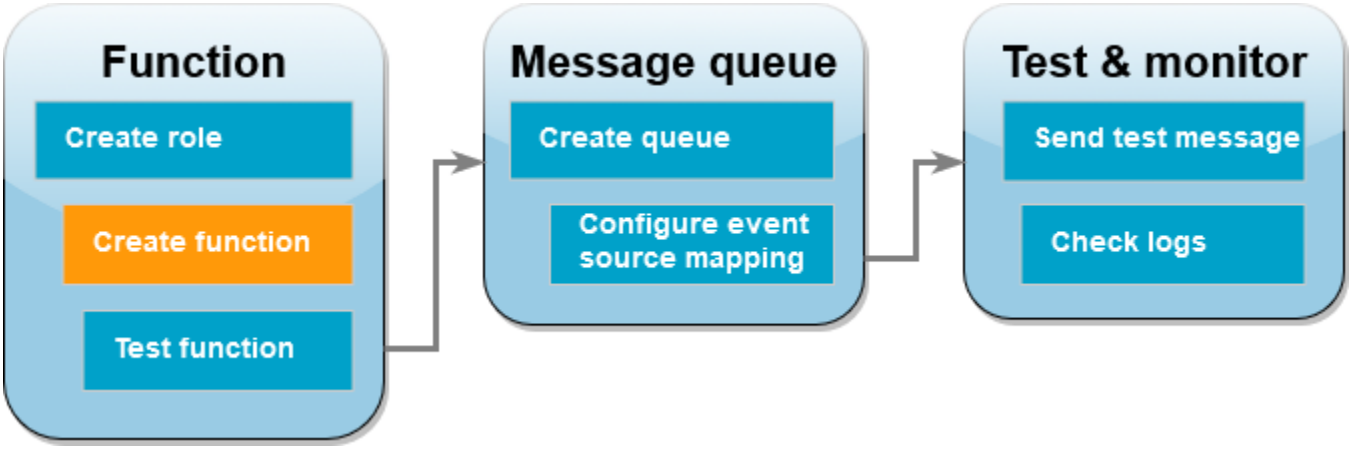
Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione Amazon SQS personalizzata

1. Aprire la pagina Roles (Ruoli) della console IAM.
2. Scegliere Create role (Crea ruolo).

3. Per Tipo di entità attendibile, scegli Servizio AWS .
4. In Caso d'uso, scegli Lambda.
5. Seleziona Successivo.
6. Nella casella di ricerca Policy di autorizzazione, inserisci **`AWSLambdaSQSQueueExecutionRole`**.
7. Seleziona la AWSLambdaSQSQueueExecutionRolepolitica, 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 CloudWatch Amazon SQS in Logs.

Questo tutorial utilizza il runtime Node.js 18.x, ma è fornito anche un 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 scheda. JavaScript

.NET

AWS SDK for .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.DefaultLambdaJsonSeri

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.

Consumo di un evento SQS con Lambda utilizzando 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 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 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
  rescue StandardError => err
    puts "An error occurred"
    raise err
  end
end
```

Rust

SDK per Rust

> **Note**
>
> C'è dell'altro 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 Lambda utilizzando 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
}
```

Creazione di una funzione Lambda in Node.js

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir sqs-tutorial
cd sqs-tutorial
```

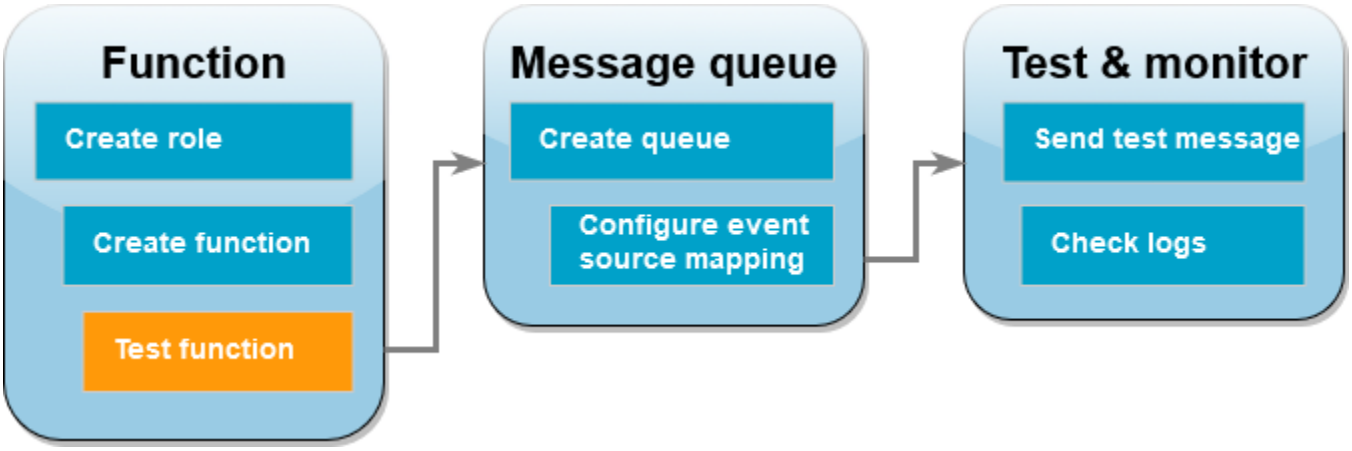
2. Copia il JavaScript codice di esempio in un nuovo file denominato. `index.js`
3. Crea un pacchetto di implementazione utilizzando il seguente comando `zip`.

```
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.

```
aws lambda create-function --function-name ProcessSQSRecord \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs18.x \
--role arn:aws:iam::111122223333:role/lambda-sqs-role
```

Test della funzione



Richiama la funzione Lambda manualmente utilizzando `invoke` AWS CLI il 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": "AIDAIENQZJOLO23YVJ4VO",
                "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 nella risposta. Per ulteriori informazioni sul recupero di oggetti, consulta Accesso ai log con AWS CLI.

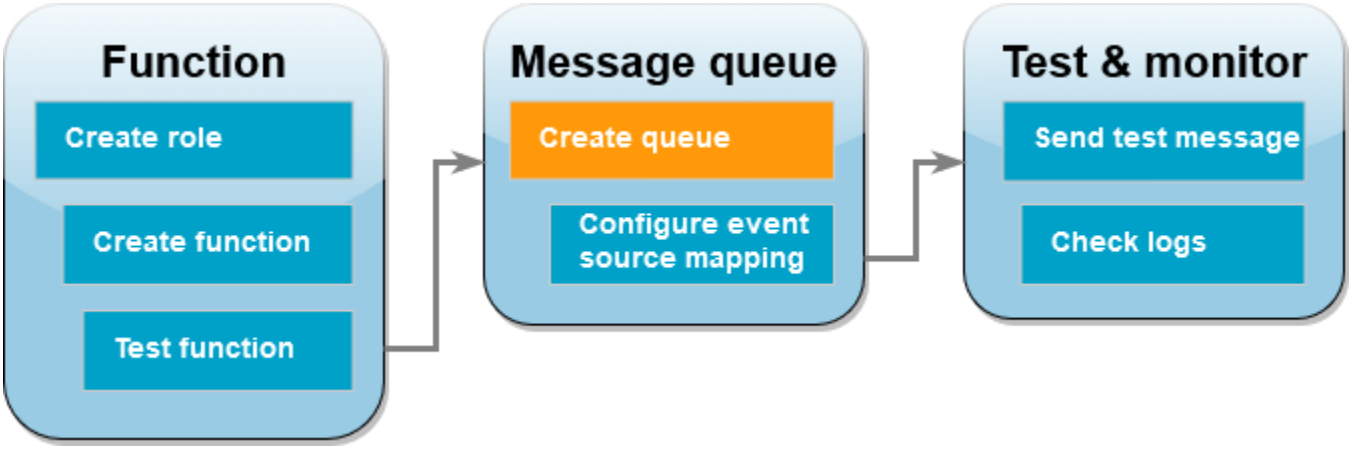
```
aws lambda invoke --function-name ProcessSQSRecord --payload file://input.json out
 --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

   L'cli-binary-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.

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



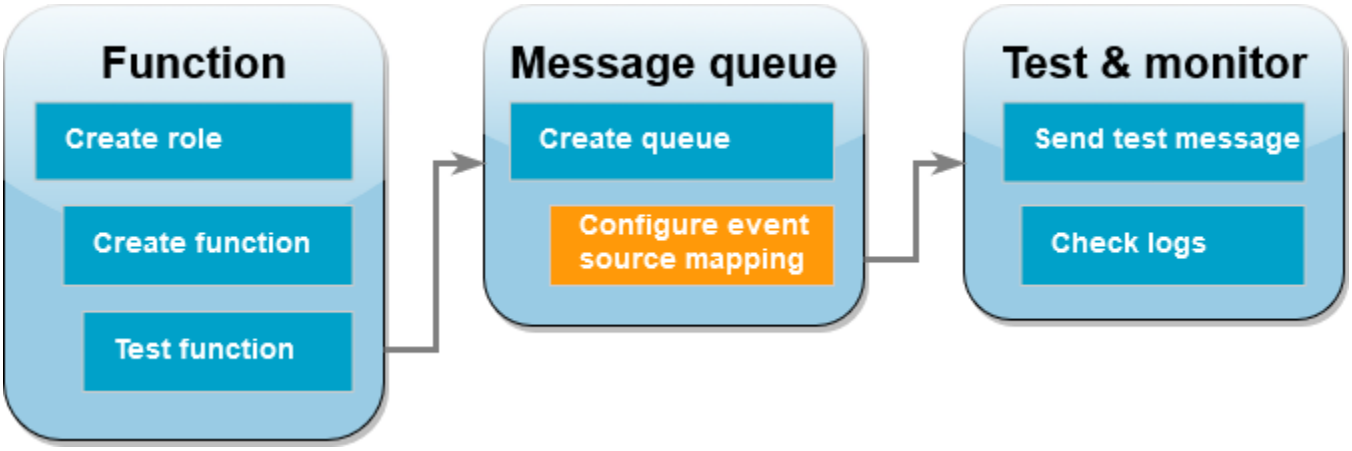
Creare una coda Amazon SQS che la funzione Lambda può utilizzare come origine eventi.

Per 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.

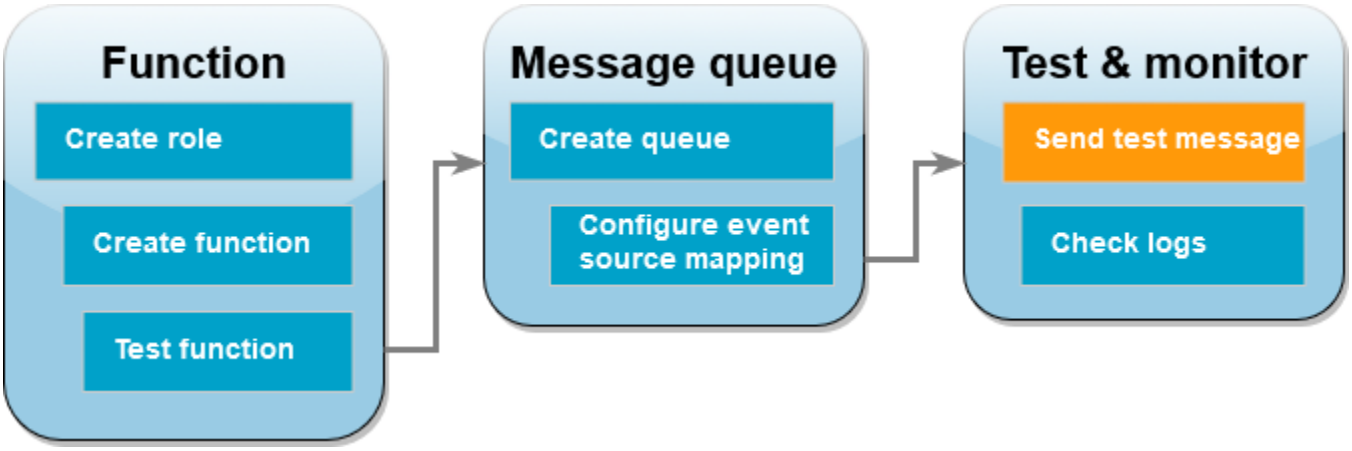
Per creare una mappatura tra la coda Amazon SQS e la funzione Lambda, usa il comando. create-event-source-mapping AWS CLI Esempio:

```
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 sorgenti 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

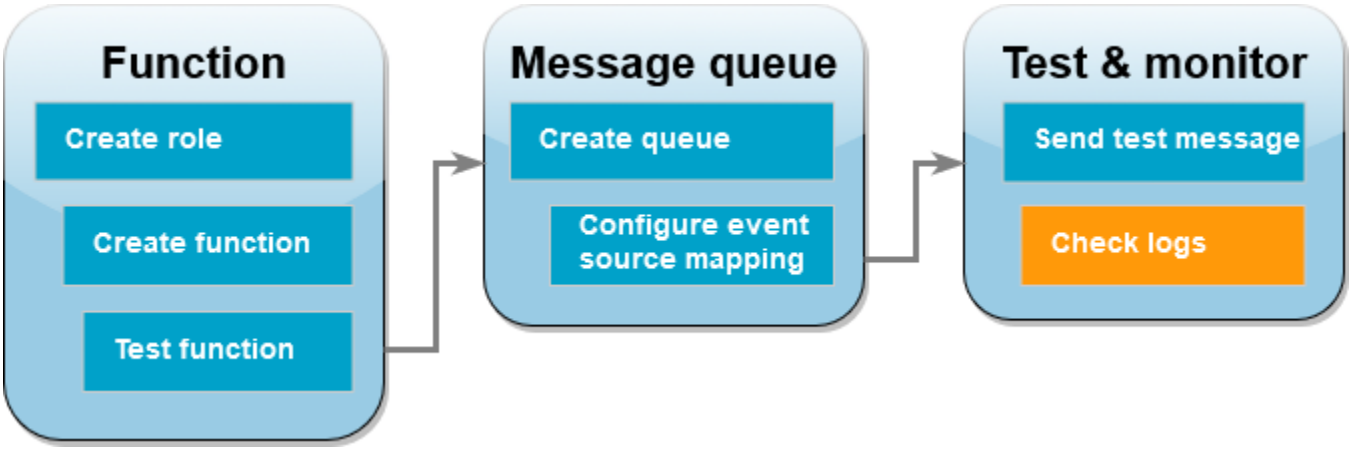


Invio di un messaggio Amazon SQS alla funzione Lambda

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 CloudWatch registri.
5. Nella CloudWatch console, scegli il flusso di log 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 addebiti inutili a tuo Account AWS carico.

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 **`delete`** nel campo di immissione testo e scegli Delete (Elimina).

Per eliminare la coda Amazon SQS

1. Accedi AWS Management Console e apri la console Amazon SQS all'indirizzo https://console.aws.amazon.com/sqs/.
2. Selezionare la coda creata.
3. Scegliere Delete (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 account diverso. AWS Questo tutorial include 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

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 l'AWS Command Line Interface (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 del ruolo di esecuzione (account A)

Nell'Account A, crea un ruolo di esecuzione che autorizzi la 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 Crea ruolo.
3. Creare un ruolo con le seguenti proprietà.
   - Entità attendibile – AWS Lambda
   - Autorizzazioni — AWSLambdaSQSQueueExecutionRole
   - Nome ruolo – **`cross-account-lambda-sqs-role`**

La AWSLambdaSQSQueueExecutionRolepolicy dispone delle autorizzazioni necessarie alla funzione per leggere elementi da Amazon SQS e scrivere log su Amazon Logs. CloudWatch

Creazione della funzione (account A)

In Account A, crea una funzione Lambda che elabora i messaggi Amazon SQS. Il seguente esempio di codice Node.js 18 scrive ogni messaggio in un log in Logs. CloudWatch

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 funzione in Node.js 18. 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. Create la funzione utilizzando il comando `create-function` AWS Command Line Interface (AWS CLI).

   ```
   aws lambda create-function --function-name CrossAccountSQSExample \
   --zip-file fileb://function.zip --handler index.handler --runtime nodejs18.x \
   --role arn:aws:iam::<AccountA_ID>:role/cross-account-lambda-sqs-role
   ```

Testa la funzione (Account A)

Nell'Account A, verifica 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": "AIDAIENQZJOLO23YVJ4VO",
                "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-formatopzione 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.

Per 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 policy JSON:

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AllActions",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "arn:aws:iam::<AccountA_ID>:role/cross-account-lambda-sqs-role"
         ]
      },
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:us-east-1:<AccountB_ID>: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 comando seguente. `create-event-source-mapping` AWS CLI

```
aws lambda create-event-source-mapping --function-name CrossAccountSQSExample --batch-
size 10 \
--event-source-arn arn:aws:sqs:us-east-1:<AccountB_ID>: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:<AccountB_ID>: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 LambdaCrossAccountQueue, che 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 continuerà a eseguire 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. La tua funzione viene eseguita e crea registri 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 ai tuoi. 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 **`delete`** 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 AWS Management Console e apri la console Amazon SQS all'indirizzo https://console.aws.amazon.com/sqs/.
2. Selezionare la coda creata.
3. Scegliere Delete (Elimina).
4. Inserisci **`confirm`** nel campo di immissione del testo.
5. Scegli Delete (Elimina).

Elaborazione di 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 chiamata simultanea da DocumentDB alla volta. Per monitorare la tua funzione, puoi tenere traccia dei relativi parametri di simultaneità.

> **Warning**
>
> Le mappature delle sorgenti degli eventi 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 ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

Argomenti

- Tutorial: Utilizzo AWS Lambda con Amazon DocumentDB Streams

Esempio di evento Amazon DocumentDB


```
{
    "eventSourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:canaryclusterb2a659a2-qo5tcmqkcl03",
    "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 Regione AWS funzione. 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 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 Configurazione della rete.
- Abilitare TLS sul cluster Amazon DocumentDB. Si tratta dell'impostazione di default. 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 utilizzi AWS Command Line Interface (AWS CLI) per la configurazione, fornisci questa chiave nell'`source-access-configuration`opzione. È possibile includere questa opzione sia con il comando `create-event-source-mapping` sia con il comando `update-event-source-mapping`. Per 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
  - RDS: descritto B ClusterParameters
  - RDS: descritto B SubnetGroups
  - ec2: Interfaccia CreateNetwork
  - ec2: Interfacce DescribeNetwork
  - ec2: DescribeVpcs
  - ec2: Interfaccia DeleteNetwork
  - ec2: DescribeSubnets
  - ec2: Gruppi DescribeSecurity
  - kms:Decrypt
  - secretsmanager: Valore GetSecret
- 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.

Configurazione della rete

Affinché Lambda possa utilizzare il cluster Amazon DocumentDB come origine di eventi, Lambda deve accedere all'Amazon VPC in cui risiede il cluster. Ti consigliamo di implementare gli endpoint AWS PrivateLink VPC per Lambda per accedere al tuo VPC. Implementa un endpoint VPC per Lambda e, se il cluster utilizza l'autenticazione, implementa anche un endpoint VPC per Secrets Manager.

In alternativa, verifica che il VPC associato al cluster Amazon DocumentDB includa un gateway NAT per sottorete pubblica. Per ulteriori informazioni, consulta the section called “Accesso a Internet per le funzioni VPC”.

Se utilizzi endpoint VPC, devi anche configurarli in modo da abilitare i nomi DNS privati.

Quando crei una mappatura dell'origine degli eventi per un cluster Amazon DocumentDB, Lambda verifica se le Interfacce di rete elastiche (ENI) sono già presenti per le sottoreti e i gruppi di sicurezza del VPC del cluster. Se Lambda trova ENI esistenti, tenta di riutilizzarli. Altrimenti, Lambda crea nuovi 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. Questi VPC vengono gestiti automaticamente dal servizio e non sono visibili ai clienti. Puoi anche connettere la tua funzione a un Amazon VPC. In entrambi i casi, la configurazione VPC della funzione non influisce sulla mappatura delle sorgenti degli eventi. Solo la configurazione del VPC dell'origine dell'evento determina il modo in cui Lambda si connette alla fonte dell'evento.

Regole del gruppo di sicurezza VPC

Configura i gruppi di sicurezza per l'Amazon VPC contenente il tuo cluster con le seguenti regole (come minimo):

- Regole in entrata: consenti tutto il traffico sulla porta del cluster Amazon DocumentDB per i gruppi di sicurezza specificati per l'origine dell'evento. Amazon DocumentDB utilizza la porta 27017 per impostazione predefinita.

- Regole in uscita: consenti tutto il traffico sulla porta 443 per tutte le destinazioni. Consenti tutto il traffico sulla porta del cluster Amazon DocumentDB. Amazon DocumentDB utilizza la porta 27017 per impostazione predefinita.
- Se si utilizzano endpoint VPC anziché gateway NAT, i gruppi di sicurezza associati agli endpoint VPC devono consentire tutto il traffico in entrata sulla porta 443 dai gruppi di sicurezza dell'origine eventi.

Uso di endpoint VPC

Quando utilizzi gli endpoint VPC, le chiamate API per richiamare la tua funzione vengono instradate attraverso questi endpoint utilizzando gli ENI. Il principale del servizio Lambda deve richiamare tutte `lambda:InvokeFunction` le funzioni che utilizzano tali ENI.

Per impostazione predefinita, gli endpoint VPC dispongono di policy IAM aperte. La migliore pratica consiste nel limitare queste policy per consentire solo a soggetti specifici di eseguire le azioni necessarie utilizzando quell'endpoint. Per garantire che la mappatura delle sorgenti degli eventi sia in grado di richiamare la funzione Lambda, la policy degli endpoint VPC deve consentire la chiamata al principio del servizio Lambda. `lambda:InvokeFunction` Limitare le policy degli endpoint VPC per consentire solo le chiamate API provenienti dall'organizzazione impedisce il corretto funzionamento della mappatura delle sorgenti degli eventi.

Il seguente esempio di policy degli endpoint VPC mostra come concedere l'accesso richiesto per gli endpoint Lambda.

Example Politica degli endpoint VPC - Endpoint Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Se il tuo cluster Amazon DocumentDB utilizza l'autenticazione, puoi anche limitare la policy degli endpoint VPC per l'endpoint Secrets Manager. Per chiamare l'API Secrets Manager, Lambda utilizza il ruolo della funzione, non il responsabile del servizio Lambda. L'esempio seguente mostra una policy per gli endpoint di Secrets Manager.

Example Politica degli endpoint VPC - Endpoint Secrets Manager

```
{
    "Statement": [
        {
            "Action": "secretsmanager:GetSecretValue",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "customer_function_execution_role_arn"
                ]
            },
            "Resource": "customer_secret_arn"
        }
    ]
}
```

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 tralasciati fino a quando l'origine degli eventi 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'FullDocumentAPI 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 degli eventi 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-
epzcyvu4pjoy",
    "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-
epzcyvu4pjoy",
    "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, pertanto potresti non essere in grado di visualizzare 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 dei parametri delle funzioni Lambda.

I flussi di modifiche di Amazon DocumentDB non sono ottimizzati per gestire ampi intervalli di tempo tra gli eventi. Se la fonte di eventi Amazon DocumentDB non riceve alcun evento per un periodo di tempo prolungato, Lambda può disabilitare la mappatura delle sorgenti 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 tralascia 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, saranno completate 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 end-to-end configurazione inserendo elementi nel tuo database Amazon DocumentDB.

Argomenti

Prerequisiti

Registrati per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come procedura consigliata in materia di sicurezza, assegnate l'accesso amministrativo a un utente e utilizzate solo l'utente root per eseguire attività che richiedono l'accesso da parte dell'utente root.

AWS ti invia un'e-mail di conferma dopo il completamento della procedura di registrazione. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .
2. In IAM Identity Center, concedi l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con le impostazioni predefinite IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accedi come utente con accesso amministrativo

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso ad altri utenti

1. In IAM Identity Center, crea un set di autorizzazioni che segua la migliore pratica di applicazione delle autorizzazioni con privilegi minimi.

   Per istruzioni, consulta Creare un set di autorizzazioni nella Guida per l'utente.AWS IAM Identity Center
2. Assegna gli utenti a un gruppo, quindi assegna l'accesso Single Sign-On al gruppo.

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente.AWS IAM Identity Center

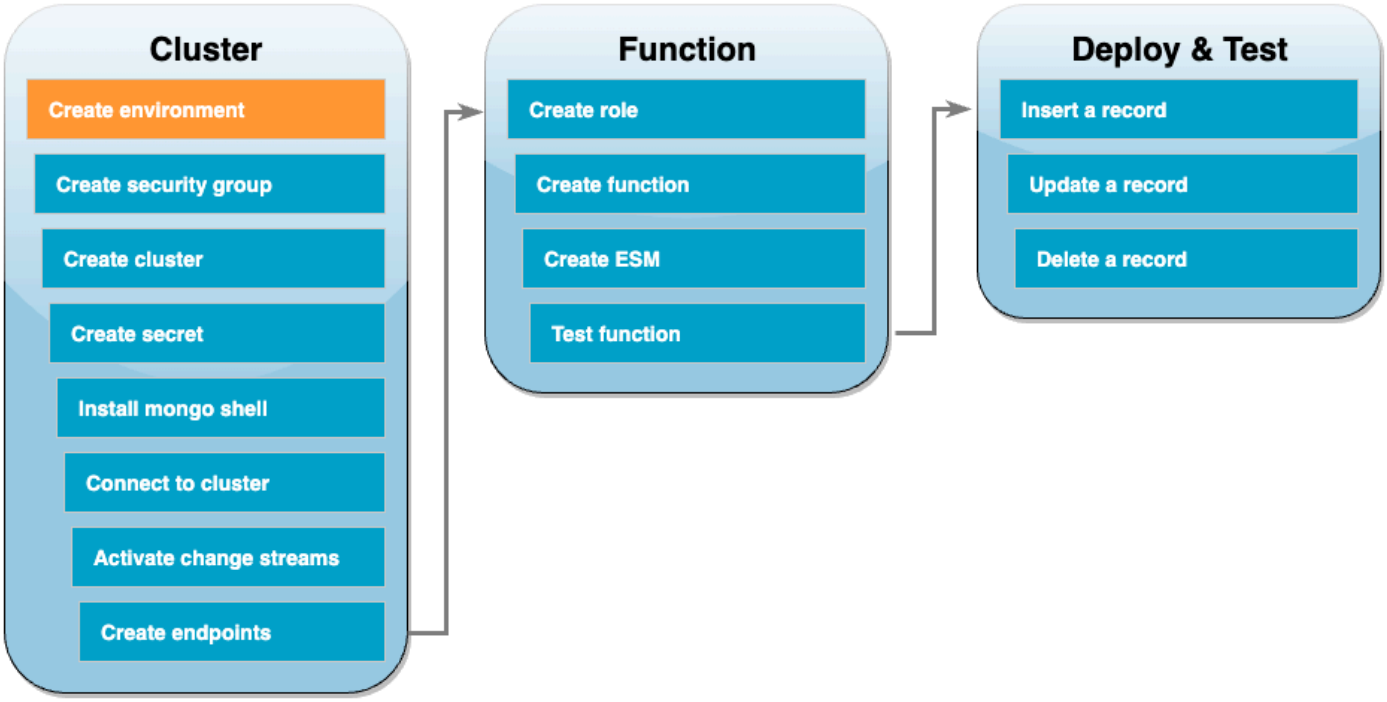
Installa il AWS Command Line Interface

Se non l'hai ancora installato AWS Command Line Interface, segui i passaggi indicati in Installazione o aggiornamento della versione più recente di 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 versione integrata su Windows di Ubuntu e Bash, installa il sottosistema Windows per Linux.

Crea l' AWS Cloud9 ambiente



Prima di creare la funzione Lambda, devi creare e configurare il cluster Amazon DocumentDB. I passaggi per configurare il cluster di questo tutorial si basano sulla procedura descritta nella Guida introduttiva ad Amazon DocumentDB.

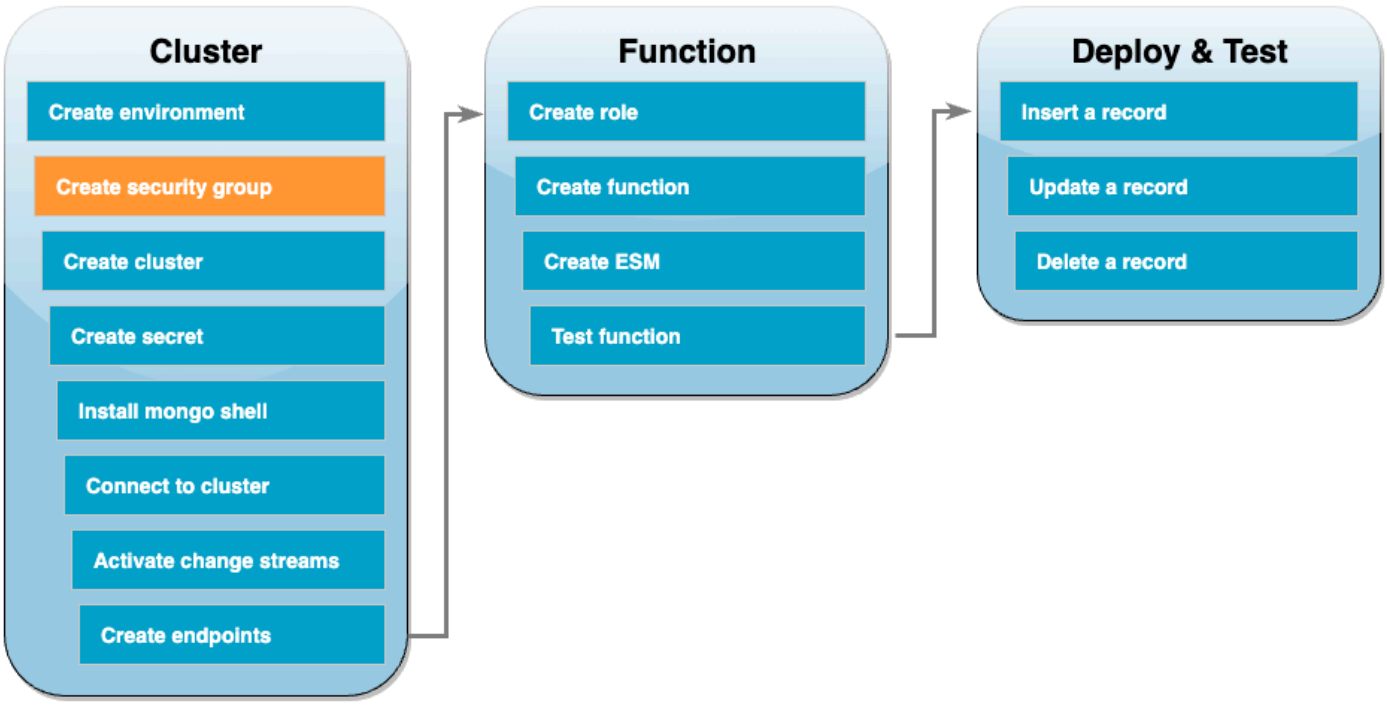
> **Note**
>
> Se hai già configurato un cluster Amazon DocumentDB, assicurati di attivare i flussi di modifica e di creare gli endpoint VPC di interfaccia necessari. Dopodiché, puoi passare direttamente ai passaggi di creazione della funzione.

Innanzitutto, crea un AWS Cloud9 ambiente. Utilizzerai l'ambiente durante questo tutorial per connetterti al cluster DocumentDB e interrogarlo.

Per creare un AWS Cloud9 ambiente

1. Apri la console Cloud9 e scegli Crea ambiente.
2. Crea un ambiente con la seguente configurazione:
   - In Dettagli:
     - Nome: `DocumentDBCloud9Environment`
     - Tipo di ambiente: nuova istanza EC2
   - In Nuova istanza EC2:
     - Tipo di istanza: `t2.micro` (1 GiB di RAM + 1 vCPU)
     - Piattaforma: Amazon Linux 2
     - Timeout: 30 minuti
   - In Impostazioni di rete:
     - Connessione — AWS Systems Manager (SSM)
     - Espandi il menu a discesa Impostazioni VPC.
     - Cloud privato virtuale (VPC) Amazon: scegli il VPC predefinito.
     - Sottorete: nessuna preferenza
   - Mantieni tutte le altre impostazioni predefinite.
3. Scegli Crea. Il provisioning del nuovo AWS Cloud9 ambiente può richiedere diversi minuti.

Creazione del gruppo di sicurezza EC2



Successivamente, crea un gruppo di sicurezza EC2 con regole che consentano il traffico tra il cluster DocumentDB e l'ambiente Cloud9.

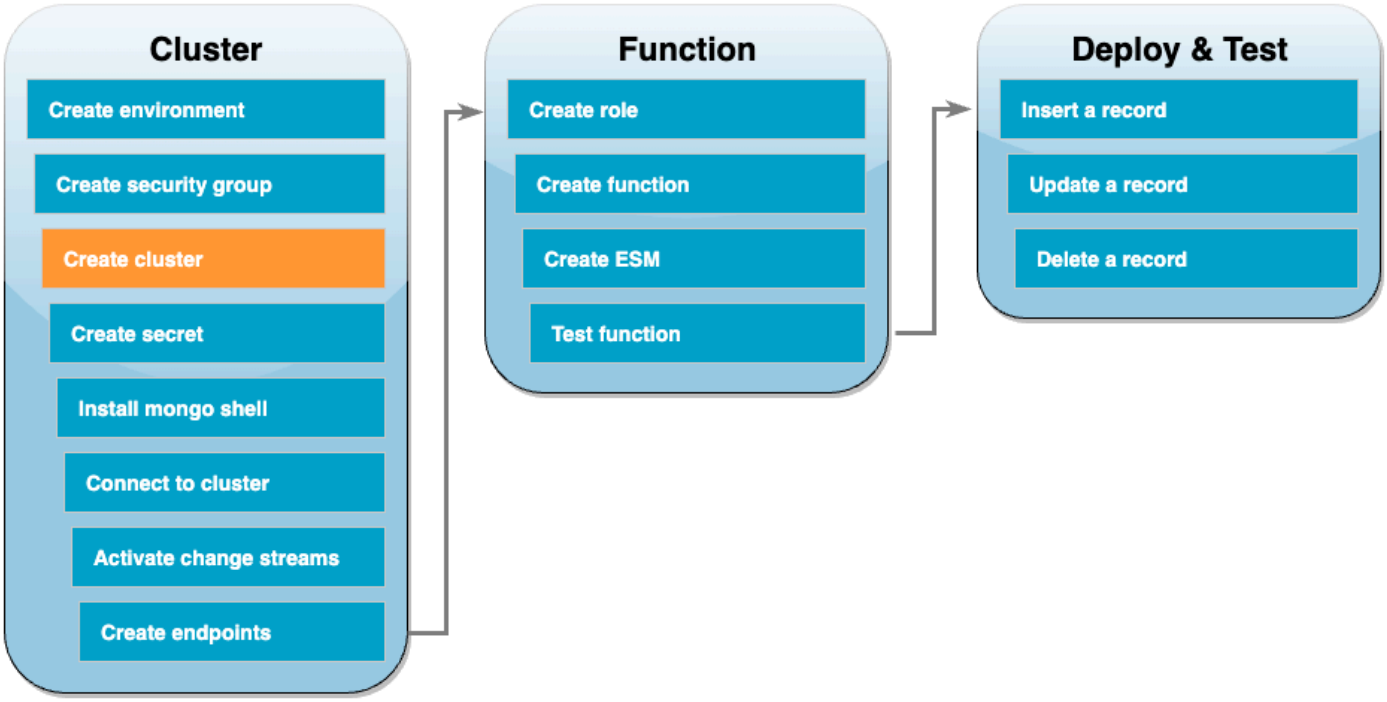
Creazione di un gruppo di sicurezza EC2

1. Apri la console EC2. In Rete e sicurezza, scegli Gruppi di sicurezza.
2. Scegliere Create Security Group (Crea gruppo di sicurezza).
3. Crea un gruppo di sicurezza con la seguente configurazione:
   - In Dettagli di base:
     - Nome del gruppo di sicurezza: `DocDBTutorial`
     - Descrizione: gruppo di sicurezza per il traffico tra Cloud9 e DocumentDB.
     - VPC: scegli il VPC predefinito.
   - Per Inbound rules (Regole in entrata), scegliere Add rule (Aggiungi regola). Creare una regola con la seguente configurazione:
     - Tipo: TCP personalizzato
     - Intervallo di porte: 27017
     - Origine: personalizzata

- Nella casella di ricerca accanto a Source, scegli il gruppo di sicurezza per l' AWS Cloud9 ambiente creato nel passaggio precedente. Per visualizzare un elenco dei gruppi di sicurezza disponibili, inserisci `cloud9` nella casella di ricerca. Scegli il gruppo di sicurezza denominato `aws-cloud9-<environment_name>`.
- Mantieni tutte le altre impostazioni predefinite.

4. Scegliere Create Security Group (Crea gruppo di sicurezza).

Creazione del cluster DocumentDB



In questo passaggio creerai un cluster DocumentDB utilizzando il gruppo di sicurezza del passaggio precedente.

Creazione di un cluster DocumentDB

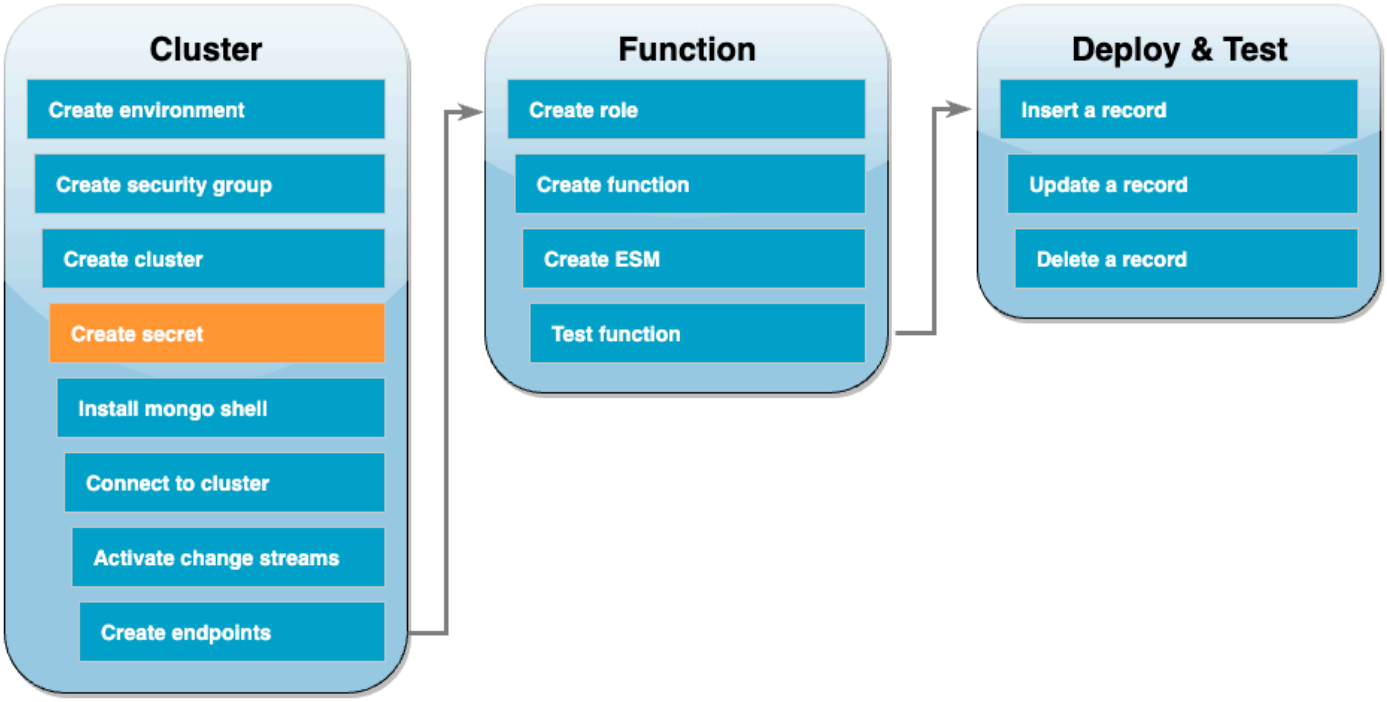
1. Apri la console DocumentDB. In Cluster, scegli Crea.
2. Crea un cluster con la seguente configurazione:
   - Per Tipo di cluster, scegli Cluster basato su istanze.
   - In Configurazione:
     - Versione del motore: 5.0.0

- Classe di istanza: db.t3.medium (idonea alla prova gratuita)
- Numero di istanze: 1.

- In Autenticazione:
  - Inserisci il nome utente e la password necessari per connetterti al tuo cluster (le stesse credenziali utilizzate per creare il segreto nel passaggio precedente). In Conferma password, conferma la password.
- Attiva Mostra impostazioni avanzate.
- In Impostazioni di rete:
  - Cloud privato virtuale (VPC): scegli il VPC predefinito.
  - Gruppo di sottoreti: predefinito
  - Gruppi di sicurezza VPC: oltre a `default (VPC)`, scegli il gruppo di sicurezza `DocDBTutorial (VPC)` che hai creato nel passaggio precedente.
- Mantieni tutte le altre impostazioni predefinite.

3. Scegli Create cluster (Crea cluster). Il provisioning del cluster DocumentDB può richiedere alcuni minuti.

Creazione del segreto di Gestione dei segreti



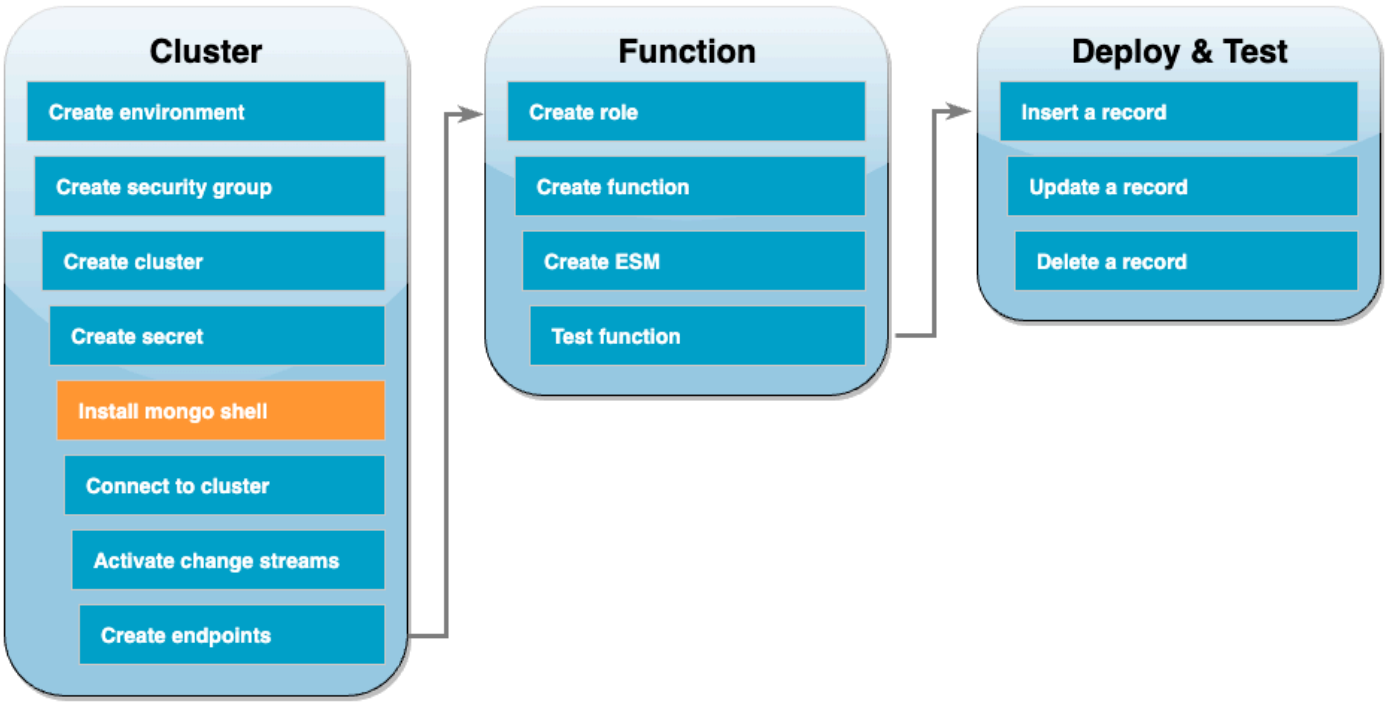
Per accedere manualmente al cluster DocumentDB, è necessario fornire le credenziali di nome utente e password. Affinché Lambda possa accedere al cluster, è necessario fornire un segreto di Gestione dei segreti che contenga le stesse credenziali di accesso utilizzate durante la configurazione della mappatura dell'origine degli eventi. In questo passaggio, creerai questo segreto.

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 il nome utente e la password che utilizzerai per accedere al cluster DocumentDB.
     - Database: scegli il tuo cluster DocumentDB.
     - Seleziona Successivo.
3. Per Configura segreto, scegli tra le seguenti opzioni:
   - Nome del segreto: `DocumentDBSecret`
   - Seleziona Successivo.
4. Seleziona Successivo.
5. Scegli Store.
6. Aggiorna la console per verificare di aver archiviato correttamente il segreto `DocumentDBSecret`.

Prendi nota dell'ARN del segreto del tuo segreto. Lo utilizzerai in un passaggio successivo.

Installazione della shell Mongo



In questo passaggio, installerai la shell (interprete di comandi) mongo nel tuo ambiente Cloud9. La shell mongo è un'utilità da riga di comando che si utilizza per connettersi al cluster DocumentDB e interrogarlo.

Installazione della shell mongo nell'ambiente Cloud9

1. Apri la console Cloud9. Accanto all'ambiente `DocumentDBCloud9Environment` creato in precedenza, fai clic sul link Apri nella colonna IDE di Cloud9.
2. In una finestra del terminale, crea il file di repository MongoDB con il comando seguente:

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://www.mongodb.org/static/pgp/server-5.0.asc" | sudo tee /etc/
yum.repos.d/mongodb-org-5.0.repo
```

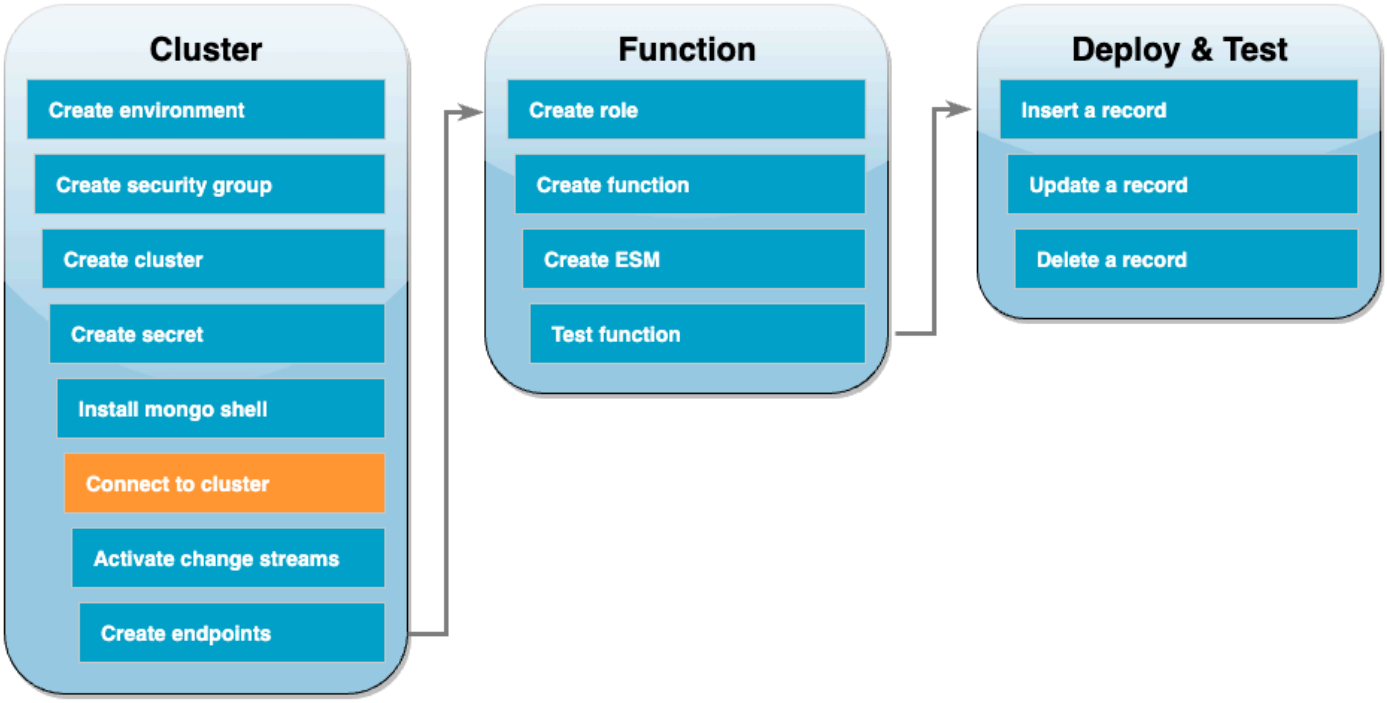
3. Quindi, installa la shell mongo con il comando seguente:

```
sudo yum install -y mongodb-org-shell
```

4. Per crittografare i dati in transito, scarica la chiave pubblica per Amazon DocumentDB. Il comando seguente scarica un file denominato `global-bundle.pem`:

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

Connessione al cluster DocumentDB



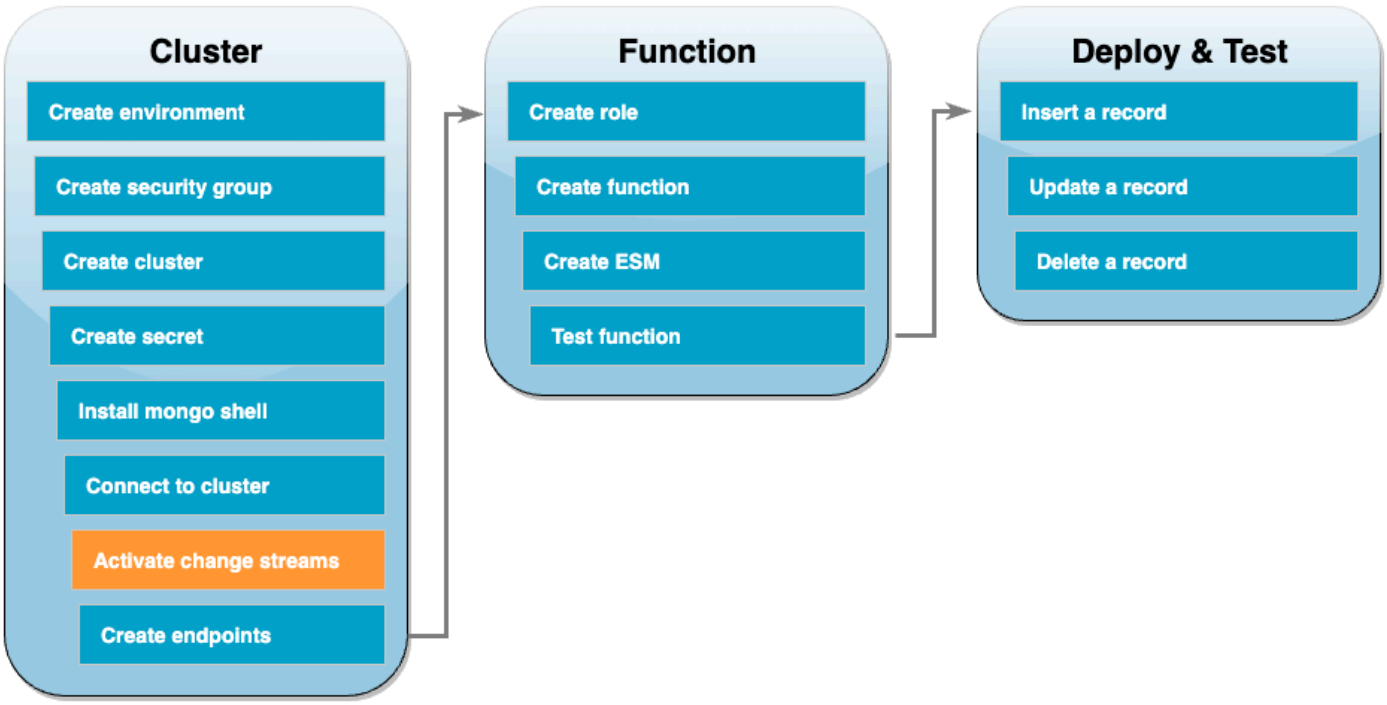
Ora puoi connetterti al cluster DocumentDB usando la shell mongo.

Connessione al cluster DocumentDB

1. Apri la console DocumentDB. In Cluster, scegli il cluster selezionando il relativo identificatore.
2. Nella scheda Connettività e sicurezza, in Connettiti a questo cluster con la shell mongo, scegli Copia.
3. Nell'ambiente Cloud9, incolla questo comando nel terminale. Sostituisci `<insertYourPassword>` con la password corretta.

Dopo aver inserito questo comando, se il prompt dei comandi diventa `rs0:PRIMARY>`, allora la connessione al cluster Amazon DocumentDB è stata stabilita.

Attivazione dei flussi di modifica



Per questo tutorial, monitorerai le modifiche alla raccolta di `products` del database `docdbdemo` nel cluster DocumentDB. Puoi farlo attivando i flussi di modifica. Innanzitutto, crea il database `docdbdemo` e testalo inserendo un record.

Creazione di un nuovo database all'interno del cluster

1. Nell'ambiente Cloud9, assicurati che la connessione al cluster DocumentDB sia ancora attiva.
2. In una finestra del terminale, utilizza il comando seguente per creare un nuovo database denominato `docdbdemo`:

   ```
   use docdbdemo
   ```

3. Quindi, utilizza il comando seguente per inserire un record in `docdbdemo`:

   ```
   db.products.insert({"hello":"world"})
   ```

   L'output visualizzato dovrebbe essere di questo tipo:

   ```
   WriteResult({ "nInserted" : 1 })
   ```

4. Utilizza il seguente comando per elencare tutti i database:

```
show dbs
```

Assicurati che l'output contenga il database `docdbdemo`:

```
docdbdemo  0.000GB
```

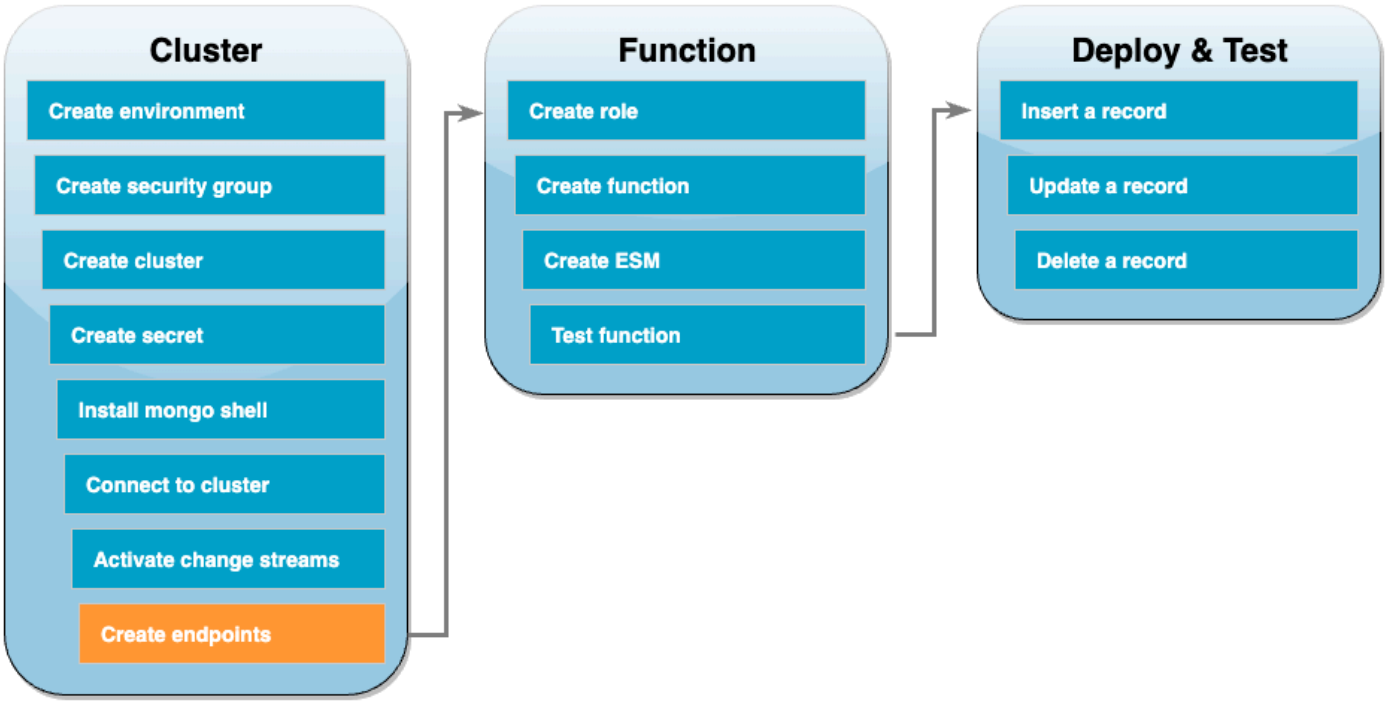
Quindi, attiva i flussi di modifica sulla raccolta `products` del database `docdbdemo` utilizzando il comando seguente:

```
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.

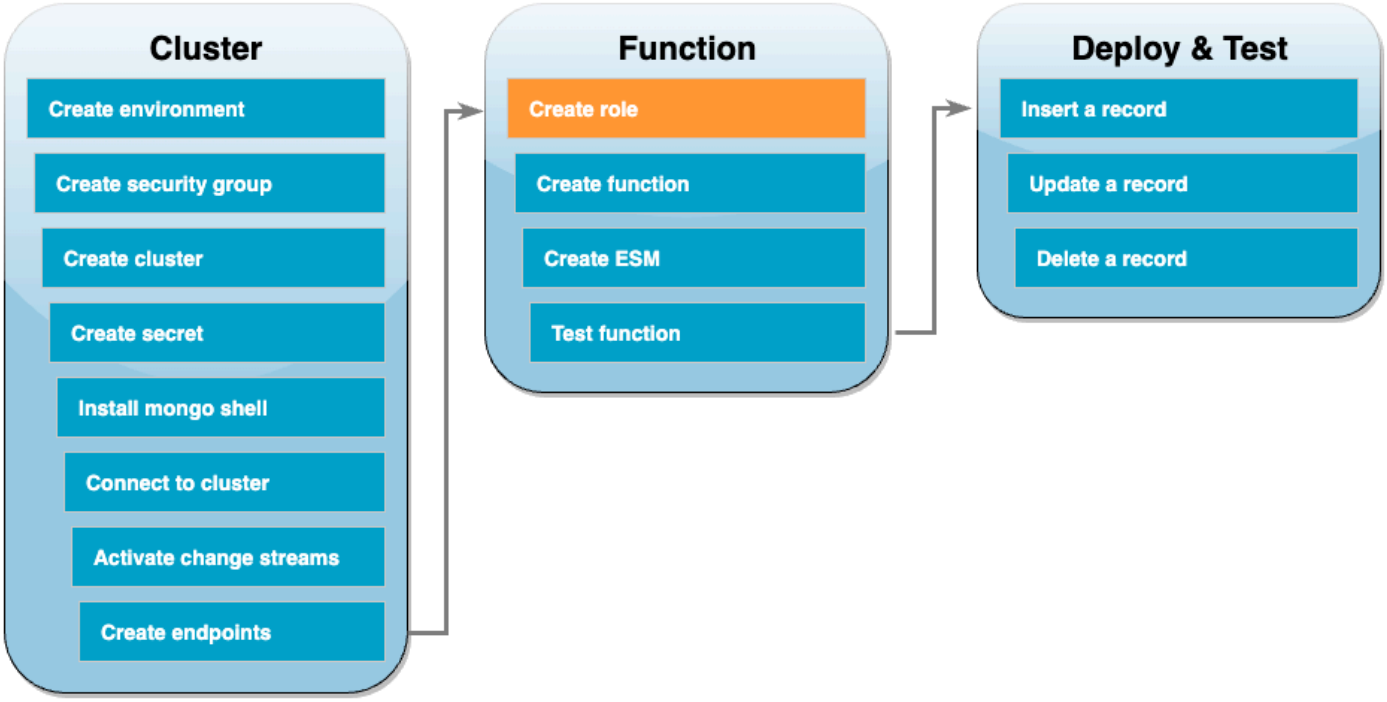
Creazione di endpoint VPC dell'interfaccia

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 di 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 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 Gruppi di sicurezza, scegli il gruppo di sicurezza VPC predefinito (nome del gruppo di `default`) e il gruppo di sicurezza che hai creato in precedenza (nome del gruppo di `DocDBTutorial`).
   - 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 Categoria di servizio, 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 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 Gruppi di sicurezza, scegli il gruppo di sicurezza VPC predefinito (nome del gruppo di `default`) e il gruppo di sicurezza che hai creato in precedenza (nome del gruppo di `DocDBTutorial`).

- 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 serie di passaggi successiva, creerai la 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.

```
{
    "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"
        }
    ]
}
```

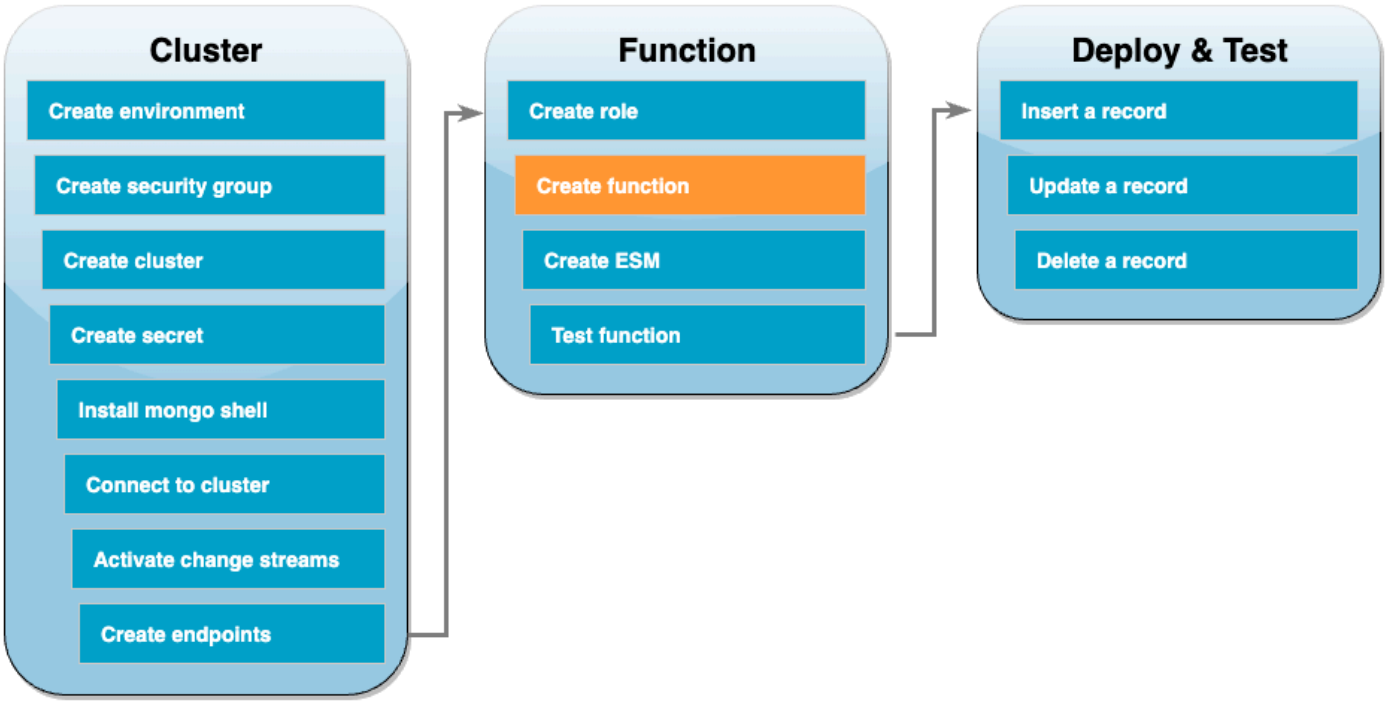
3. Scegli Successivo: Tag, quindi Successivo: Verifica.
4. In Nome, inserisci `AWSDocumentDBLambdaPolicy`.
5. Scegli Create Policy (Crea policy).

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 affidabile: AWS servizio
   - Caso d'uso: Lambda
   - Seleziona Successivo.
3. Per Aggiungi autorizzazioni, scegli la `AWSDocumentDBLambdaPolicy` policy che hai appena creato e `AWSLambdaBasicExecutionRole` autorizza la funzione a scrivere su Amazon CloudWatch Logs.
4. Seleziona Successivo.
5. Per Nome ruolo, inserisci `AWSDocumentDBLambdaExecutionRole`.
6. Scegli Crea ruolo.

Creazione della funzione Lambda



Il seguente codice di esempio riceve un input di evento DocumentDB ed elabora il messaggio in esso contenuto.

Go

SDK per Go V2

> **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.

Consumo di un evento Amazon DocumentDB con Lambda utilizzando Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

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))
}
```

JavaScript

SDK per (v3 JavaScript)

> **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));
};
```

Python

SDK per Python (Boto3)

> **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 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 di più su. 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 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
```

Creazione della funzione Lambda

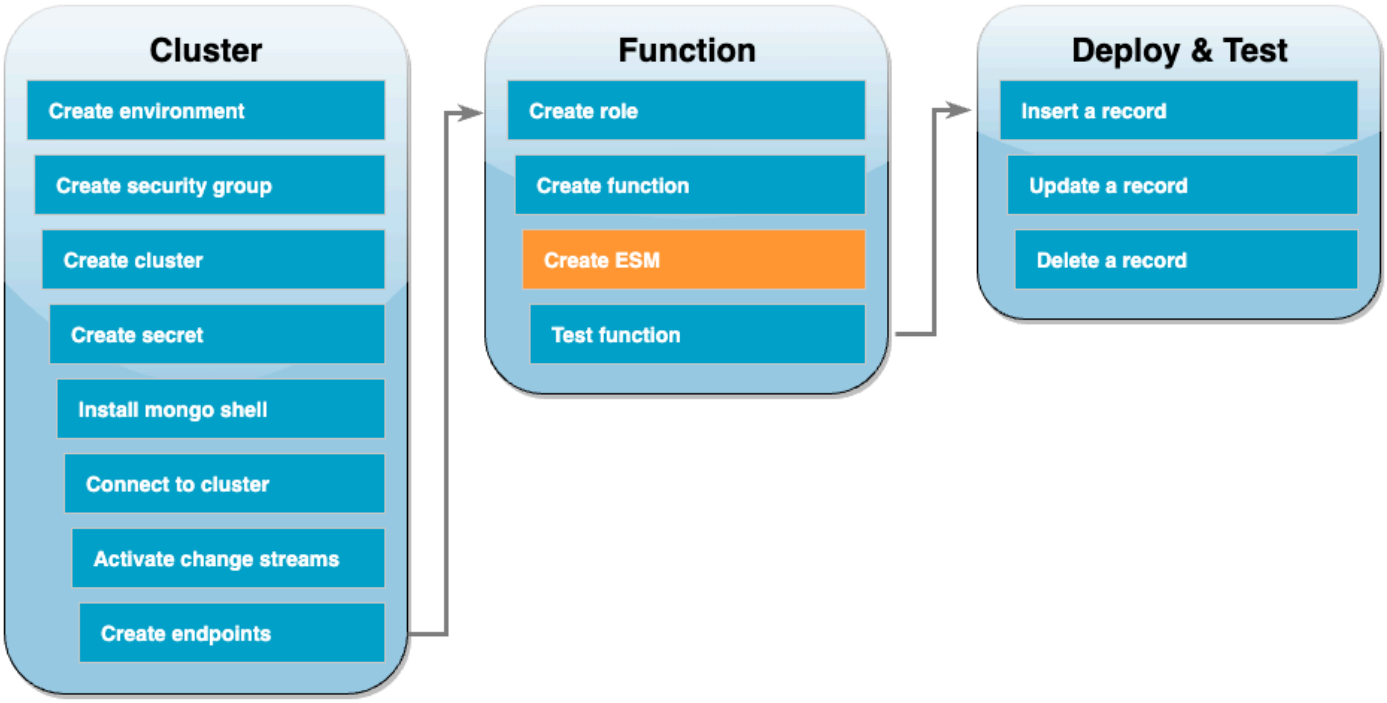
1. Copiare il codice di esempio in un file denominato `index.js`.
2. Crea un pacchetto di implementazione utilizzando il seguente comando.

```
zip function.zip index.js
```

3. Utilizza il seguente comando della CLI per creare la funzione. Sostituisci `us-east-1` con la regione e `123456789012` con l'ID account in uso.

```
aws lambda create-function --function-name ProcessDocumentDBRecords \
    --zip-file fileb://function.zip --handler index.handler --runtime nodejs20.x \
    --region us-east-1 \
    --role arn:aws:iam::123456789012:role/AWSDocumentDBLambdaExecutionRole
```

Creazione della mappatura dell'origine degli eventi Lambda



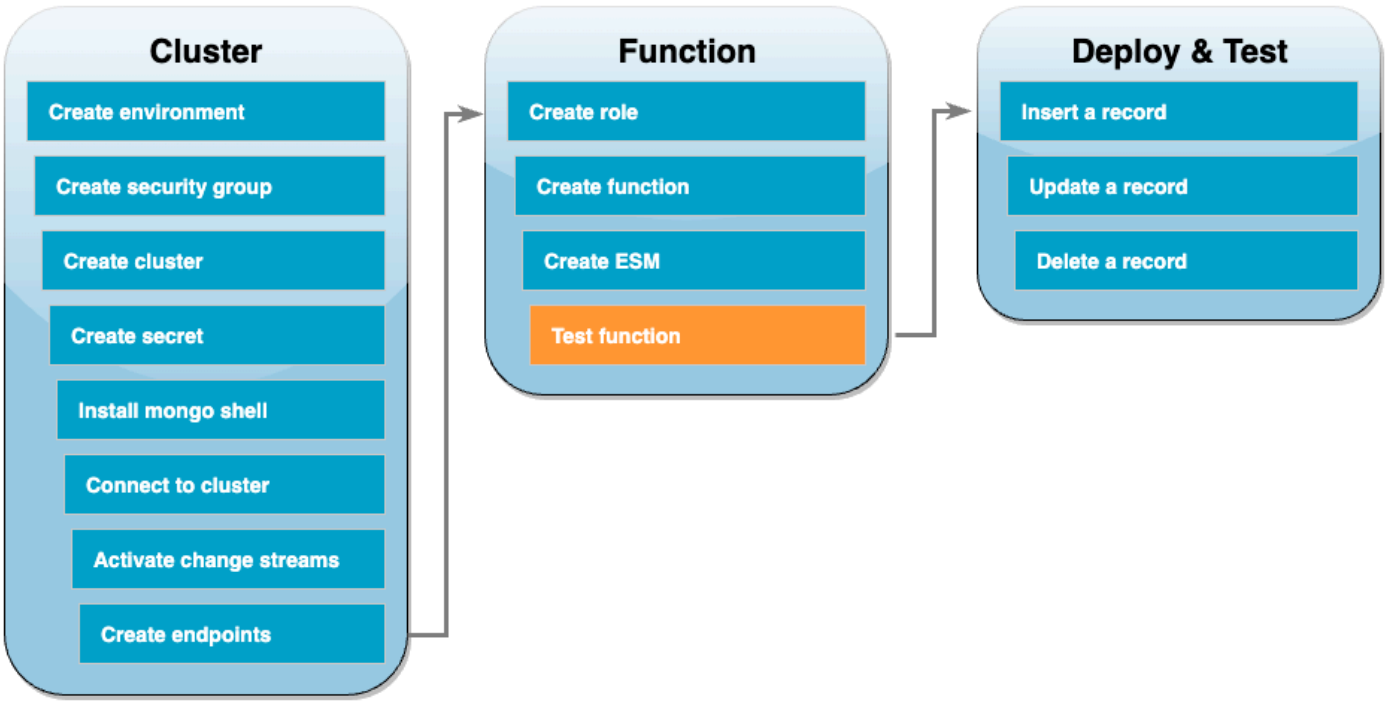
Crea la mappatura dell'origine degli eventi che associa il flusso di modifica DocumentDB alla funzione Lambda. Dopo aver creato questa mappatura delle sorgenti degli eventi, inizia AWS Lambda immediatamente il polling dello stream.

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 DocumentDB.
6. Crea la mappatura dell'origine degli eventi con la seguente configurazione:
   - Cluster DocumentDB: scegli il cluster che hai creato in precedenza.
   - Nome del database: `docdbdemo`
   - Nome della collezione: prodotti
   - Dimensione batch: 1
   - Posizione di partenza: ultima
   - Autenticazione: BASIC_AUTH
   - Chiave di Gestione dei segreti: scegli il `DocumentDBSecret` che hai appena creato.
   - Finestra batch: 1
   - Configurazione completa del documento — UpdateLookup
7. Scegli Aggiungi. La creazione della mappatura dell'origine degli eventi può richiedere alcuni minuti.

Test della funzione: richiamo manuale



Per verificare di aver creato correttamente la funzione e la mappatura dell'origine degli eventi, richiama la funzione utilizzando il comando `invoke`. Per fare ciò, copia prima il seguente evento JSON in un file denominato `input.txt`:

```
{
  "eventSourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:canaryclusterb2a659a2-
qo5tcmqkcl03",
  "events": [
    {
      "event": {
        "_id": {
          "_data": "0163eeb6e7000000090100000009000041e1"
        },
        "clusterTime": {
          "$timestamp": {
            "t": 1676588775,
            "i": 9
          }
        },
        "documentKey": {
          "_id": {
            "$oid": "63eeb6e7d418cd98afb1c1d7"
          }
        },
        "fullDocument": {
          "_id": {
            "$oid": "63eeb6e7d418cd98afb1c1d7"
          },
          "anyField": "sampleValue"
        },
        "ns": {
          "db": "docdbdemo",
          "coll": "products"
        },
        "operationType": "insert"
      }
    }
  ],
  "eventSource": "aws:docdb"
}
```

Dopodiché, utilizza il comando seguente per richiamare la funzione con questo evento:

```
aws lambda invoke --function-name ProcessDocumentDBRecords \
    --cli-binary-format raw-in-base64-out \
    --region us-east-1 \
    --payload file://input.txt out.txt
```

La risposta visualizzata sarà simile alla seguente:

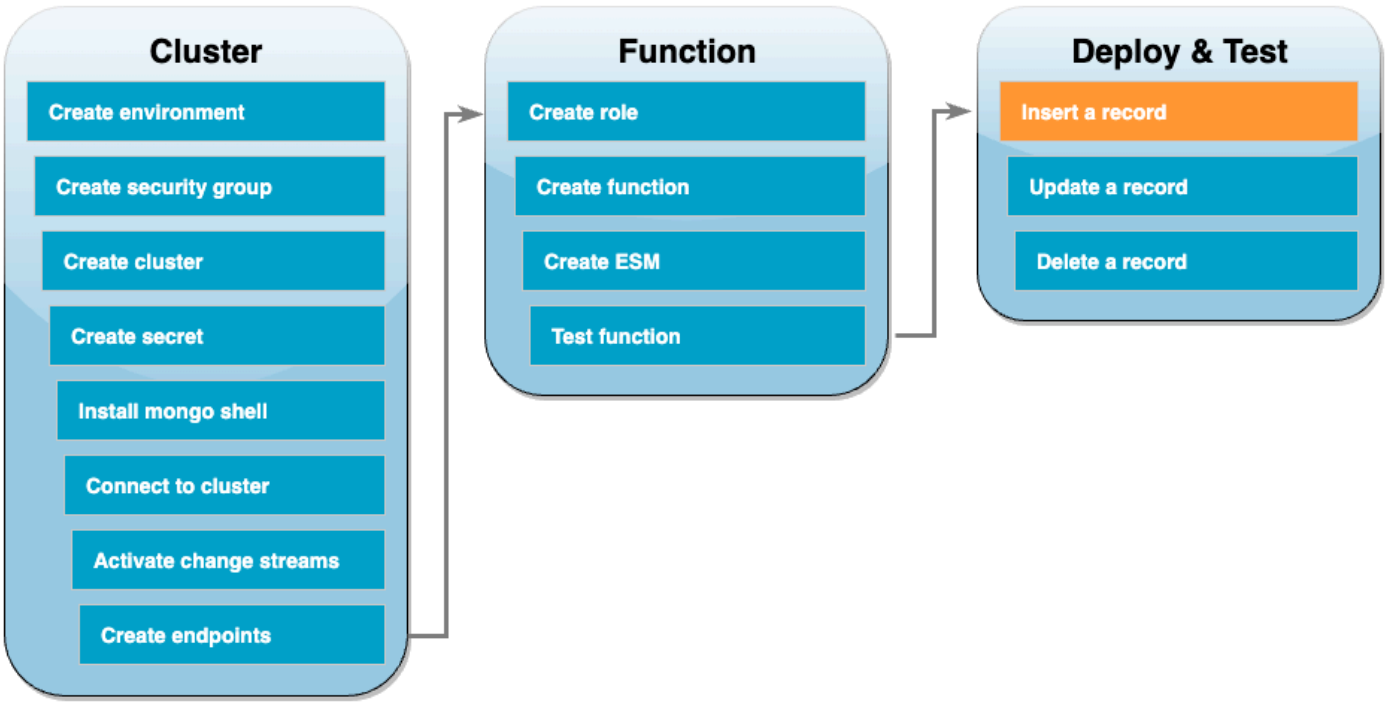
```
{
   "StatusCode": 200,
   "ExecutedVersion": "$LATEST"
}
```

È possibile verificare che la funzione abbia elaborato correttamente l'evento controllando CloudWatch i registri.

Per verificare la chiamata manuale tramite Logs CloudWatch

1. Apri la pagina Funzioni della console Lambda.
2. Scegli la scheda Monitor, quindi scegli Visualizza registri. CloudWatch Questo ti porta al gruppo di log specifico associato alla tua funzione nella CloudWatch console.
3. Scegli il flusso di log più recente. All'interno dei messaggi di log, dovresti visualizzare l'evento JSON.

Test della funzione: inserimento di un record



Metti alla prova la tua end-to-end configurazione interagendo direttamente con il tuo database DocumentDB. Nella serie di passaggi successiva, inserirai un record, lo aggiornerai e quindi lo eliminerai.

Inserimento di un record

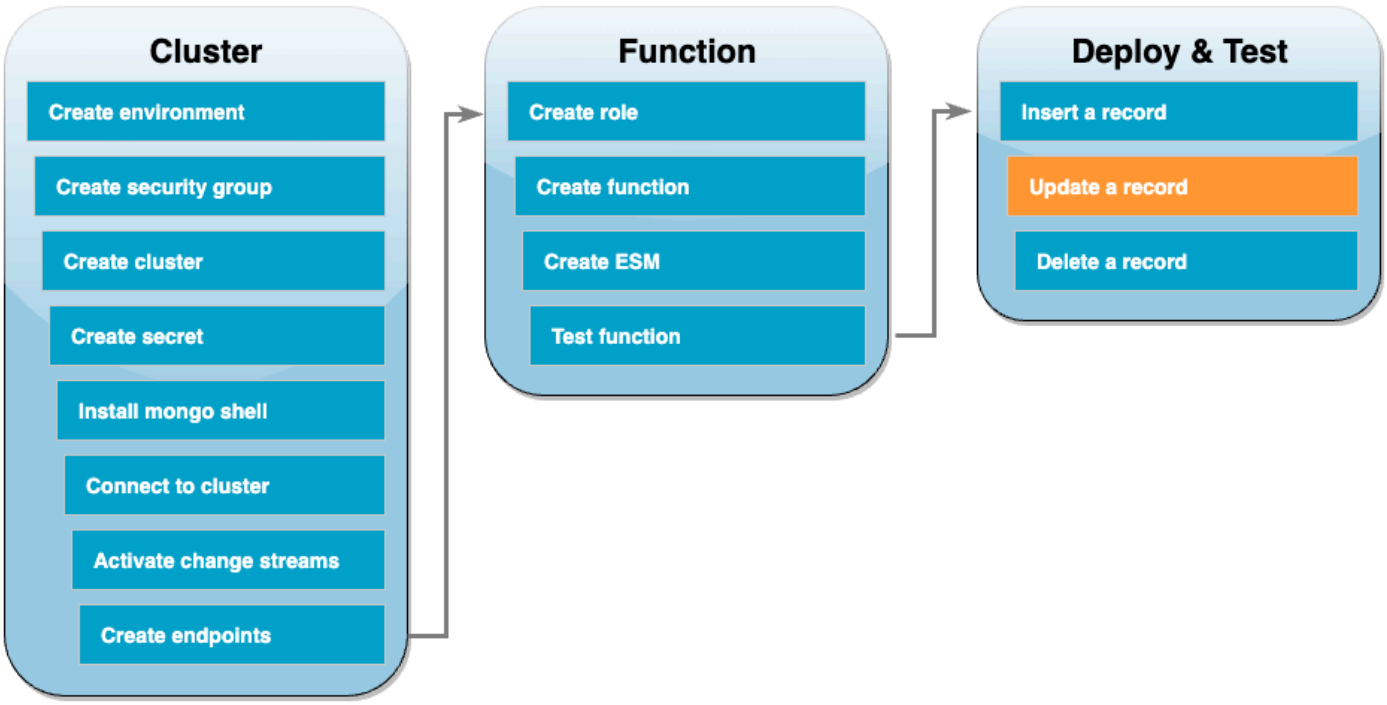
1. Riconnettiti al cluster DocumentDB nel tuo ambiente Cloud9.
2. Utilizza questo comando per verificare che stai attualmente utilizzando il database `docdbdemo`:

```
use docdbdemo
```

3. Inserisci un record nella raccolta `products` del database `docdbdemo`:

```
db.products.insert({"name":"Pencil", "price": 1.00})
```

Test della funzione: aggiornamento di un record

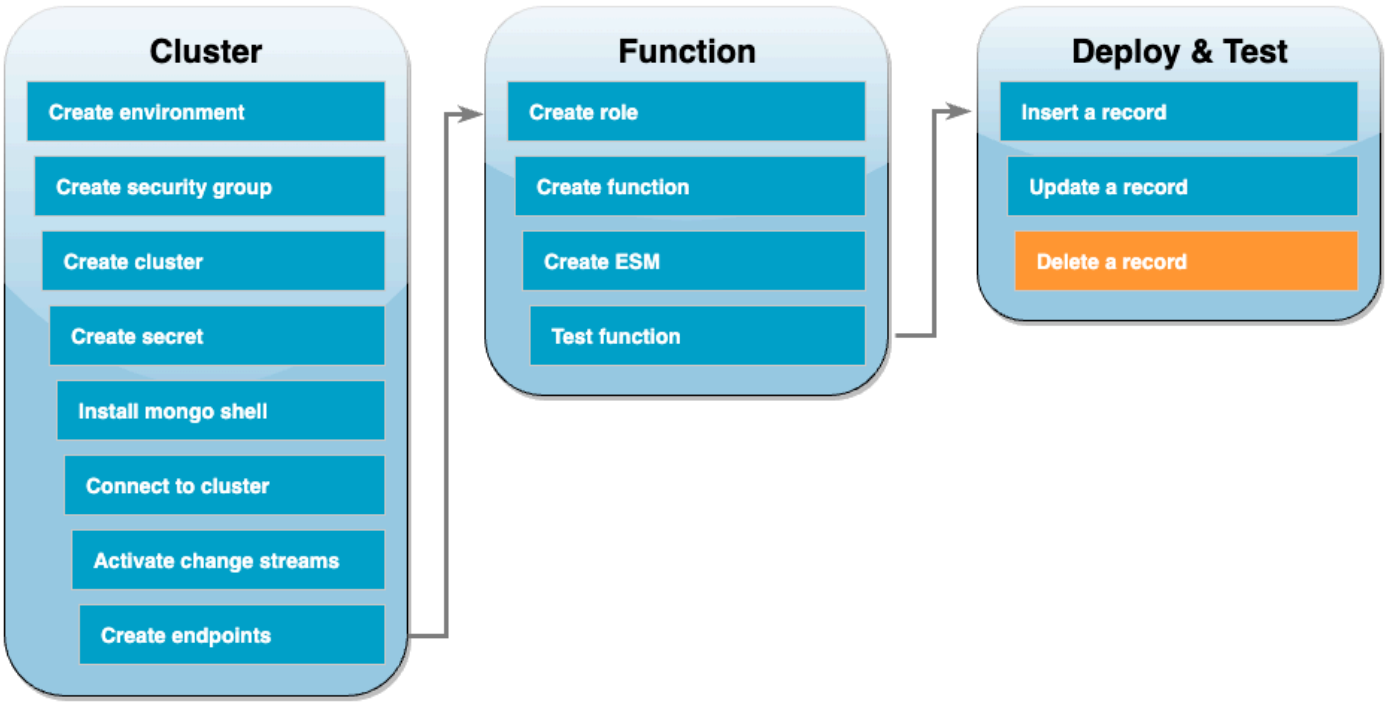


Successivamente, aggiorna il record appena inserito con il comando seguente:

```
db.products.update(
    { "name": "Pencil" },
    { $set: { "price": 0.50 }}
)
```

Verifica che la tua funzione abbia elaborato correttamente questo evento CloudWatch controllando Logs.

Test della funzione: eliminazione di un record



Infine, elimina il record appena aggiornato con il seguente comando:

```
db.products.remove( { "name": "Pencil" } )
```

Verifica che la tua funzione abbia elaborato correttamente questo evento controllando CloudWatch Logs.

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 **`delete`** 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 DocumentDB.
2. Scegli il cluster DocumentDB che hai creato per questo tutorial e disabilita la protezione dall'eliminazione.
3. Nella pagina principale Cluster, scegli nuovamente il tuo cluster 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).

Eliminazione del gruppo di sicurezza Amazon EC2

1. Apri la console EC2. In Rete e sicurezza, scegli Gruppi di sicurezza.
2. Seleziona il gruppo di sicurezza creato per questo tutorial.
3. Scegli Operazioni, Elimina gruppi di sicurezza.
4. Scegli Elimina.

Eliminazione dell'ambiente Cloud9

1. Apri la console Cloud9.
2. Seleziona l'ambiente creato per questo tutorial.
3. Scegli Elimina.
4. Inserisci **`delete`** nel campo di immissione del testo.
5. Scegli Delete (Elimina).

Filtro eventi Lambda

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 con le mappature dell'origine degli eventi. È possibile aggiungere filtri alle mappature delle sorgenti degli eventi per i seguenti servizi: AWS

- Amazon DynamoDB
- Flusso di dati Amazon Kinesis
- Amazon MQ
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Apache Kafka gestito dal cliente
- Amazon Simple Queue Service (Amazon SQS)

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 degli eventi, puoi richiedere un aumento della quota fino a 10 filtri per ciascuna origine degli eventi. Se tenti di aggiungere più filtri di quelli consentiti dalla quota corrente, Lambda genera un errore quando tenti di creare l'origine degli eventi.

Argomenti

Nozioni di base del filtro evento

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 di chi funge da origine dell'evento. Servizio AWS 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": "AIDAIENQZJOLO23YVJ4VO",
        "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 `messageIDeventSourceArn` 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 nel modello del filtraggio. Per un elenco delle chiavi di filtraggio dei dati e per vedere esempi di modelli di filtro per ciascuna delle chiavi supportate Servizio AWS, fare 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 vengono gestiti i record che non soddisfano il filtro specificato dipende dall'origine dell'evento.

- 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 elaborano 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 Apache Kafka gestito dal cliente, 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 EventBridge le regole di 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]

Operatore di confronto	Esempio	Sintassi delle regole
Empty	LastName è vuoto	"LastName": [""]
Equals	Il nome è "Alice"	"Nome": ["Alice"]
Uguale a (ignora maiuscole e minuscole)	Il nome è "Alice"	«Nome»: [{"equals-ignore-case«: «alice»}]
And	La posizione è "New York" e il giorno è "lunedì"	"Luogo": ["New York"], "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] }]
Numerico (intervallo)	Il prezzo è superiore a 10 e inferiore o uguale a 20	"Prezzo": [{ "numerico": [">", 10, "<=", 20] }]
Exists	ProductName esiste	"ProductName«: [{«exists»: vero}]
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 la corrispondenza character-by-character esatta senza ripiegamento tra 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.

Nota che l'operatore Exists funziona solo sui nodi leaf nel codice JSON di origine dell'evento. Non corrisponde ai nodi intermedi. Ad esempio, con il seguente codice JSON, il pattern di filtro non `{ "person": { "address": [ { "exists": true } ] } }` 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.

Collegamento dei criteri di filtro a una mappatura dell'origine 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 dell'origine degli 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 ]}"}]}'
```

Questo create-event-source-mappingcomando crea una nuova mappatura delle sorgenti di eventi Amazon SQS per una funzione `my-function` con quanto specificato. `FilterCriteria`

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": "{ \"Metadata\" : [ 1, 2 ]}"}]}'
```

Si noti che per aggiornare una mappatura di origine eventi, è necessario il suo UUID. È possibile ottenere l'UUID da una chiamata. list-event-source-mappings Lambda restituisce anche l'UUID nella risposta CLI. create-event-source-mapping

Per rimuovere i criteri di filtro da un'origine di eventi, puoi eseguire il update-event-source-mappingcomando seguente con un oggetto vuoto. `FilterCriteria`

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria "{}"
```

Per altri esempi di creazione di filtri di eventi utilizzando il AWS CLI, consultaUtilizzo di filtri con diversi Servizi AWS.

Collegamento dei criteri di filtro a una mappatura dell'origine evento (AWS SAM)

Supponiamo di voler configurare un'origine 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 delle sorgenti di un evento, consultate la EventSourcesezione della Guida per gli AWS SAM sviluppatori. Per altri esempi di creazione di filtri di eventi utilizzando AWS SAM modelli, consulta. Utilizzo di filtri con diversi Servizi AWS

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 nel modello del filtraggio. La tabella seguente fornisce le chiavi di filtro per ogni supporto Servizio AWS.

Servizio AWS	Chiave di filtraggio
DynamoDB	`dynamodb`
Kinesis	`data`
Amazon MQ	`data`
MSK Amazon	`value`
Apache Kafka gestito dal cliente	`value`
Amazon SQS	`body`

Le sezioni seguenti forniscono esempi di modelli di filtraggio per diversi tipi di origine degli eventi. Forniscono inoltre definizioni dei formati di dati in entrata supportati e dei formati del corpo dei modelli di filtraggio per ciascun servizio supportato.

Filtraggio con DynamoDB

Supponiamo di avere una tabella DynamoDB con la chiave primaria `CustomerName` e gli attributi `AccountManager` e `PaymentTerms`. Di seguito è riportato un esempio di record dal flusso della 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 sezioni seguenti 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 dell'origine degli 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-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"Keys\" :
 { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi 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'attributo `AccountManager` 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 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" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat Candella",
 "Shirley Rodriguez" ] } } } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli 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, aggiungi 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 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.

```
{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" :
 [ "Terry Whitlock" ] } } } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli 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, aggiungi 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 filtraggio degli eventi DynamoDB non supporta l'uso di operatori numerici (uguali numerici 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 con DynamoDB

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 nell'evento JSON, quindi se il modello di filtro utilizza Exists per testare un nodo intermedio, non funzionerà. Considerate 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 non `"Organizations"` è un nodo foglia. L'esempio seguente mostra come utilizzare correttamente l'operatore Exists per costruire il pattern di filtro desiderato:

```
{ "dynamodb" : { "NewImage" : {"Organizations": {"L": {"S": [ {"exists":
 true } ] } } } } }
```

Formato JSON per il filtraggio 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 proprietà dei dati deve essere in un formato JSON valido.
Non-JSON	JSON valido	Lambda rilascia il registro.

Formato dei dati in entrata	Formato del modello di filtro per le proprietà di dati	Operazione risultante
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.

Filtraggio con 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":
 "eyJSZWNvcmROdW1iZXIiOiAiMDAwMSIsICJUaW1lU3RhbXAiOiAieXl5eS1tbS1kZFRoaDptbTpzcyIsICJSZXF1ZXN0Q
        "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"
}
```

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 "acquista", l'oggetto `FilterCriteria` dovrebbe avere la struttura 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 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.

```
{ "data" : { "order" : { "type" : [ "buy" ] } } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli 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-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"order\" : { \"type
\" : [ \"buy\" ] } } }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi 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.
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 la velocità di trasmissione effettiva 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 velocità di trasmissione effettiva dedicata 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 ulteriori informazioni sull'utilizzo di Lambda con il fan-out avanzato di Kinesis, consulta Incrementare le prestazioni di elaborazione dei flussi in tempo reale con Amazon Kinesis Data Streams Enhanced fan-out e Lambda sul blog di elaborazione. AWS AWS

Filtraggio con 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-
west-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":"QUJDOkFBQUE=",
    "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 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.

```
{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli 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: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-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :
 [ { \"numeric\": [ \">\", 0 ] } ] } }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML per la fonte dell'evento.

```
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 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.

```
{ "data" : [ { "prefix": "Result: " } ] }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli 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: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-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : [ { \"prefix\":
 \"Result: \" } ] }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML per la fonte dell'evento.

```
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 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.

Formato messaggio in arrivo	Formato del modello di filtro per le proprietà di messaggi	Operazione risultante
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.

Filtraggio con Amazon MSK e Apache Kafka autogestito

Supponiamo che un produttore stia scrivendo messaggi su un argomento nel tuo cluster Amazon MSK o Apache Kafka autogestito, 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 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.

```
{ "value" : { "device_ID" : [ { "prefix":  "AB" } ] } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli 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: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 per la fonte dell'evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : { "device_ID" : [ { "prefix":  "AB" } ] } }'
```

Con Amazon MSK e Apache Kafka autogestito, 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 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.

```
{ "value" : [ { "anything-but": [ "error" ] } ] }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli 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: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-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":
 [ \"error\" ] } ] }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML per la fonte dell'evento.

```
FilterCriteria:
```

```
Filters:
  - Pattern: '{ "value" : [ { "anything-but": [ "error" ] } ] }'
```

I messaggi Amazon MSK e Apache Kafka autogestito devono essere stringhe codificate in UTF-8, semplici o in formato JSON. Questo perché Lambda decodifica gli array di byte Amazon MSK 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 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.

Filtraggio con Amazon MQ

Supponiamo che la coda Amazon SQS contenga messaggi nel formato JSON seguente.

```
{
    "RecordNumber": 0000,
    "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": 0000,\n "TimeStamp": "yyyy-mm-ddThh:mm:ss",\n
 "RequestCode": "AAAA"\n}",
    "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
        "SenderId": "AIDAIENQZJOLO23YVJ4VO",
        "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 nei quali il `RequestCode` del messaggio Amazon SQS è "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 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" : { "RequestCode" : [ "BBBB" ] } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli 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-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RequestCode\" :
 [ \"BBBB\" ] } }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi 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 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.

```
{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli 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-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RecordNumber\" :
 [ { \"numeric\": [ \">\", 9999 ] } ] } }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi 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 **body** messaggio in arrivo	Formato **body** 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.

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.

Puoi anche accedere agli eventi di test salvati e non salvati nella scheda Code (codice). Da lì, scegli Test, quindi scegli l'evento di 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. Seleziona Salvataggio delle modifiche.

È inoltre possibile creare nuovi eventi di test sulla scheda Code (codice). Da lì, scegli Test, Configure test event (configura l'evento di test).

Creazione di eventi di test condivisibili

Gli eventi di test condivisibili sono eventi di test che è possibile 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 di sistema per memorizzare e chiamare gli eventi di test condivisibili creati, è consigliabile non modificare tale registro o creare un registro utilizzando il nome `lambda-testevent-schemas`.

Per visualizzare, condividere e modificare gli eventi di test condivisibili, devi disporre delle autorizzazioni per tutte le seguenti operazioni dell'API del registro degli schemi EventBridge (CloudWatch Events):

- schemas.CreateRegistry
- schemas.CreateSchema
- schemas.DeleteSchema
- schemas.DeleteSchemaVersion
- schemas.DescribeRegistry
- schemas.DescribeSchema
- schemas.GetDiscoveredSchema
- schemas.ListSchemaVersions
- schemas.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 necessarie 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. Seleziona Salvataggio delle modifiche.

Usa eventi di test condivisibili con AWS Serverless Application Model.

Puoi usare AWS SAM per richiamare eventi di test condivisibili. Consultare `sam remote test-event` nella Guida per gli sviluppatori di 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 si elimina la funzione, Lambda non elimina gli schemi di eventi di test condivisibili associati. È necessario pulire queste risorse manualmente dalla console (Events). EventBridge CloudWatch

Stati funzione Lambda

Per indicare quando la funzione è pronta per la chiamata, Lambda include un campo di stato nella configurazione della funzione per tutte le funzioni. `State` fornisce informazioni sullo stato corrente della funzione, incluso se sia possibile richiamare correttamente la funzione. Gli stati della funzione non modificano il comportamento delle chiamate della funzione o il modo in cui la funzione esegue il codice. 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 altre operazioni API che operano sulla funzione avranno 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.
- `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 torna allo stato inattivo. Se la funzione è bloccata nello stato inattivo, consulta la funzione `StatusCode` e `StatusCodeReason` gli attributi per un'ulteriore risoluzione dei problemi. Potrebbe essere necessario risolvere eventuali errori e ridistribuire la funzione per ripristinarla allo stato attivo.

Se utilizzi flussi di lavoro di automazione basati su SDK o richiami direttamente le API di servizio di Lambda, prima dell'invocazione assicurati di controllare lo stato di una funzione per verificare che sia attiva. Puoi farlo con l'azione GetFunctionAPI Lambda o configurando un cameriere utilizzando l'SDK for AWS Java 2.0.

```
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 l'aggiornamento

Lambda fornisce un contesto aggiuntivo per le funzioni in fase di aggiornamento con l'attributo `LastUpdateStatus`, che può avere i seguenti stati:

- `InProgress` – È in corso un aggiornamento su una funzione esistente. Mentre è in corso un aggiornamento della funzione, le chiamate vanno al codice e alla configurazione precedenti della funzione.
- `Successful` – L'aggiornamento è stato eseguito. Una volta che Lambda ha terminato l'aggiornamento, questo rimane impostato fino a un aggiornamento ulteriore.
- `Failed` – L'aggiornamento della funzione non è riuscito. Lambda interrompe l'aggiornamento e il codice e la configurazione precedenti della funzione rimangono disponibili.

Example

Quel che segue è il risultato di `get-function-configuration` su una funzione in fase di aggiornamento.

```
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "Runtime": "nodejs20.x",
    "VpcConfig": {
        "SubnetIds": [
            "subnet-071f712345678e7c8",
            "subnet-07fd123456788a036",
            "subnet-0804f77612345cacf"
```

```
        ],
        "SecurityGroupIds": [
            "sg-085912345678492fb"
        ],
        "VpcId": "vpc-08e1234569e011e83"
    },
    "State": "Active",
    "LastUpdateStatus": "InProgress",
    ...
}
```

FunctionConfiguration ha altri due attributi, `LastUpdateStatusReason` e `LastUpdateStatusReasonCode`, per contribuire a risolvere i problemi relativi all'aggiornamento.

Le operazioni seguenti non riescono mentre è in corso un aggiornamento asincrono:

- UpdateFunctionCode
- UpdateFunctionConfiguration
- PublishVersion
- TagResource

Comprensione del comportamento dei 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 Invocazione asincrona.
- 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 fonti di eventi basate su stream e code e gli argomenti specifici del servizio in Richiamare Lambda con eventi di altri servizi AWS.
- AWS services: 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 vengono richiesti tramite proxy a un utente o client upstream possono avere una strategia di riprova o inoltrare la risposta all'errore al richiedente. Ad esempio, API Gateway inoltra sempre la risposta di errore al richiedente.

Per la chiamata asincrona, il comportamento è lo stesso di quando si richiama la funzione in modo sincrono. Per ulteriori informazioni, consulta gli argomenti specifici del servizio in Richiamare Lambda con eventi di altri servizi AWS e l'invocazione della documentazione del servizio.

- 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 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 e risoluzione dei problemi delle funzioni Lambda.

Usa il rilevamento ricorsivo del loop 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 all'utente. 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 ricorsivi involontari, Lambda è in grado di rilevare determinati tipi di cicli ricorsivi poco dopo che si sono verificati. Quando Lambda rileva un ciclo ricorsivo, interrompe l'invocazione della funzione e invia una notifica.

Se il tuo progetto utilizza intenzionalmente modelli ricorsivi, puoi richiedere di disattivare il rilevamento dei cicli ricorsivi in Lambda. Per richiedere questa modifica, contatta AWS Support.

> **Important**
>
> Se il tuo progetto utilizza intenzionalmente una funzione Lambda per riscrivere i dati sulla AWS stessa risorsa che richiama la funzione, fai attenzione e implementa protezioni adeguate per evitare che vengano fatturati addebiti imprevisti. Account AWS Per ulteriori informazioni sulle best practice per l'utilizzo di modelli di invocazione ricorsivi, consulta la pagina Schemi ricorsivi che causano loop indeterminati delle funzioni Lambdain Serverless Land.

Sections

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 AWS SDK, aggiorna questi metadati. I metadati aggiornati includono il numero di volte in cui l'evento ha richiamato la funzione.

> **Note**
>
> Non è necessario abilitare il tracciamento attivo X-Ray per far funzionare questa funzionalità. 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 ti avvisa. Ad esempio, se la funzione è configurata con più trigger, le invocazioni da altri trigger non sono interessate.

> **Note**
>
> Quando l'impostazione `maxReceiveCount` sulla policy di reindirizzamento della coda di origine è superiore a 16, la protezione da ricorsione 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. Ciò incrementa il `receiveCount` valore del 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 di lettere non scritte configurata.

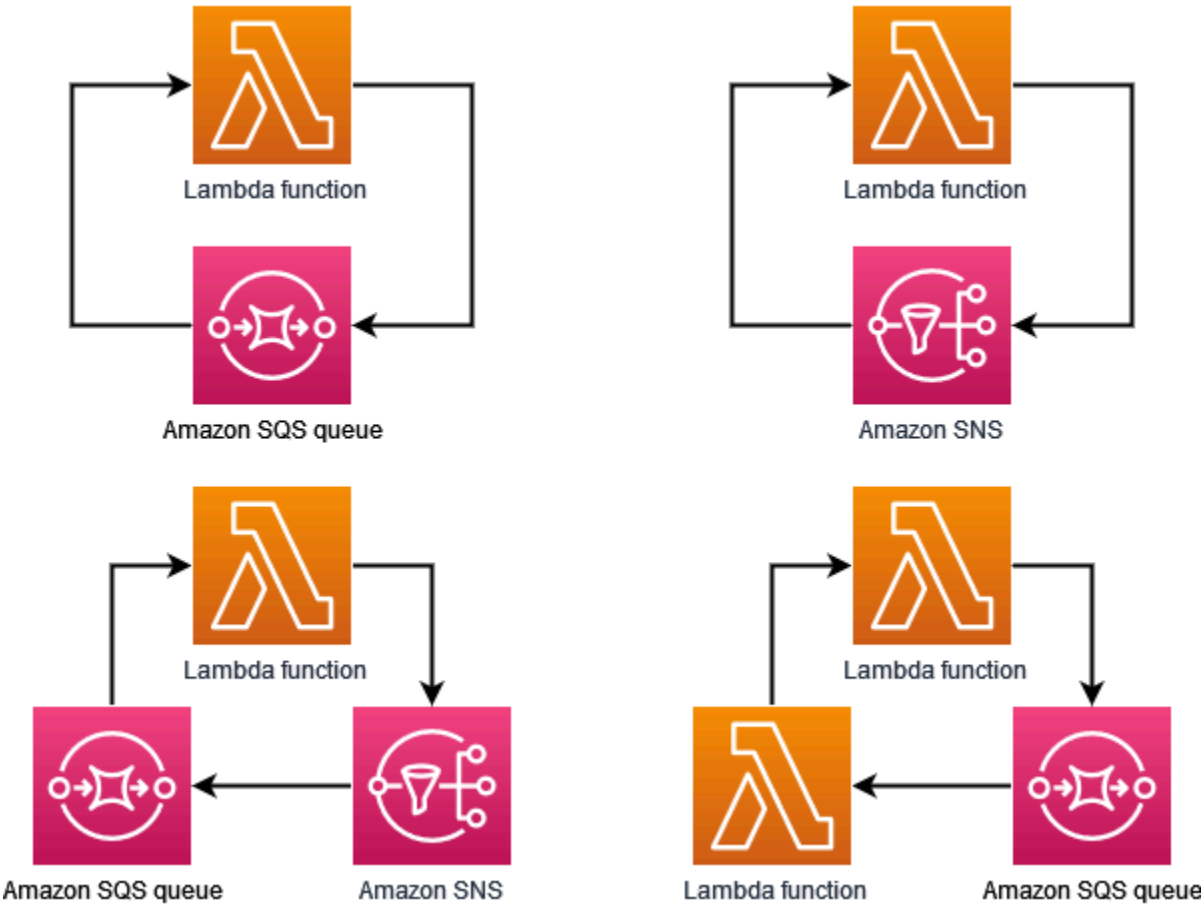
Se hai configurato una destinazione in caso di errore o una coda DLQ per la funzione, Lambda invia anche l'evento dall'invocazione interrotta alla destinazione o alla coda DLQ. Se configuri una destinazione o una coda DLQ per la funzione, assicurati di non utilizzare un argomento Amazon SNS o una coda Amazon SQS che la tua funzione utilizza anche come trigger di eventi o strumento di mappatura dell'origine degli eventi. Se invii eventi alla stessa risorsa che richiama la tua funzione, puoi creare un altro ciclo ricorsivo.

Supportati e SDK Servizi AWS

Lambda è in grado di rilevare solo i loop ricorsivi che includono determinati loop supportati. Servizi AWS Affinché i loop ricorsivi vengano rilevati, la funzione deve utilizzare anche uno degli SDK supportati. AWS

Supportato Servizi AWS

Lambda attualmente rileva i cicli ricorsivi tra le tue funzioni, Amazon SQS e Amazon SNS. Lambda rileva anche i cicli ricorsivi composti solo da funzioni Lambda, che possono richiamarsi reciprocamente 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 tipo di Amazon DynamoDB o Amazon Simple Storage Service (Amazon S3) fa parte del ciclo, Lambda al momento non è in grado di rilevarlo e fermarlo.

Poiché attualmente Lambda rileva solo loop ricorsivi che coinvolgono Amazon SQS e Amazon SNS, è ancora possibile che i loop che coinvolgono Servizi AWS altri possano comportare un utilizzo non intenzionale delle funzioni Lambda.

Per evitare che ti vengano addebitati addebiti imprevisti Account AWS, ti consigliamo di configurare gli CloudWatchallarmi Amazon per avvisarti di modelli di utilizzo insoliti. Ad esempio, puoi CloudWatch configurare la notifica dei picchi nella concorrenza o nelle chiamate della funzione 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.

SDK supportati AWS

Affinché Lambda possa rilevare i cicli ricorsivi, la funzione deve utilizzare un SDK delle versioni seguenti o successive:

Runtime	Versione AWS SDK minima richiesta
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	1.12.200 (SDK versione 1) 2.17.135 (SDK versione 2)
Java 17	2.20.81
Java 21	2,21,24
.NET	3,7,293,0
Ruby	3,134,0
PHP	3,232,0

Alcuni runtime Lambda come Python e Node.js includono una versione dell'SDK. 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.

Il rilevamento della ricorsione Lambda non è supportato per il runtime Lambda di Go.

Notifiche dei cicli ricorsivi

Quando Lambda interrompe un ciclo ricorsivo, ricevi notifiche tramite AWS Health Dashboard e tramite e-mail. Puoi anche utilizzare le CloudWatch metriche per monitorare il numero di chiamate ricorsive interrotte da Lambda.

AWS Health Dashboard notifiche

Quando Lambda interrompe una chiamata ricorsiva, AWS Health Dashboard visualizza una notifica nella pagina Lo stato del tuo account, in Problemi aperti e recenti. Tieni presente che possono essere necessarie fino a tre ore dopo che Lambda interrompe un'invocazione ricorsiva prima che venga visualizzata questa notifica. Per ulteriori informazioni sulla visualizzazione degli eventi dell'account in AWS Health Dashboard, consulta Getting started with your AWS Health Dashboard — Lo stato del tuo account nella AWS Health User Guide.

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 tre ore dopo che Lambda interrompe un'invocazione ricorsiva prima che tu riceva questo avviso via e-mail.

Lambda invia avvisi e-mail a ciclo ricorsivo al contatto principale Account AWS dell'account e al contatto operativo alternativo 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 .

CloudWatch Metriche Amazon

La CloudWatch metrica `RecursiveInvocationsDropped` registra il numero di chiamate di funzioni che Lambda ha interrotto perché la funzione è stata richiamata più di 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 relative alla 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 riscrivono gli eventi nella 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.

Disabilitazione di una mappatura dell'origine degli eventi per la funzione (AWS CLI)

1. Per trovare l'UUID per la mappatura delle sorgenti degli eventi che desideri disabilitare, esegui il comando list-event-source-mappings AWS Command Line Interface .AWS CLI

```
aws lambda list-event-source-mappings
```

2. Per disabilitare la mappatura delle sorgenti degli eventi, esegui il seguente comando update-event-source-mapping. AWS CLI

```
aws lambda update-event-source-mapping --function-name MyFunction \
--uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 --no-enabled
```

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. 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 nelle seguenti regioni: Asia Pacifico (Hyderabad) (`ap-south-2`), Asia Pacifico (Melbourne) (`ap-southeast-4`), Canada occidentale (Calgary) (), Europa (Spagna`ca-west-1`) (), Europa (Zurigo`eu-south-2`) (), Israele (Tel Aviv`eu-central-2`) () e Medio Oriente (Emirati Arabi Uniti`il-central-1`) (). `me-central-1`

Gli URL della funzione sono abilitati alla rete dual stack e supportano IPv4 e IPv6. Dopo aver configurato un URL della funzione per la funzione utilizzata, è possibile richiamare la funzione attraverso il relativo endpoint HTTP(S) tramite un browser Web, curl, Postman o un 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.

Argomenti

- Creazione e gestione degli URL della funzione Lambda
- Controlla l'accesso agli URL delle funzioni Lambda
- Richiamo di URL di funzioni Lambda
- Monitoraggio degli URL della funzione Lambda
- Tutorial: Creazione di una funzione Lambda con un URL della funzione

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. 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 nelle seguenti regioni: Asia Pacifico (Hyderabad) (`ap-south-2`), Asia Pacifico (Melbourne) (`ap-southeast-4`), Canada occidentale (Calgary) (`ca-west-1`), Europa (Spagna) (`eu-south-2`), Europa (Zurigo) (`eu-central-2`), Israele (Tel Aviv) (`il-central-1`) e Medio Oriente (Emirati Arabi Uniti) (`me-central-1`).

La sezione seguente mostra come creare e gestire l'URL di una funzione utilizzando la console Lambda e il AWS CLI modello AWS CloudFormation

Argomenti

Creazione di un URL della funzione (console)

Segui queste fasi per creare un URL della funzione usando la console.

Creazione di un URL della funzione per una funzione esistente (console)

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 Resource Sharing (CORS).
7. Selezionare 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 un URL della funzione per un alias esistente (console)

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 Resource Sharing (CORS).
8. Selezionare 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.

Creazione di una nuova funzione con un URL della funzione (console)

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 il runtime del linguaggio desiderato, ad esempio Node.js 18.x.
   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 Resource Sharing (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 di un URL di funzione (AWS CLI)

Per creare un URL di funzione per una funzione Lambda esistente utilizzando AWS Command Line Interface (AWS CLI), esegui il comando seguente:

```
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 il riferimento CreateFunctionUrlConfigall'API.

> **Note**
>
> Per creare l'URL di una funzione tramite 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 AWS 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
```

Parametri

- (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 `AWS::Lambda::Url` risorsa in CloudFormation, utilizzate la seguente sintassi.

  Example AWS::Lambda::Url.Cors (JSON)

  ```
  {
    "AllowCredentials" : Boolean,
    "AllowHeaders" : [ String, ... ],
    "AllowMethods" : [ String, ... ],
    "AllowOrigins" : [ String, ... ],
    "ExposeHeaders" : [ String, ... ],
    "MaxAge" : Integer
  }
  ```

  Example AWS::Lambda::Url.Cors (YAML)

  ```
    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 Resource Sharing (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

Access-Control-Allow-Origin	`AllowOrigins`	`*` (consente tutte le origini)
		`https://www.example.com`
		`http://localhost:60905`

Access-Control-Allow-Methods	`AllowMethods`	`GET`, `POST`, `DELETE`, `*`
Access-Control-Allow-Headers	`AllowHeaders`	`Date`, `Keep-Alive` , `X-Custom-Header`
Access-Control-Expose-Headers	`ExposeHeaders`	`Date`, `Keep-Alive` , `X-Custom-Header`
Access-Control-Allow-Credentials	`AllowCredentials`	`TRUE`
Access-Control-Max-Age	`MaxAge`	`5` (default), `300`

Quando configuri 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, le intestazioni CORS configurate sull'URL della funzione hanno la precedenza.

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 concorrenza 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 determinerà 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 elimini una funzione con un URL di funzione, Lambda elimina in modo asincrono l'URL della funzione. Se crei 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.

Controlla l'accesso agli URL delle funzioni Lambda

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 policy di identità del principale IAM e alla policy basata sulle risorse della funzione. Scegli questa opzione se desideri che solo utenti e ruoli autenticati richiamino la funzione tramite 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.

Oltre a `AuthType`, puoi anche utilizzare policy basate sulle risorse per concedere autorizzazioni ad altri Account AWS per richiamare la funzione. Per ulteriori informazioni, consulta Utilizzo di politiche basate sulle risorse in Lambda.

Per ulteriori approfondimenti sulla sicurezza, puoi utilizzare per ottenere un'analisi completa dell'accesso esterno AWS Identity and Access Management Access Analyzer 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. L'uso di IAM Access Analyzer è gratuito per qualsiasi AWS cliente. Per iniziare a usare IAM Access Analyzer, consulta Using AWS IAM Access Analyzer.

Questa pagina contiene esempi di politiche basate sulle risorse per entrambi i tipi di autenticazione e anche come creare queste politiche utilizzando l'operazione AddPermissionAPI o la console Lambda. Per informazioni su come richiamare l'URL della funzione dopo aver configurato 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 l'autorizzazione `lambda:InvokeFunctionUrl`. A seconda di chi effettua la richiesta di richiamo, potrebbe essere necessario concedere questa autorizzazione utilizzando una policy basata sulle risorse.

Se il principale che effettua la richiesta corrisponde all'URL della funzione, il principale deve disporre delle `lambda:InvokeFunctionUrl` autorizzazioni nella propria politica basata sull'identità o avere le autorizzazioni concesse nella politica basata sulle risorse della funzione. Account AWS In altre parole, una policy basata sulle risorse è facoltativa se l'utente ha già autorizzazioni `lambda:InvokeFunctionUrl` nella policy basata sull'identità. La valutazione delle policy segue le regole delineate nella sezione Determinare se una richiesta è consentita o rifiutata in un account.

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`, sia autorizzazioni concesse in una policy basata sulle risorse nella funzione che si sta tentando di richiamare. Nei casi di interazione tra account, la valutazione delle policy segue le regole delineate in Determinare se una richiesta tra account è consentita.

Per un esempio di interazione tra account, la seguente politica basata sulle risorse consente al ruolo di richiamare l'URL della funzione associata alla funzione: `example` Account AWS `444455556666` `my-function`

Example Policy di richiamo tra account dell'URL della funzione

```
{
    "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"
                }
            }
        }
    ]
}
```

È possibile creare questa dichiarazione di policy tramite la console seguendo questi 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 (Policy basata sulle risorse), scegli Add permissions (Aggiungi autorizzazioni).
5. Scegli Function URL (URL funzione).
6. Per Auth type (Tipo di autenticazione), scegli AWS_IAM.
7. (Opzionale) Per Statement ID (ID dichiarazione), inserisci un ID dichiarazione per la dichiarazione di policy.
8. Per Principale, inserisci l'ID o il nome della risorsa Amazon (ARN) dell'utente o del ruolo a cui desideri concedere le autorizzazioni. Ad esempio: **444455556666**.
9. Selezionare Salva.

In alternativa, è possibile creare questa dichiarazione di policy utilizzando il seguente comando add-permission (): AWS Command Line Interface AWS CLI

```
aws lambda add-permission --function-name my-function \
    --statement-id example0-cross-account-statement \
    --action lambda:InvokeFunctionUrl \
    --principal 444455556666 \
    --function-url-auth-type AWS_IAM
```

Nell'esempio precedente, il valore della chiave di condizione `lambda:FunctionUrlAuthType` è `AWS_IAM`. Questa policy consente l'accesso solo quando anche il tipo di autenticazione dell'URL della funzione è `AWS_IAM`.

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 richiamare la funzione.

In alcuni casi, potrebbe essere preferibile che l'URL della 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, gli utenti devono comunque avere autorizzazioni `lambda:InvokeFunctionUrl` per richiamare correttamente l'URL della funzione. Puoi concedere le autorizzazioni `lambda:InvokeFunctionUrl` utilizzando la seguente policy basata sulle risorse:

Example Policy di richiamo dell'URL della funzione per tutte le entità principali non autenticate

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "lambda:InvokeFunctionUrl",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
            "Condition": {
                "StringEquals": {
                    "lambda:FunctionUrlAuthType": "NONE"
                }
            }
        }
    ]
}
```

> **Note**
>
> Quando crei un URL di funzione con tipo di autenticazione `NONE` tramite la console o AWS Serverless Application Model (AWS SAM), Lambda crea automaticamente la precedente dichiarazione politica basata sulle risorse per te. Se la policy esiste già o l'utente o il ruolo che crea l'applicazione non dispone delle autorizzazioni appropriate, Lambda non la crea per tuo conto. Se utilizzi direttamente l'API Lambda AWS CLI AWS CloudFormation, o l'API Lambda, devi aggiungere tu stesso le `lambda:InvokeFunctionUrl` autorizzazioni. Questo rende pubblica la funzione.
>
> Inoltre, 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.

In questa dichiarazione, il valore della chiave di condizione `lambda:FunctionUrlAuthType` è `NONE`. Questa dichiarazione di policy consente l'accesso solo quando anche il tipo di autenticazione dell'URL della funzione è `NONE`.

Se la policy basata sulle risorse di una funzione non concede le autorizzazioni `lambda:invokeFunctionUrl`, gli utenti riceveranno un codice di errore 403 Non consentito quando tentano di richiamare l'URL della funzione, 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.

> **Note**
>
> La console Lambda supporta l'aggiunta di autorizzazioni solo per `lambda:InvokeFunctionUrl`. Per tutte le altre azioni, è necessario aggiungere autorizzazioni utilizzando l'API Lambda o AWS CLI.

Per consentire o negare l'accesso all'URL della funzione ad altre AWS entità, includi queste azioni nelle policy IAM. Ad esempio, la seguente politica 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

```
{
    "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 accurato degli accessi sugli URL della funzione, utilizzare una chiave di condizione. Lambda supporta una chiave di condizione aggiuntiva per gli URL della funzione: `FunctionUrlAuthType`. La chiave `FunctionUrlAuthType` definisce un valore enum che descrive il tipo di autenticazione utilizzato dall'URL della funzione. Il valore può essere `AWS_IAM` o `NONE`.

È possibile utilizzare questa chiave di condizione in policy associate alla 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

```
{
    "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 `CreateFunctionUrlConfig` e nelle `UpdateFunctionUrlConfig` richieste sulle funzioni con tipo di autenticazione URL`AWS_IAM`, puoi definire la seguente politica:

Example Policy dell'URL della funzione con permesso esplicito

```
{
    "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

```
{
    "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. 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 nelle seguenti regioni: Asia Pacifico (Hyderabad) (`ap-south-2`), Asia Pacifico (Melbourne) (`ap-southeast-4`), Canada occidentale (Calgary) (), Europa (Spagna`ca-west-1`) (), Europa (Zurigo`eu-south-2`) (), Israele (Tel Aviv`eu-central-2`) () e Medio Oriente (Emirati Arabi Uniti`il-central-1`) (). `me-central-1`

Gli URL della funzione sono abilitati alla rete dual stack e supportano IPv4 e IPv6. Dopo aver configurato l'URL della funzione, è possibile richiamare la funzione attraverso il relativo endpoint HTTP(S) tramite un browser Web, curl, Postman o un client HTTP. Per richiamare un URL della funzione, è necessario disporre di autorizzazioni `lambda:InvokeFunctionUrl`. 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 awscurl, Postman e AWS SigV4 Proxy 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 le istruzioni su come firmare manualmente le richieste con SigV4, consulta Firma delle richieste AWS con Signature Version 4 nella Guida di riferimento generale di 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	Descrizione	Esempio
`version`	Il tipo di formato del payload per questo evento. Gli URL della funzione Lambda attualmente supportano il tipo di formato del payload versione 2.0.	`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 supportati includono `a-z`, `A-Z`, `0-9`, `.`, `_`, `-`, `%`, `&`, `=` e `+`.	`"?parameter1=value1&parameter2=value2"`
`cookies`	Un array contenente tutti i cookie inviati come parte della richiesta.	`["Cookie_1=Value_1", "Cookie_2=Value_2"]`

Parametro	Descrizione	Esempio
`headers`	L'elenco delle intestazioni della richiesta, presentate come coppie chiave-valore.	`{"header1": "value1", "header2": "value2"}`
`queryStringParameters`	I parametri di query per la richiesta. Ad esempio, se l'URL della richiesta è `https://{url-id}.lambda-url.{region}.on.aws/example?name=Jane` , il valore `queryStringParameters` è un oggetto JSON con una chiave di `name` e un valore di `Jane`.	`{"name": "Jane"}`
`requestContext`	Un oggetto contenente informazioni aggiuntive sulla richiesta, ad esempio `requestId` , l'ora della richiesta e l'identità del chiamante se autorizzato tramite AWS Identity and Access Management (IAM).	
`requestContext.accountId`	L'ID Account AWS del proprietario della funzione.	`"123456789012"`
`requestContext.apiId`	L'ID dell'URL della funzione.	`"33anwqw8fj"`
`requestContext.authentication`	Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su `null`.	`null`

Parametro	Descrizione	Esempio
`requestContext.authorizer`	Un oggetto contenente informazioni sull'identità del chiamante, se l'URL della funzione utilizza il tipo di autenticazione `AWS_IAM`. Altrimenti, Lambda lo imposta su `null`.	
`requestContext.authorizer.iam.accessKey`	La chiave di accesso dell'identità del chiamante.	`"AKIAIOSFODNN7EXAMPLE"`
`requestContext.authorizer.iam.accountId`	L'ID Account AWS dell'identità del chiamante.	`"111122223333"`
`requestContext.authorizer.iam.callerId`	L'ID (ID utente) del chiamante.	`"AIDACKCEVSQ6C2EXAMPLE"`
`requestContext.authorizer.iam.cognitoIdentity`	Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su `null` o lo esclude dal JSON.	`null`
`requestContext.authorizer.iam.principalOrgId`	L'ID dell'organizzazione principale associato all'identità del chiamante.	`"AIDACKCEVSQORGEXAMPLE"`
`requestContext.authorizer.iam.userArn`	L'Amazon Resource Name (ARN) utente dell'identità del chiamante.	`"arn:aws:iam::111122223333:user/example-user"`
`requestContext.authorizer.iam.userId`	L'ID utente dell'identità del chiamante.	`"AIDACOSFODNN7EXAMPLE2"`

Parametro	Descrizione	Esempio
`requestContext.domainName`	Il nome di dominio dell'URL della funzione.	`"<url-id>.lambda-url.us-west-2.on.aws"`
`requestContext.domainPrefix`	Il prefisso di dominio dell'URL della funzione.	`"<url-id>"`
`requestContext.http`	Un oggetto contenente i dettagli sulla richiesta HTTP.	
`requestContext.http.method`	Il metodo HTTP utilizzato nella richiesta. I valori validi includono `GET`, `POST`, `PUT`, `HEAD`, `OPTIONS`, `PATCH` e `DELETE`.	`GET`
`requestContext.http.path`	Percorso della richiesta. Ad esempio, se l'URL della richiesta è `https://{url-id}.lambda-url.{region}.on.aws/example/test/demo`, il valore del percorso è `/example/test/demo`.	`/example/test/demo`
`requestContext.http.protocol`	Il protocollo della richiesta.	`HTTP/1.1`
`requestContext.http.sourceIp`	L'indirizzo IP di origine della connessione TCP immediata da cui proviene la richiesta.	`123.123.123.123`
`requestContext.http.userAgent`	Il valore dell'intestazione della richiesta User-Agent.	`Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) Gecko/20100101 Firefox/42.0`

Parametro	Descrizione	Esempio
`requestContext.requestId`	L'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`
`requestContext.routeKey`	Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su `$default` come segnaposto.	`$default`
`requestContext.stage`	Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su `$default` come segnaposto.	`$default`
`requestContext.time`	Il timestamp della richiesta.	`"07/Sep/2021:22:50:22 +0000"`
`requestContext.timeEpoch`	Il timestamp della richiesta, in formato temporale Unix.	`"1631055022677"`
body	Il corpo della richiesta. Se il tipo di contenuto della richiesta è binario, il corpo è con codifica base64.	`{"key1": "value1", "key2": "value2"}`
`pathParameters`	Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su `null` o lo esclude dal JSON.	`null`
`isBase64Encoded`	`TRUE` se il corpo è un payload binario e con codifica base64. `FALSE` in caso contrario.	`FALSE`

Parametro	Descrizione	Esempio
`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`.
- `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!"`	`{ "statusCode": 200, "body": "Hello, world!", "headers": { "content-type": "application/json" }, "isBase64Encoded": false }`	`HTTP/2 200`
		`date: Wed, 08 Sep 2021 18:02:24 GMT`
		`content-type: application/json`
		`content-length: 15`
		
		`"Hello, world!"`

Esempio di output per una risposta JSON

Risultato della funzione Lambda	Output di risposta interpretata	Risposta HTTP (cosa vede il client)

`{ "message": "Hello, world!" }`	`{ "statusCode": 200, "body": { "message": "Hello, world!" }, "headers": { "content-type": "application/json" }, "isBase64Encoded": false }`	`HTTP/2 200`
		`date: Wed, 08 Sep 2021 18:02:24 GMT`
		`content-type: application/json`
		`content-length: 34`
		
		`{ "message": "Hello, world!" }`

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.

Esempio di output per una risposta che restituisce cookie

Risultato della funzione Lambda	Risposta HTTP (cosa vede il client)

<pre>{
   "statusCode": 201,
    "headers": {
        "Content-Type": "application/
json",
        "My-Custom-Header": "Custom
 Value"
    },
    "body": JSON.stringify({</pre>
	<pre>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</pre>

Risultato della funzione Lambda	Risposta HTTP (cosa vede il client)

<pre>"message": "Hello, world!"
}),
"cookies": [
"Cookie_1=Value1; Expires=21 Oct 2021 07:48 GMT",
"Cookie_2=Value2; Max-Age=78000"
],
"isBase64Encoded": false
}</pre>
	<pre>{
	"message": "Hello, world!"
	}</pre>

Monitoraggio degli URL della funzione Lambda

Puoi utilizzare AWS CloudTrail Amazon CloudWatch per monitorare gli URL delle tue funzioni.

Argomenti

URL delle funzioni di monitoraggio con CloudTrail

Per gli URL delle funzioni, Lambda supporta automaticamente la registrazione delle seguenti operazioni API come eventi CloudTrail nei file di registro:

- 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 CloudTrail degli eventi. Per conservare i record degli ultimi 90 giorni, puoi creare un percorso.

Per impostazione predefinita, CloudTrail non registra `InvokeFunctionUrl` le richieste, che sono considerate eventi relativi ai dati. Tuttavia, puoi attivare la registrazione degli eventi relativi ai dati. CloudTrail Per ulteriori informazioni, consulta Registrazione di eventi di dati per i percorsi nella Guida per l'utente di AWS CloudTrail .

CloudWatch metriche per gli URL delle funzioni

Lambda invia metriche aggregate sulle richieste URL delle funzioni a. CloudWatch Con queste metriche, puoi monitorare gli URL delle tue funzioni, creare dashboard e configurare gli allarmi nella console. 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.

Tutorial: Creazione di una funzione Lambda con un URL della funzione

In questo tutorial viene creata una funzione Lambda definita come archivio di file .zip con un endpoint URL della funzione pubblico che restituisce il prodotto di due numeri. Per ulteriori informazioni sulla configurazione degli URL della funzione, consulta Creazione e gestione degli URL della funzione.

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 l'AWS Command Line Interface (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. Apri la pagina Ruoli della console AWS Identity and Access Management (IAM).
2. Scegli Crea ruolo.
3. Per il tipo di entità affidabile seleziona AWS servizio, quindi per Use case, seleziona Lambda.
4. Seleziona Successivo.
5. Nel riquadro Politiche di autorizzazione, inserisci **`AWSLambdaBasicExecutionRole`** nella casella di ricerca.
6. Seleziona la casella di controllo accanto alla politica **`AWSLambdaBasicExecutionRole`** AWS gestita, quindi scegli Avanti.
7. Inserisci **`lambda-url-role`** il nome del ruolo, quindi scegli Crea ruolo.

La `AWSLambdaBasicExecutionRole` policy dispone delle autorizzazioni necessarie alla funzione per scrivere log su Amazon CloudWatch Logs. Più avanti nel tutorial, avrai bisogno dell'Amazon Resource Name (ARN) del ruolo per creare la tua funzione Lambda.

Per trovare l'ARN del tuo ruolo esecutivo

1. Apri la pagina Ruoli della console AWS Identity and Access Management (IAM).
2. Seleziona il ruolo che hai appena creato (`lambda-url-role`).
3. Nel riquadro Riepilogo, copia l'ARN.

Crea una funzione Lambda con un URL della funzione (archivio di file.zip)

Crea una funzione Lambda con un endpoint URL della funzione utilizzando un archivio di file .zip.

Creazione della funzione

1. Copiare il codice di esempio seguente in un file denominato `index.js`.

   Example index.js

```
exports.handler = async (event) => {
    let body = JSON.parse(event.body);
    const product = body.num1 * body.num2;
    const response = {
        statusCode: 200,
```

```
        body: "The product of " + body.num1 + " and " + body.num2 + " is " +
 product,
    };
    return response;
};
```

2. Crea un pacchetto di implementazione.

```
zip function.zip index.js
```

3. Creare una funzione Lambda con il comando `create-function`. Assicurati di sostituire il ruolo ARN con l'ARN del tuo ruolo di esecuzione che hai copiato in precedenza nel tutorial.

```
aws lambda create-function \
    --function-name my-url-function \
    --runtime nodejs18.x \
    --zip-file fileb://function.zip \
    --handler index.handler \
    --role arn:aws:iam::123456789012:role/lambda-url-role
```

4. Aggiungi una policy basata sulle risorse alla funzione che concede le autorizzazioni per consentire l'accesso pubblico all'URL della funzione.

```
aws lambda add-permission \
    --function-name my-url-function \
    --action lambda:InvokeFunctionUrl \
    --principal "*" \
    --function-url-auth-type "NONE" \
    --statement-id url
```

5. Crea un endpoint URL per la funzione con il comando `create-function-url-config`.

```
aws lambda create-function-url-config \
    --function-name my-url-function \
    --auth-type NONE
```

Verifica l'endpoint URL della funzione

Richiama la funzione Lambda chiamando l'endpoint URL della funzione utilizzando un client HTTP, ad esempio curl o Postman.

```
curl 'https://abcdefg.lambda-url.us-east-1.on.aws/' \
-H 'Content-Type: application/json' \
-d '{"num1": "10", "num2": "10"}'
```

Verrà visualizzato l'output seguente:

```
The product of 10 and 10 is 100
```

Crea una funzione Lambda con una funzione URL () CloudFormation

Puoi anche creare una funzione Lambda con un endpoint URL della funzione utilizzando il tipo. AWS CloudFormation `AWS::Lambda::Url`

```
Resources:
  MyUrlFunction:
    Type: AWS::Lambda::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs18.x
      Role: arn:aws:iam::123456789012:role/lambda-url-role
      Code:
        ZipFile: |
          exports.handler = async (event) => {
              let body = JSON.parse(event.body);
              const product = body.num1 * body.num2;
              const response = {
                  statusCode: 200,
                  body: "The product of " + body.num1 + " and " + body.num2 + " is " +
 product,
              };
              return response;
          };
      Description: Create a function with a URL.
  MyUrlFunctionPermissions:
    Type: AWS::Lambda::Permission
    Properties:
      FunctionName: !Ref MyUrlFunction
      Action: lambda:InvokeFunctionUrl
      Principal: "*"
      FunctionUrlAuthType: NONE
  MyFunctionUrl:
    Type: AWS::Lambda::Url
```

```
    Properties:
      TargetFunctionArn: !Ref MyUrlFunction
      AuthType: NONE
```

Crea una funzione Lambda con un URL della funzione (AWS SAM)

Puoi anche creare una funzione Lambda configurata con un URL di funzione utilizzando AWS Serverless Application Model ()AWS SAM.

```
ProductFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: function/.
    Handler: index.handler
    Runtime: nodejs18.x
    AutoPublishAlias: live
    FunctionUrlConfig:
      AuthType: NONE
```

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 ai tuoi. 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 **`delete`** nel campo di immissione testo e scegli Delete (Elimina).

Gestione delle AWS Lambda funzioni

Scopri come regolare e proteggere le risorse associate alla tua funzione Lambda utilizzando l'API o la console Lambda.

Usare Lambda con AWS CLI

È possibile AWS Command Line Interface utilizzarlo per gestire funzioni e altre AWS Lambda risorse. Lo AWS CLI utilizza AWS SDK for Python (Boto) per interagire con l'API Lambda. In questo tutorial, è possibile gestire e invocare le funzioni Lambda con AWS CLI.

Dimensionamento della funzione

È possibile configurare due controlli di simultaneità a livello di funzione: la simultaneità riservata e la simultaneità fornita. La simultaneità è il numero di istanze delle funzioni attive e può essere configurata per garantire che le funzioni critiche evitino la limitazione (della larghezza di banda della rete).

Firma del codice

La firma del codice per Lambda fornisce controlli di attendibilità e integrità che consentono di verificare che nelle funzioni Lambda sia implementato solo il codice inalterato pubblicato dagli sviluppatori approvati.

Organizzazione mediante i tag

Puoi aggiungere un tag alle funzioni Lambda per attivare il controllo degli accessi basato su attributi (ABAC) e organizzarle in base al proprietario, al progetto o al reparto.

Uso dei livelli

È possibile applicare i livelli creati in precedenza per ridurre le dimensioni del pacchetto di implementazione e per promuovere la condivisione del codice e la separazione delle responsabilità in modo da poter iterare più velocemente la scrittura della logica di business.

Utilizzo di Lambda con AWS CLI

È possibile utilizzare AWS Command Line Interface per gestire le funzioni e altre risorse AWS Lambda. 'AWS CLI usa AWS SDK for Python (Boto) per l'interazione con le API Lambda. È possibile utilizzarla per ulteriori informazioni sull'API e applicare tali conoscenze nella creazione di applicazioni che utilizzano Lambda con SDK AWS.

In questo tutorial, è possibile gestire e invocare le funzioni Lambda con AWS CLI. Per ulteriori informazioni, consulta Cos'è AWS CLI? nella Guida per l'utente di AWS Command Line Interface.

Prerequisiti

Questo tutorial presuppone una certa conoscenza delle operazioni di base di Lambda e della console relativa. Se non lo hai già fatto, segui le istruzioni in the section called “Creare una funzione Lambda con la console”.

Per completare i passaggi seguenti, è necessaria l'AWS Command Line Interface (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 del ruolo di esecuzione

Creare il ruolo di esecuzione che offre l'autorizzazione della funzione per accedere alle risorse AWS. Per creare un ruolo di esecuzione con AWS CLI, utilizzare il comando `create-role`.

Nell'esempio seguente occorre specificare la policy di affidabilità in riga. I requisiti per l'escape delle virgolette nella stringa JSON variano in base alla 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"}]}'
```

È inoltre possibile definire la policy di affidabilità per il ruolo utilizzando un file JSON. Nell'esempio seguente, `trust-policy.json` è un file che si trova nella directory attuale. Questa policy di attendibilità consente a Lambda di utilizzare le autorizzazioni del ruolo assegnando al principal del servizio l'autorizzazione `lambda.amazonaws.com` per chiamare l'operazione AWS Security Token Service (AWS STS) `AssumeRole`.

Example trust-policy.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
```

Verrà visualizzato l'output seguente:

```
{
```

```
    "Role": {
        "Path": "/",
        "RoleName": "lambda-ex",
        "RoleId": "AROAQFOXMPL6TZ6ITKWND",
        "Arn": "arn:aws:iam::123456789012:role/lambda-ex",
        "CreateDate": "2020-01-17T23:19:12Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lambda.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

Utilizza il comando attach-policy-to-role per aggiungere le autorizzazioni al ruolo. Iniziare aggiungendo la policy gestita `AWSLambdaBasicExecutionRole`.

```
aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole
```

La AWSLambdaBasicExecutionRolepolicy dispone delle autorizzazioni necessarie alla funzione per scrivere i log in Logs. CloudWatch

Creazione della funzione

L'esempio seguente registra i valori delle variabili di ambiente e l'oggetto evento.

Example index.js

```
exports.handler = async function(event, context) {
  console.log("ENVIRONMENT VARIABLES\n" + JSON.stringify(process.env, null, 2))
  console.log("EVENT\n" + JSON.stringify(event, null, 2))
  return context.logStreamName
}
```

Per creare la funzione

1. Copiare il codice di esempio in un file denominato `index.js`.
2. Crea un pacchetto di implementazione.

```
zip function.zip index.js
```

3. Creare una funzione Lambda con il comando `create-function`. Sostituire il testo evidenziato nel ruolo ARN con l'ID dell'account.

```
aws lambda create-function --function-name my-function \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs20.x \
--role arn:aws:iam::123456789012:role/lambda-ex
```

Verrà visualizzato l'output seguente:

```
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "Runtime": "nodejs20.x",
    "Role": "arn:aws:iam::123456789012:role/lambda-ex",
    "Handler": "index.handler",
    "CodeSha256": "FpFMvUhayLkOoVBpNuNiIVML/tuGv2iJQ7t0yWVTU8c=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "88ebe1e1-bfdf-4dc3-84de-3017268fa1ff",
    ...
}
```

Per ottenere i log per una chiamata dalla riga di comando, utilizza l'opzione `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 della chiamata.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
```

```
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Puoi usare l'utilità `base64` per decodificare i log.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text |  base64 -d
```

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'utility `base64` è disponibile su Linux, macOS e Ubuntu su Windows. Per macOS, il comando è `base64 -D`.

Per ottenere eventi di log completi dalla riga di comando, puoi includere il nome del flusso di log nell'output della funzione, come mostrato nell'esempio precedente. Lo script di esempio seguente richiama una funzione denominata `my-function` e scarica gli ultimi 5 eventi di log.

Example Script get-logs.sh

Questo esempio richiede che `my-function` restituisca un ID del flusso di log.

```
#!/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
 $(cat out) --limit 5
```

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`.

```
./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"
}
```

Aggiorna la funzione

Dopo aver creato una funzione, è possibile configurare funzionalità aggiuntive per la funzione, quali trigger, accesso alla rete e accesso al file system. È inoltre possibile modificare le risorse associate alla funzione, ad esempio la memoria e la concorrenza. Queste configurazioni si applicano a funzioni definite come archivi di file .zip e a funzioni definite come immagini di container.

Utilizzate il update-function-configurationcomando per configurare le funzioni. L'esempio seguente imposta la memoria delle funzioni su 256 MB.

Example update-function-configuration comando

```
aws lambda update-function-configuration \
--function-name my-function \
--memory-size 256
```

Elenco delle funzioni Lambda nell'account

Esegui il seguente comando `list-functions` AWS CLI per recuperare un elenco delle funzioni create.

```
aws lambda list-functions --max-items 10
```

Verrà visualizzato l'output seguente:

```
{
    "Functions": [
        {
            "FunctionName": "cli",
            "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-
function",
            "Runtime": "nodejs20.x",
            "Role": "arn:aws:iam::123456789012:role/lambda-ex",
            "Handler": "index.handler",
            ...
        },
        {
```

```
            "FunctionName": "random-error",
            "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:random-
error",
            "Runtime": "nodejs20.x",
            "Role": "arn:aws:iam::123456789012:role/lambda-role",
            "Handler": "index.handler",
            ...
        },
        ...
      ],
      "NextToken": "eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxMH0="
}
```

Nella risposta, Lambda restituisce un elenco con un massimo di 10 funzioni. Se sono presenti altre funzioni da recuperare, `NextToken` fornisce un contrassegno che è possibile utilizzare nella richiesta `list-functions` successiva. Il seguente comando `list-functions` AWS CLI è un esempio che mostra il l'utilizzo del parametro `--starting-token`.

```
aws lambda list-functions --max-items 10 --starting-
token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxMH0=
```

Recupera una funzione Lambda

Il comando `get-function` della CLI Lambda restituisce i metadati della funzione Lambda e un URL prefirmato da utilizzare per scaricare il pacchetto di implementazione della funzione.

```
aws lambda get-function --function-name my-function
```

Verrà visualizzato l'output seguente:

```
{
    "Configuration": {
        "FunctionName": "my-function",
        "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
        "Runtime": "nodejs20.x",
        "Role": "arn:aws:iam::123456789012:role/lambda-ex",
        "CodeSha256": "FpFMvUhayLkOoVBpNuNiIVML/tuGv2iJQ7t0yWVTU8c=",
        "Version": "$LATEST",
        "TracingConfig": {
            "Mode": "PassThrough"
        },
```

```
        "RevisionId": "88ebe1e1-bfdf-4dc3-84de-3017268fa1ff",
        ...
    },
    "Code": {
        "RepositoryType": "S3",
        "Location": "https://awslambda-us-east-2-tasks.s3.us-east-2.amazonaws.com/
snapshots/123456789012/my-function-4203078a-b7c9-4f35-..."
    }
}
```

Per ulteriori informazioni, vedere GetFunction.

Eliminazione

Per eliminare la funzione `delete-function`, eseguire il comando `my-function` riportato di seguito.

```
aws lambda delete-function --function-name my-function
```

Eliminare il ruolo IAM creato nella console IAM. Per informazioni sull'eliminazione di un ruolo, consultare la sezione Eliminazione di ruoli o profili delle istanze nella Guida per l'utente di IAM.

Comprendere il ridimensionamento delle funzioni Lambda

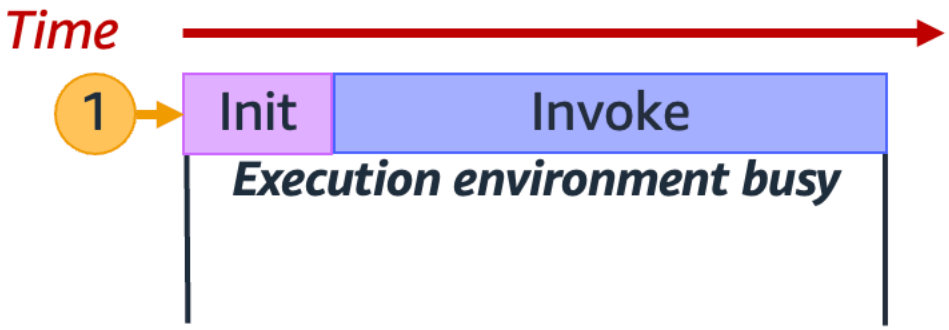
La concorrenza è il numero di richieste in corso che la AWS Lambda funzione gestisce contemporaneamente. Per ogni richiesta simultanea, Lambda fornisce un'istanza separata del tuo ambiente di esecuzione. Man mano che le funzioni ricevono più richieste, Lambda gestisce automaticamente il dimensionamento 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.

Questo argomento spiega i concetti di concorrenza 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

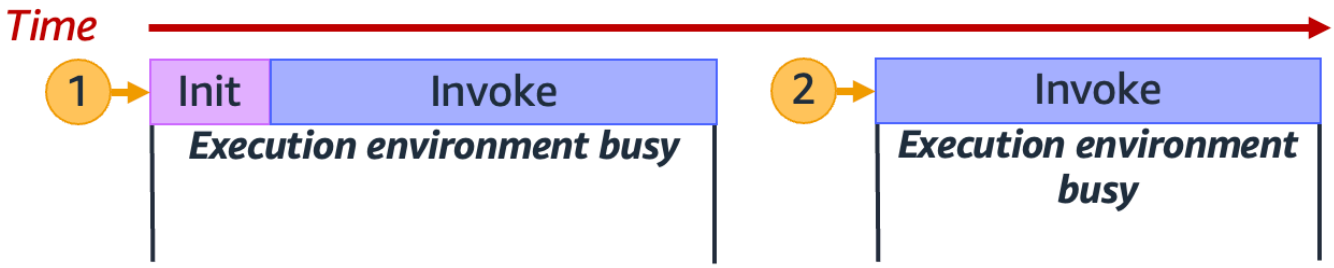
Lambda richiama la funzione in un ambiente di esecuzione sicuro e isolato. Per gestire una richiesta, Lambda deve prima inizializzare un ambiente di esecuzione (la fase Init) e poi utilizzare tale ambiente per richiamare la funzione (la fase Invoke):



> **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.

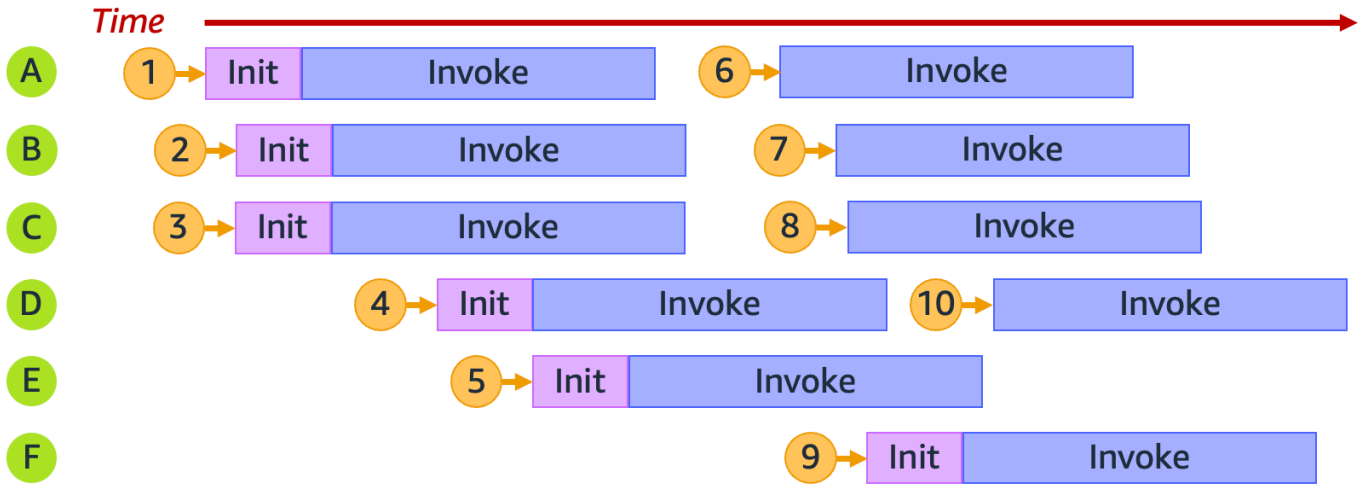


Nel diagramma precedente, Lambda riutilizza l'ambiente di esecuzione per gestire la seconda richiesta (rappresentata dal cerchio giallo con etichetta 2).

Finora ci siamo concentrati su una sola istanza dell'ambiente di esecuzione (ossia una simultaneità pari a 1). In pratica, Lambda potrebbe dover fornire più istanze dell'ambiente di esecuzione in parallelo in modo da gestire tutte le richieste in entrata. 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:



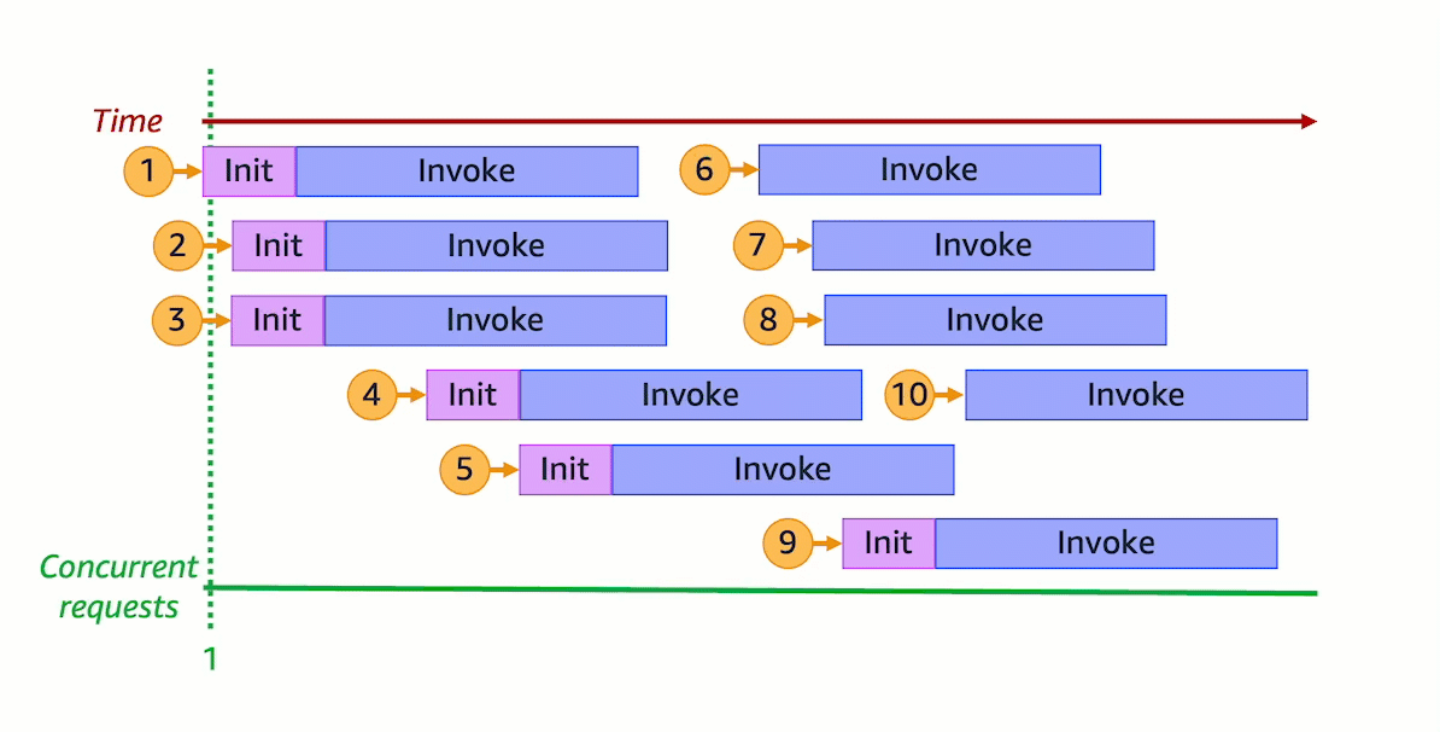
Nel diagramma precedente, ogni piano orizzontale rappresenta una singola istanza dell'ambiente di esecuzione (etichettata da A a F). Ecco come Lambda gestisce ogni richiesta:

Comportamento di Lambda per le richieste da 1 a 10

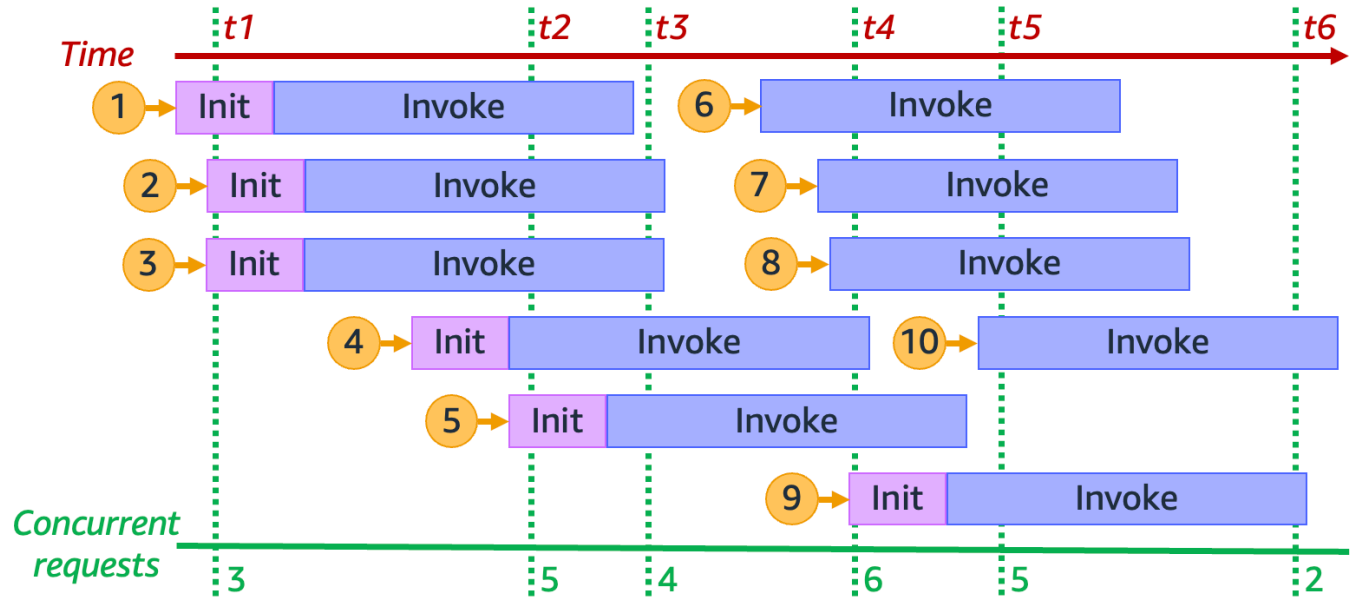
Richiesta	Comportamento di Lambda	Ragionamento
1	Fornisce il nuovo ambiente A	Questa è la prima richiesta; nessuna istanza dell'ambiente di esecuzione è disponibile.
2	Fornisce il nuovo ambiente B	L'istanza A dell'ambiente di esecuzione esistente è occupata.

Richiesta	Comportamento di Lambda	Ragionamento
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.
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'ambienteC	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 `t4`, 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 concorrenza 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:

```
Concurrency = (average requests per second) * (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:

```
Concurrency = (100 requests/second) * (1 second/request) = 100
```

Tuttavia, se la durata media della richiesta è di 500 ms, la simultaneità è 50:

```
Concurrency = (100 requests/second) * (0.5 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:

```
Concurrency = (100 requests/second) / (2 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:

```
Concurrency = (200 requests/second) * (0.25 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:

```
Concurrency = (5,000 requests/second) * (0.2 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:

```
Concurrency = (5,000 requests/second) / (5 requests/second) = 1,000
```

Differenziazione tra concorrenza e 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 generale, ogni istanza dell'ambiente di esecuzione è in grado di gestire al massimo 10 richieste al secondo. Questo limite si applica alle funzioni on demand sincrone e alle funzioni che utilizzano la simultaneità assegnata. Se non conosci questo limite, potrebbe non esserti chiaro il motivo per cui tali funzioni potrebbero subire una limitazione della larghezza di banda della rete in determinati scenari.

Ad esempio, consideriamo una funzione con una durata media della richiesta di 50 ms. A 200 richieste al secondo, ecco la simultaneità di questa funzione:

```
Concurrency = (200 requests/second) * (0.05 second/request) = 10
```

In base a questo risultato, potresti aspettarti di aver bisogno soltanto di 10 istanze dell'ambiente di esecuzione per gestire questo carico. Tuttavia, ogni ambiente di esecuzione può gestire solo 10 esecuzioni al secondo. Ciò significa che con 10 ambienti di esecuzione, la funzione può gestire solo 100 richieste al secondo su 200 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, hai bisogno di 20 ambienti di esecuzione per la tua funzione, anche se la simultaneità è soltanto di 10.

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 3.000 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 60:

```
Concurrency = (3,000 requests/second) * (0.02 seconds/request) = 60
```

Tuttavia, ogni ambiente di esecuzione può gestire solo 10 richieste al secondo. Con 60 ambienti di esecuzione, la funzione può gestire un massimo di 600 richieste al secondo. Per soddisfare tutte le 3.000 richieste, la funzione necessita di almeno 300 istanze dell'ambiente di esecuzione.

Comprensione della concorrenza riservata e della concorrenza 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. Se si esaurisce la simultaneità disponibile, la tua funzione subisce una limitazione della larghezza di banda della rete, ossia inizia a ignorare le richieste.

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 simultaneità riservata per riservare una parte della simultaneità riservata del tuo account per una funzione. Ciò è utile se non si desidera che altre funzioni occupino tutta la simultaneità non riservata disponibile.
- 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.

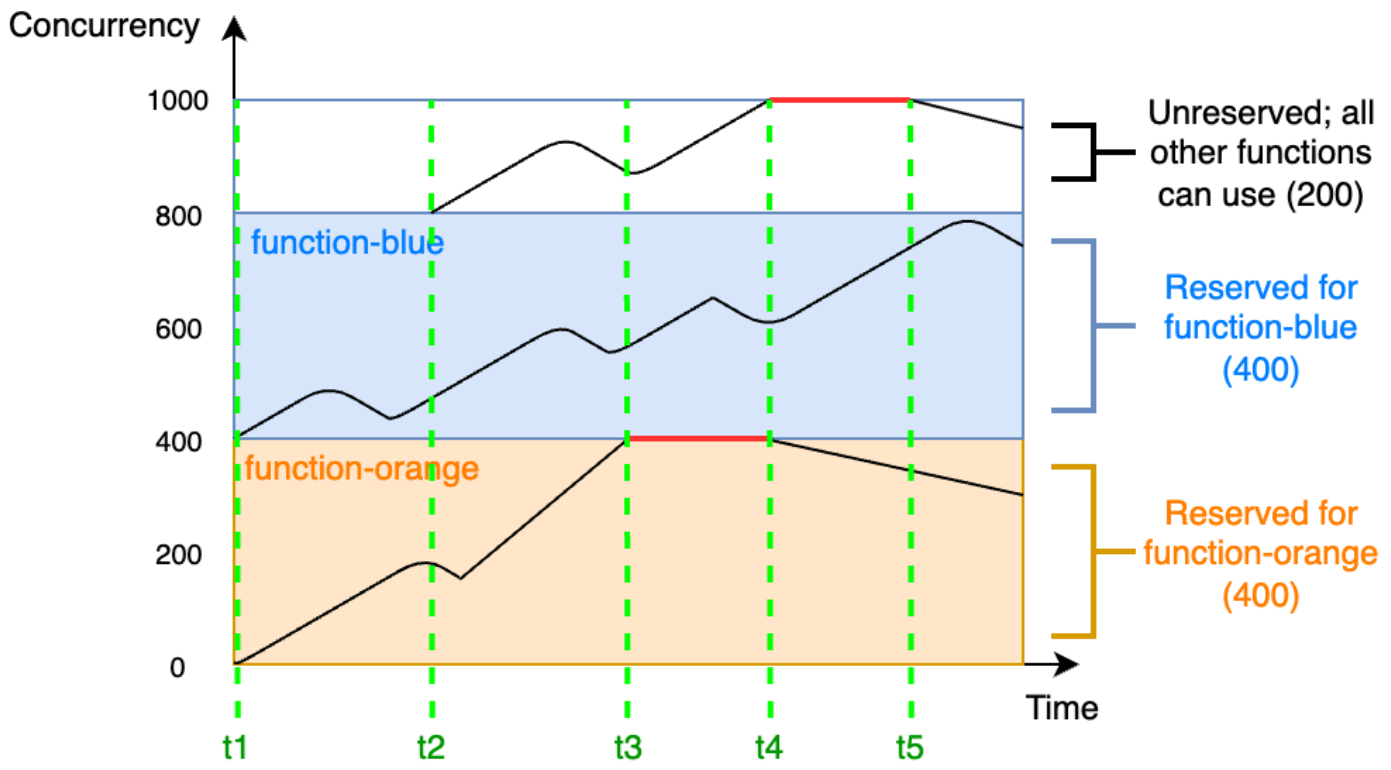
Simultaneità riservata

Se vuoi garantire che una certa quantità di simultaneità sia disponibile per la tua funzione in qualsiasi momento, usa la simultaneità riservata.

La simultaneità riservata è il numero massimo di istanze simultanee che desideri allocare alla 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 `t1`, 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 `t2`, `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 `t3`, `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 per `function-orange` si sta verificando una limitazione e Lambda potrebbe eliminare le richieste.

- Su `t4`, `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 `t5`, 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 simultaneità riservata pone un limite alla simultaneità massima della funzione. Ciò significa che la funzione non può utilizzare la simultaneità riservata ad altre funzioni o la simultaneità dal pool non riservato. 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 simultaneità 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 concorrenza riservata per una funzione.

Simultaneità fornita

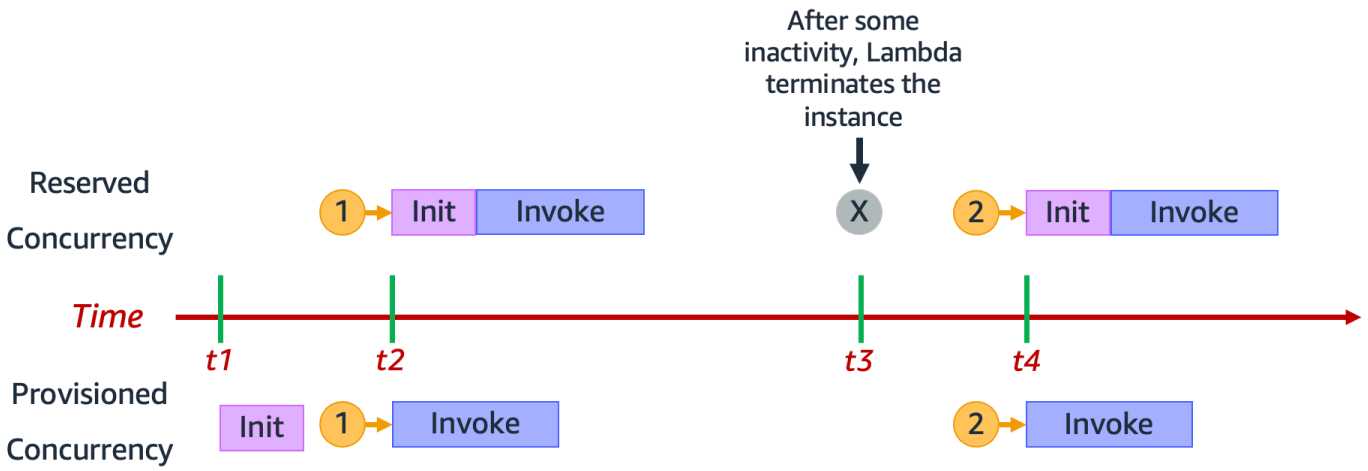
La simultaneità riservata viene utilizzata per definire il numero massimo di ambienti di esecuzione riservati a una funzione Lambda. Tuttavia, nessuno di questi ambienti è pre-inizializzato. Di conseguenza, le chiamate delle funzioni potrebbero richiedere più tempo perché Lambda deve inizializzare il 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 usare SnapStart Lambda 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 assegnare la concorrenza nella stessa versione della funzione. Per ulteriori informazioni su SnapStart funzionalità, limitazioni e regioni supportate, consulta. Migliorare le prestazioni di avvio con Lambda SnapStart

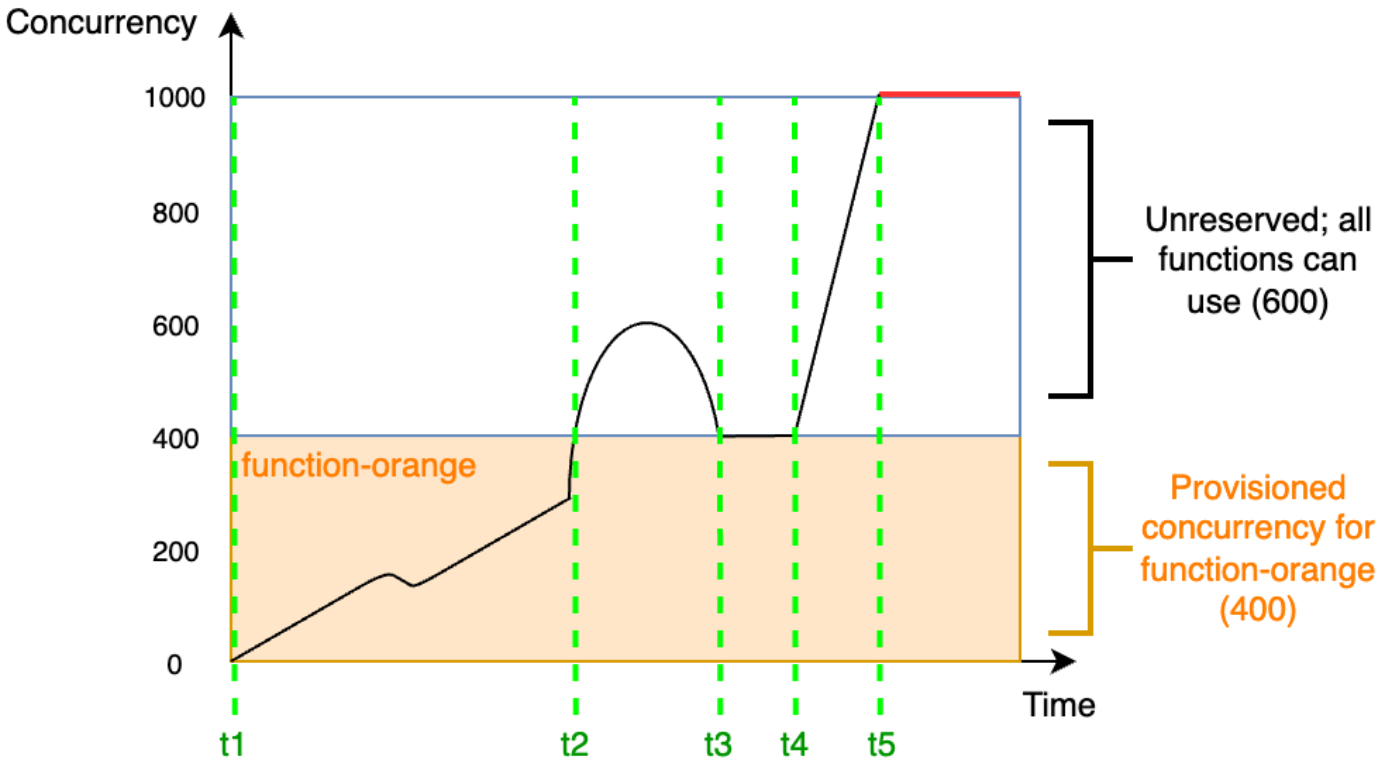
Quando si utilizza la simultaneità fornita, Lambda riavvia comunque gli ambienti di esecuzione in background. Tuttavia, in qualsiasi momento, Lambda garantisce sempre che il numero di ambienti pre-inizializzati sia uguale al valore dell'impostazione di simultaneità fornita dalla funzione. Questo comportamento è diverso dalla simultaneità riservata, in cui Lambda può terminare 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 fornita
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-inizi alizzata dell'ambiente.
t3	Dopo una certa inattività, Lambda termina l'istanza dell'ambiente attivo.	Non succede niente.
t4	Arriva la richiesta 2. Lambda deve inizializzare una nuova istanza dell'ambiente di esecuzione.	Arriva la richiesta 2. Lambda utilizza l'istanza pre-inizi alizzata dell'ambiente.

Per comprendere meglio la simultaneità fornita, considera il seguente diagramma:



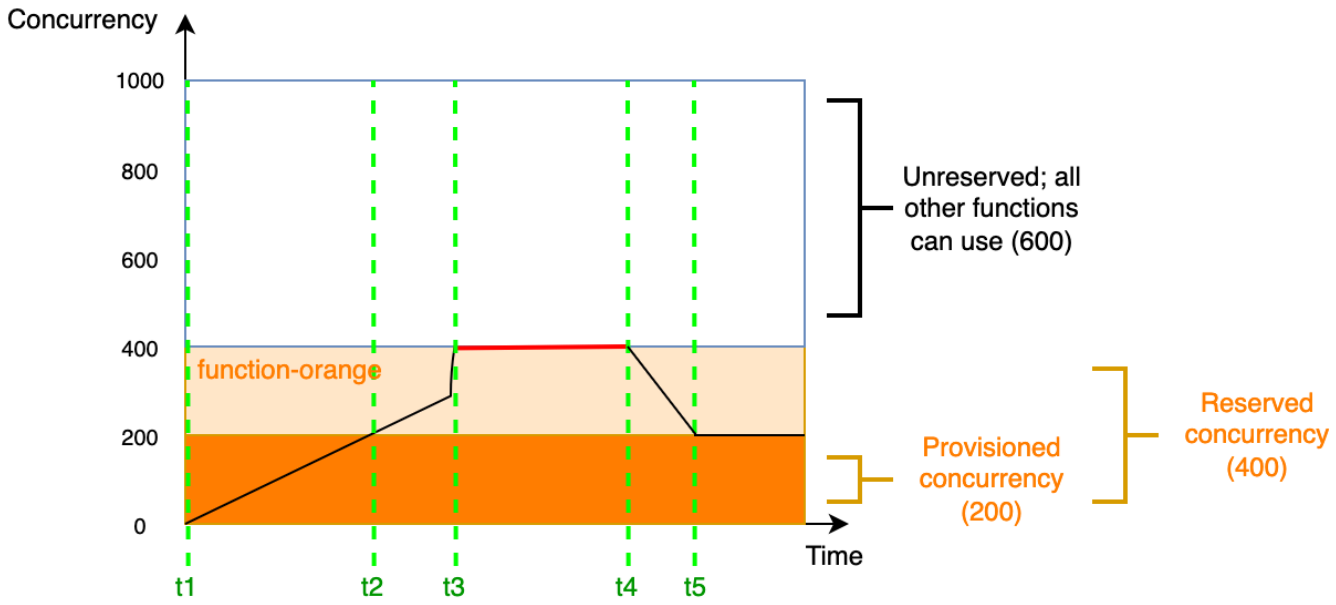
In questo diagramma, hai un limite di simultaneità dell'account pari a 1.000. Decidi di assegnare 400 unità di simultaneità fornita a `function-orange`. Tutte le funzioni del tuo account, inclusa `function-orange`, possono utilizzare le restanti 600 unità di simultaneità non riservata.

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 presentare latenze di avvio a freddo.
- Su `t3`, `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 `t4`, 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 `t5`, 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.
- Su `t2`, `function-orange` utilizza tutta la simultaneità fornita. `function-orange` può continuare a gestire le richieste utilizzando la simultaneità 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 fornita 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 concorrenza 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 simultanea fornita.

Come Lambda alloca la simultaneità fornita

La simultaneità fornita non è disponibile online immediatamente dopo la configurazione. Lambda avvia l'allocazione della simultaneità con provisioning 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 AWSÈ 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 ambienti di esecuzione simultanei, nessuna delle tue richieste può utilizzare la concorrenza fornita fino a quando Lambda non avrà completato completamente 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.

Argomento	Simultaneità riservata	Simultaneità fornita
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 provision ing	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 limitazio ne	Funzione limitata quando viene raggiunto il limite di simultaneità riservata.	Se la simultaneità riservata non è impostata: quando viene raggiunto il limite di simultane ità fornita la funzione utilizza la simultaneità non riservata. Se è impostata la simultane ità riservata: quando viene raggiunto il limite di simultane ità riservata la funzione viene limitata.
Comportamento predefinito se non impostato	La funzione utilizza la simultaneità non riservata disponibile nel tuo account.	Lambda non fornisce alcuna istanza in anticipo. Invece, se la simultaneità riservata non è impostata: la funzione utilizza

Argomento	Simultaneità riservata	Simultaneità fornita
		la simultaneità non riservata disponibile nel tuo account. Se è impostata la simultaneità riservata: la funzione utilizza la simultaneità riservata.
Prezzi	Nessun costo aggiuntivo.	Incorre in costi aggiuntivi.

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.

Per verificare la quota di concorrenza a livello di account corrente, utilizzate il comando AWS Command Line Interface ()AWS CLI per eseguire il comando seguente:

```
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 simultanea è la velocità massima alla quale le funzioni del tuo account possono scalare in risposta a un aumento delle richieste. Si tratta della velocità con cui Lambda può creare nuovi ambienti di esecuzione. Il tasso di scalabilità simultanea è diverso dal limite di concorrenza a livello di account, che è la quantità totale di concorrenza disponibile per le tue funzioni.

In ogni funzione e per ogni funzione Regione AWS, il tasso di scalabilità simultanea è di 1.000 istanze dell'ambiente di esecuzione ogni 10 secondi. In altre parole, ogni 10 secondi, Lambda può allocare al massimo 1.000 istanze aggiuntive dell'ambiente di esecuzione 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 simultanea 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 concorrenza 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 di istanze simultanee allocate alla funzione. Quando una funzione ha la simultaneità riservata, nessun'altra funzione può utilizzare tale simultaneità. La concorrenza riservata è utile per garantire che le funzioni più critiche abbiano sempre una concorrenza sufficiente per gestire le richieste in arrivo. 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 concorrenza fornita è utile per ridurre le latenze di avvio a freddo delle funzioni. La configurazione della concorrenza fornita comporta costi aggiuntivi per l'utente. 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 AWS Support.

Sections

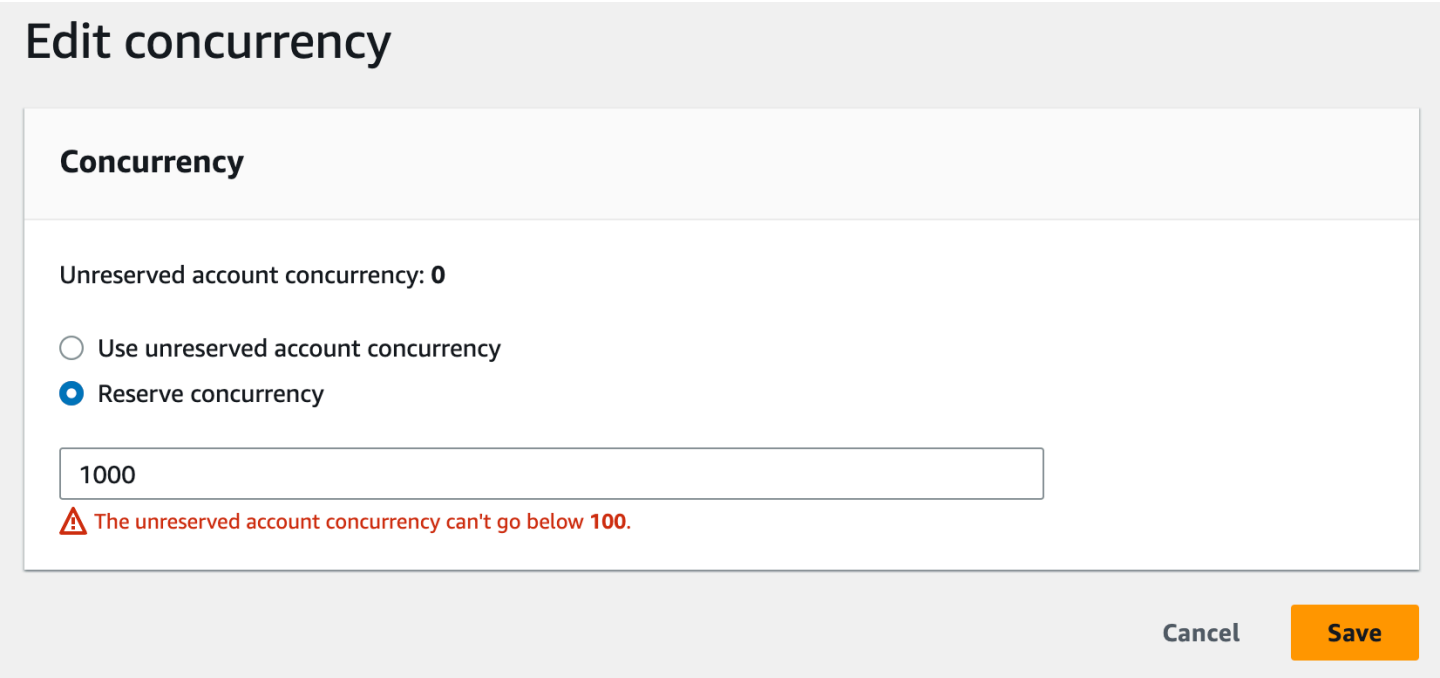
Configurazione della simultaneità riservata

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 la quantità di simultaneità da riservare per la funzione.
6. Selezionare 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.



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.

- PutFunctionConcorrenza
- GetFunctionConcorrenza
- DeleteFunctionConcorrenza

Ad esempio, per configurare la concorrenza riservata con AWS Command Line Interface (CLI), utilizzate `put-function-concurrency` il comando. Il comando seguente riserva 100 unità di simultaneità per una funzione denominata `my-function`:

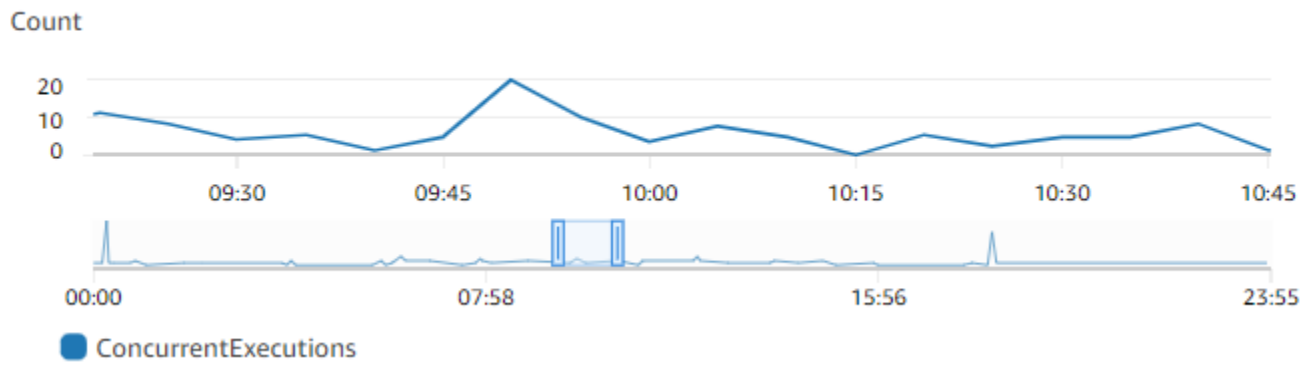
```
aws lambda put-function-concurrency --function-name my-function \
    --reserved-concurrent-executions 100
```

L'output restituito dovrebbe essere simile al seguente:

```
{
    "ReservedConcurrentExecutions": 100
}
```

Stima accurata della concorrenza riservata richiesta per una funzione

Se la tua funzione attualmente serve traffico, puoi visualizzare facilmente le sue 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 dei parametri delle funzioni Lambda.

Configurazione della concorrenza 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 di istanze simultanee allocate alla funzione. Quando una funzione ha la simultaneità riservata, nessun'altra funzione può utilizzare tale simultaneità. La concorrenza riservata è utile per garantire che le funzioni più critiche abbiano sempre una concorrenza sufficiente per gestire le richieste in arrivo. 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 concorrenza fornita è utile per ridurre le latenze di avvio a freddo delle funzioni. La configurazione della concorrenza fornita comporta costi aggiuntivi per l'utente. 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 AWS Support.

Sections

- Utilizzo di variabili di ambiente per visualizzare e controllare il comportamento di concorrenza assegnato
- Comprensione del comportamento di registrazione e fatturazione con la concorrenza fornita
- Utilizzo di Application Auto Scaling per automatizzare la gestione simultanea fornita

Configurazione della simultaneità fornita

È possibile configurare le impostazioni di simultaneità fornita per una funzione utilizzando la console Lambda o l'API Lambda.

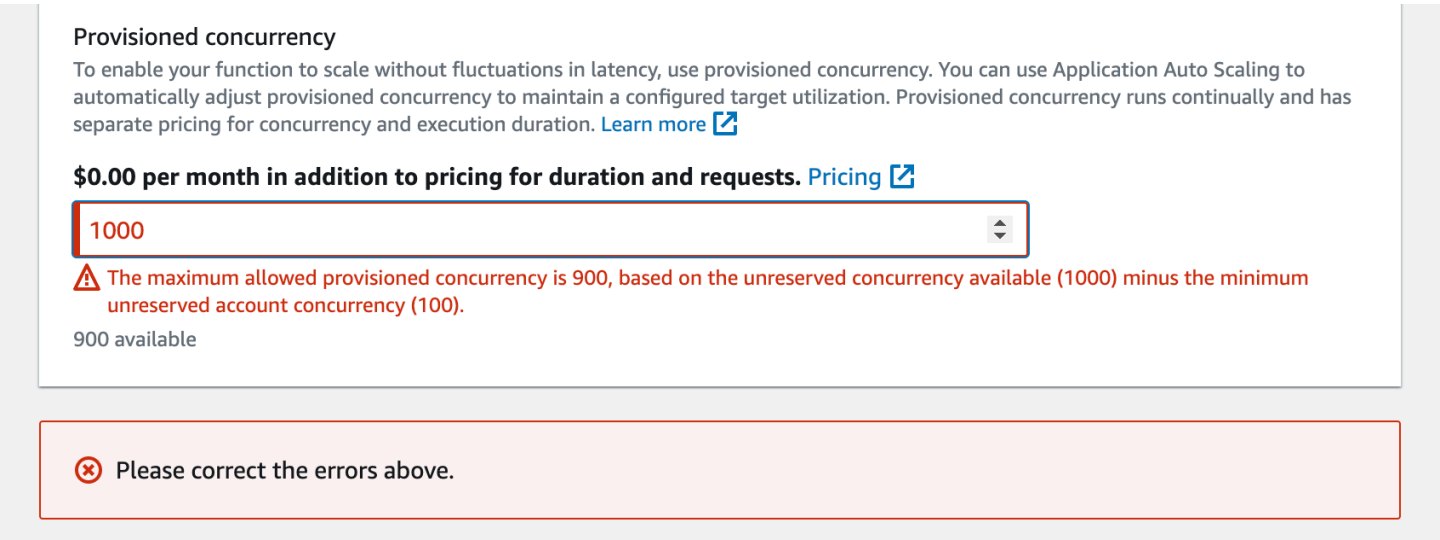
Allocazione della simultaneità fornita per una funzione (console)

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. Lambda fornisce una stima dei costi mensili.
7. Selezionare 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.



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 fornita con (AWS Command Line Interface CLI), utilizzate 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 \
```

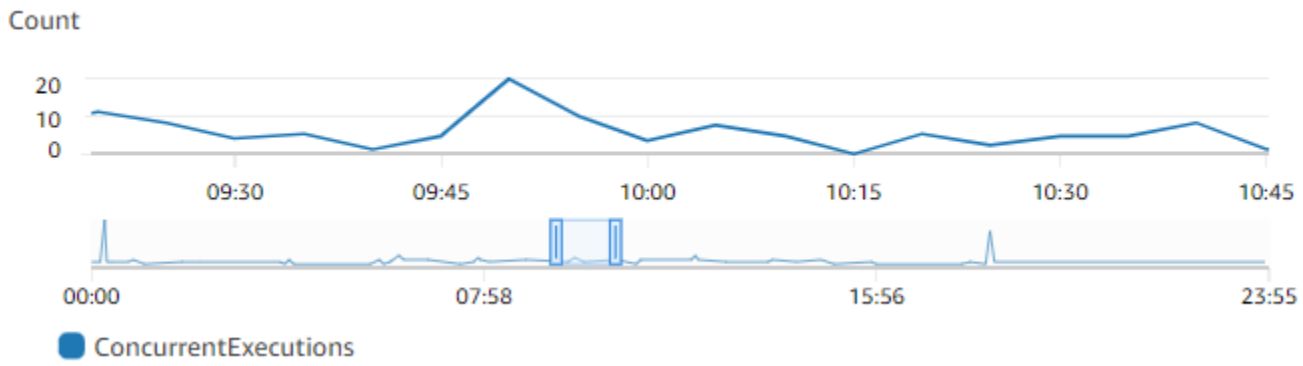
```
  --provisioned-concurrent-executions 100
```

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 concorrenza 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:

```
Concurrency = (average requests per second) * (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 concorrenza fornita

Se utilizzi la concorrenza fornita, prendi in considerazione la possibilità di ristrutturare il codice della funzione per ottimizzarlo per una bassa latenza. Per le funzioni che utilizzano la concorrenza fornita, Lambda esegue qualsiasi codice di inizializzazione, come il caricamento di librerie e l'istanziazione dei client, durante il tempo 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 delle librerie o l'istanziazione di client all'interno del codice del gestore principale significa che la funzione deve essere eseguita ogni volta che viene richiamata (ciò avviene indipendentemente dal fatto che si stia utilizzando la concorrenza fornita).

Per le invocazioni on demand, 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 rinviare l'inizializzazione di una funzionalità specifica fino a quando la funzione non ne ha necessità. 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.

Utilizzo di variabili di ambiente per visualizzare e controllare il comportamento di concorrenza assegnato

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, vedere. ???

Se utilizzi runtime .NET 6 o .NET 7, puoi configurare la variabile di ambiente `AWS_LAMBDA_DOTNET_PREJIT` per migliorare la latenza delle funzioni, anche laddove non utilizzino la simultaneità fornita. 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 invocazione di una funzione Lambda può richiedere più tempo delle successive. Per ovviare a questo problema, puoi scegliere tra tre valori per `AWS_LAMBDA_DOTNET_PREJIT`:

- `ProvisionedConcurrency`: Lambda esegue la compilazione ahead-of-time JIT per tutti gli ambienti utilizzando la concorrenza fornita. Si tratta del valore di default.
- `Always`: Lambda esegue la compilazione ahead-of-time JIT per ogni ambiente, anche se la funzione non utilizza la concorrenza fornita.
- `Never`: Lambda disabilita la compilazione ahead-of-time JIT per tutti gli ambienti.

Comprensione del comportamento di registrazione e fatturazione con la concorrenza 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. Puoi vedere il tempo di inizializzazione nei log e nelle tracce dopo che un'istanza dell'ambiente elabora una richiesta. È importante tenere presente che Lambda addebita i costi di inizializzazione anche se l'istanza non elabora mai richieste. 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.

Inoltre, quando configuri una funzione Lambda con la concorrenza fornita, Lambda preinizializza l'ambiente di esecuzione in modo che sia disponibile prima delle richieste di invocazione della

funzione. Tuttavia, la funzione pubblica i log delle chiamate solo quando la funzione viene effettivamente richiamata. CloudWatch Pertanto, il campo Init Duration viene visualizzato nella riga di `REPORT` registro della prima chiamata della funzione, anche se l'inizializzazione è avvenuta in anticipo. Ciò non significa che la funzione abbia subito un avvio a freddo.

Utilizzo di Application Auto Scaling per automatizzare la gestione simultanea 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.

Dimensionamento programmato

Con Application Auto Scaling, puoi creare una pianificazione personalizzata in base alle variazioni di carico prevedibili. Per ulteriori informazioni ed esempi, consulta Scheduled scaling for Application Auto Scaling nella Application Auto Scaling User Guide AWS Lambda e Scheduling Provisioned Concurrency per i picchi di utilizzo ricorrenti sul blog di Compute. AWS

Monitoraggio degli obiettivi

Con il tracciamento degli obiettivi, Application Auto Scaling crea e gestisce una serie di CloudWatch allarmi in base alla definizione della politica di scalabilità. 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 la 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. L'esempio seguente configura Application Auto Scaling per modificare la configurazione di concorrenza fornita per un alias in

modo da mantenere l'utilizzo vicino al 70 percento, 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" }}'
```

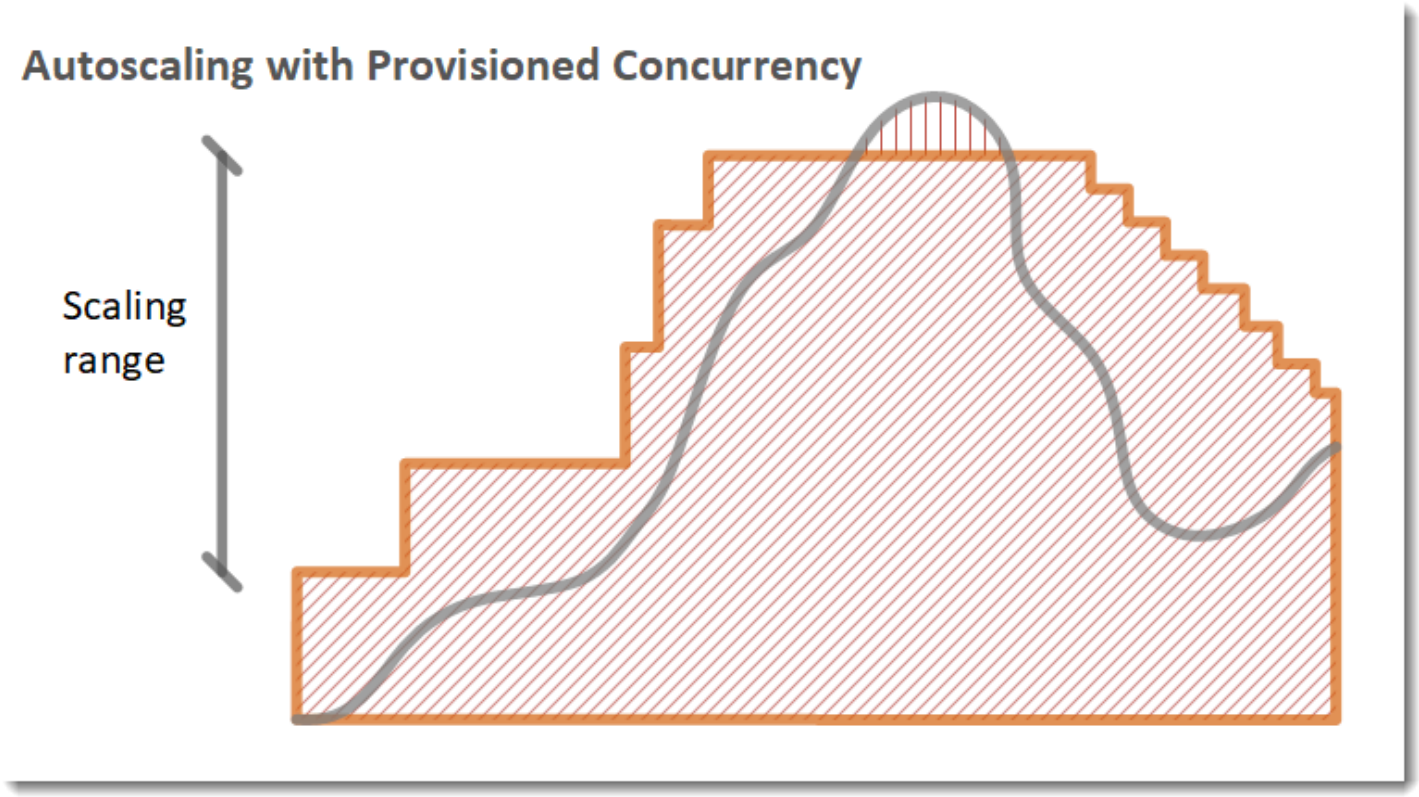
L'output visualizzato dovrebbe essere di questo tipo:

```
{
    "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. 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 con provisioning 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.

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.



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 rilevano picchi di traffico rapidi 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 carico di picco duri almeno 3 minuti allo scopo di fornire ambienti aggiuntivi. Inoltre, entrambi gli CloudWatch allarmi richiedono 3 punti dati che raggiungano la media target per attivare la politica di auto scaling. Se la tua funzione registra rapidi picchi di traffico, l'utilizzo della statistica Maximum anziché della statistica Average può essere più efficace nel ridimensionare la concorrenza fornita per ridurre al minimo gli avviamenti 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. Questo tasso di scalabilità simultanea è la velocità massima alla quale le funzioni del tuo account possono scalare in risposta a un aumento delle richieste. Si tratta della velocità con cui Lambda può creare nuovi ambienti di esecuzione. Il tasso di scalabilità simultanea è diverso dal limite di concorrenza a livello di account, che è la quantità totale di concorrenza disponibile per le tue 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. In altre parole, ogni 10 secondi, Lambda può allocare al massimo 1.000 istanze aggiuntive dell'ambiente di esecuzione 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 il tasso di scalabilità della concorrenza 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 i parametri di CloudWatch 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

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 con provisioning

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.

- `ProvisionedConcurrentInvocations`: 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.

`ProvisionedConcurrentInvocations` differisce da `ProvisionedConcurrentExecutions` per il fatto che `ProvisionedConcurrentInvocations` 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 (`ProvisionedConcurrentInvocations`) 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 (`ProvisionedConcurrentInvocations`) 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 `ProvisionedConcurrentInvocations` + `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 (`ProvisionedConcurrentUtilization`) è 0,6.

Un valore elevato di `ProvisionedConcurrencySpilloverInvocations` può indicare che è necessario allocare simultaneità fornita aggiuntiva per la funzione. In alternativa, è possibile configurare Application Auto Scaling per gestire il dimensionamento automatico della simultaneità fornita in base a soglie predefinite.

Viceversa, valori costantemente bassi di `ProvisionedConcurrencyUtilization` possono indicare che hai allocato una quantità eccessiva di simultaneità fornita 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:

```
ClaimedAccountConcurrency = UnreservedConcurrentExecutions + (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"):

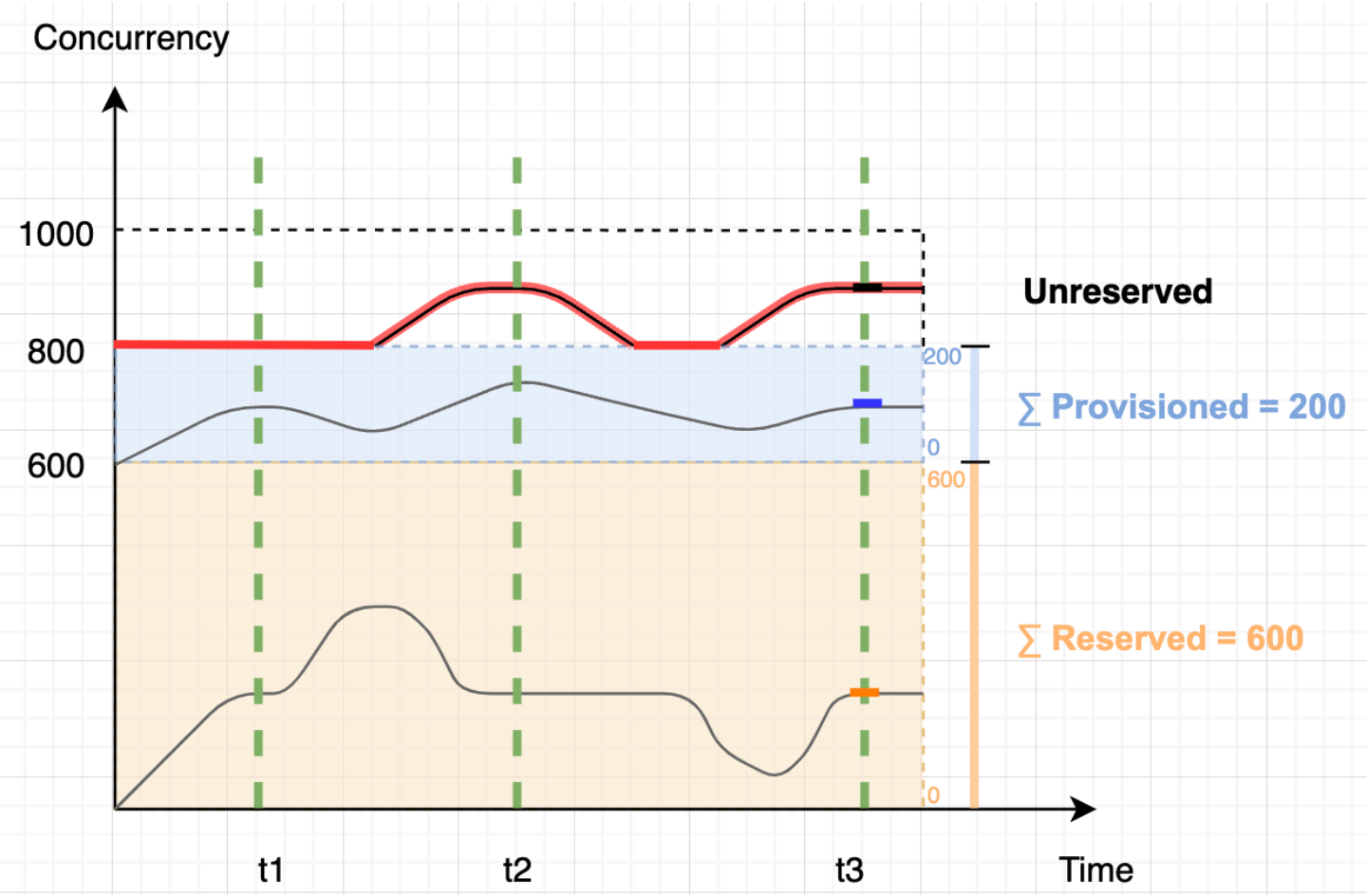
- Il valore RC totale in tutte le funzioni di una Regione.
- Il valore PC totale in tutte le funzioni di una Regione che utilizzanoPC, escluse le funzioni che utilizzano RC.

> **Note**
>
> Non puoi allocare più PC di RC per una funzione. Pertanto, il valore RC di una funzione è sempre maggiore o uguale al suo valore 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`. Assegni 600 unità di simultaneità riservata per `function-orange`. Assegni 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`).

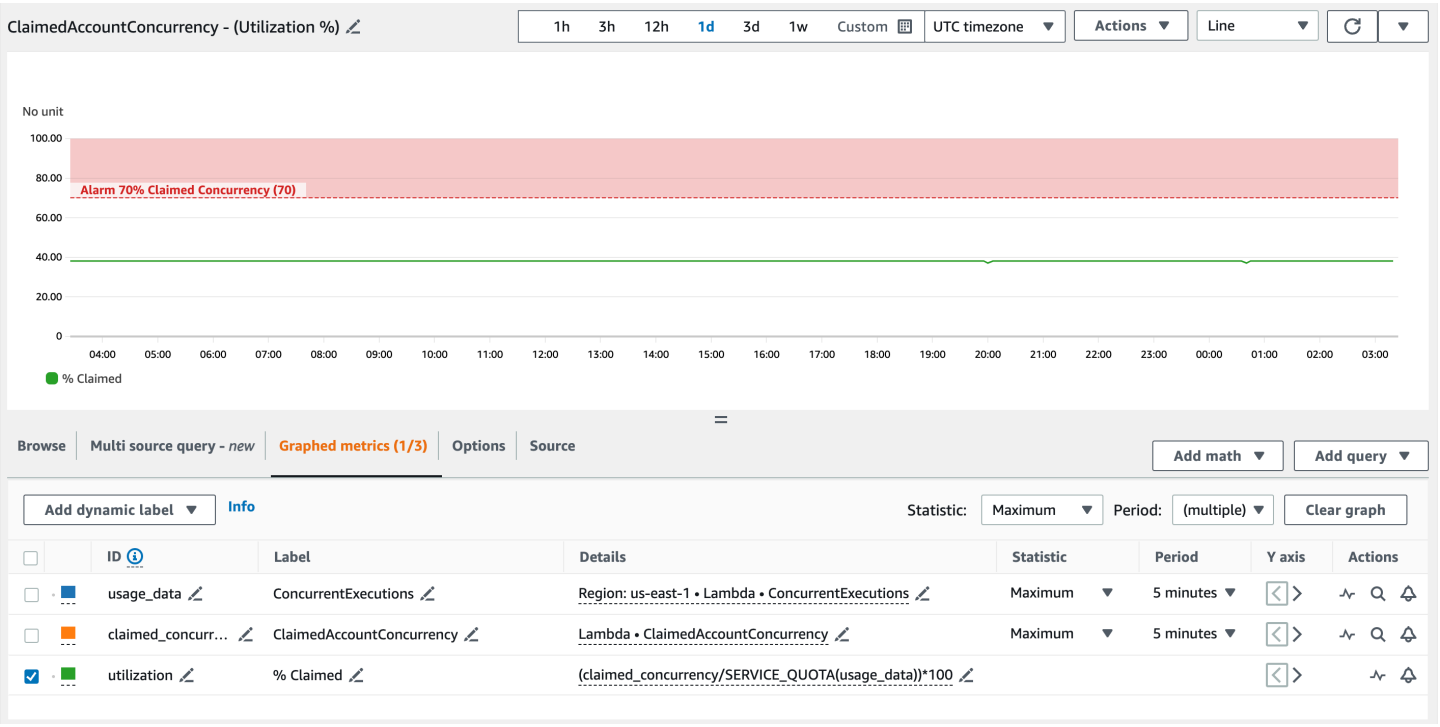
- In `t3`, `ClaimedAccountConcurrency` è di nuovo 900 (800 + 100 `UnreservedConcurrentExecutions`).

Configurazione 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:

```
Utilization = (ClaimedAccountConcurrency/SERVICE_QUOTA(ConcurrentExecutions)) * 100%
```

La schermata seguente illustra come inserire graficamente questa formula. 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 entra in `ALARM` stato 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.

Configurazione della firma del codice per AWS Lambda

La firma del codice AWS Lambda aiuta a garantire che nelle funzioni Lambda venga eseguito solo codice affidabile. Quando si attiva la firma del codice per una funzione, Lambda controlla ogni distribuzione del codice e verifica che il pacchetto di codice sia firmato da un'origine attendibile.

> **Note**
>
> Le funzioni definite come immagini di container non supportano la firma del codice.

Per verificare l'integrità del codice, utilizzare AWS Signer per creare pacchetti di codice con firma digitale per funzioni e livelli. Quando un utente tenta di distribuire un pacchetto di codice, Lambda esegue controlli di convalida sul pacchetto di codice prima di accettarne la distribuzione. Poiché i controlli di convalida della firma del codice vengono eseguiti al momento della distribuzione, non vi è alcun impatto sulle prestazioni nell'esecuzione delle funzioni.

Puoi utilizzare AWS Signer anche per creare profili di firma. È possibile utilizzare un profilo di firma per creare il pacchetto di codice firmato. Utilizza AWS Identity and Access Management (IAM) per controllare chi può firmare pacchetti di codice e creare profili di firma. Per ulteriori informazioni, consultare Autenticazione e controllo degli accessi nel manuale Guida per l'utente di AWS Signer.

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.

I livelli Lambda seguono lo stesso formato del pacchetto di codice firmato dei pacchetti di codici della funzione. Quando si aggiunge un livello a una funzione per la quale è attivata la firma del codice, Lambda verifica che il livello sia firmato da un profilo di firma consentito. Quando si attiva la firma del codice per una funzione, tutti i livelli aggiunti alla funzione devono essere firmati anche da uno dei profili di firma consentiti.

Utilizzare IAM per controllare chi può creare configurazioni di firma del codice. In genere, è consentito solo a utenti amministrativi specifici di avere questa capacità. Inoltre, è possibile impostare delle policy IAM per imporre agli sviluppatori di creare solo funzioni per le quali è abilitata la firma del codice.

È possibile configurare la firma del codice per registrare le modifiche su AWS CloudTrail. Le implementazioni delle funzioni riuscite e bloccate vengono registrate CloudTrail con informazioni sui controlli di firma e convalida.

Puoi configurare la firma del codice per le tue funzioni utilizzando la console Lambda, il AWS Command Line Interface (AWS CLI) e il AWS Serverless Application Model (AWS SAM). AWS CloudFormation

Non sono previsti costi aggiuntivi per l'utilizzo di AWS Signer o per la firma del codice. AWS Lambda

Sections

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 uno dei profili di firma consentiti per la funzione Lambda. Si verifica una mancata corrispondenza anche se non è presente una firma.
4. Revoca - Convalida che la firma del pacchetto di codice non sia stata revocata.

La policy di convalida della firma definito nella configurazione della firma del codice determina quale delle seguenti azioni sia eseguito da Lambda se uno dei controlli di convalida ha esito negativo:

- Avvisa - Lambda consente la distribuzione del pacchetto di codice, ma emette un avviso. Lambda emette una nuova CloudWatch metrica Amazon e memorizza anche l'avviso nel registro. CloudTrail
- Applica - Lambda emette un avviso (lo stesso dell'azione Avvisa) e blocca la distribuzione del pacchetto di codice.

È possibile configurare la policy per i controlli di convalida di scadenza, mancata corrispondenza e revoca. Si noti che non è possibile configurare una policy per il controllo dell'integrità. Se il controllo dell'integrità non riesce, Lambda blocca la distribuzione.

Prerequisiti di configurazione

Prima di configurare la firma del codice per una funzione Lambda, usa 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.

Per ulteriori informazioni, consultare la sezione Creazione di profili di firma (Console) nella Guida per lo sviluppatore di AWS Signer.

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 Create configuration (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 Create configuration (Crea configurazione).

Aggiornamento di una configurazione di firma del codice

Quando si aggiorna una configurazione di firma del codice, le modifiche influiscono sulle distribuzioni future di funzioni a cui è collegata la configurazione della firma del codice.

Per aggiornare una configurazione di firma del codice (console)

1. Aprire la pagina Configurazioni di firma del codice della console Lambda.
2. Selezionare una configurazione di firma del codice da aggiornare, quindi scegliere Edit (Modifica).
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. Seleziona Salvataggio delle modifiche.

Eliminazione di una configurazione di firma del codice

È possibile eliminare una configurazione di firma del codice solo se non viene utilizzata da alcuna funzione.

Per eliminare una configurazione di firma del codice (console)

1. Aprire la pagina Configurazioni di firma del codice della console Lambda.
2. Selezionare una configurazione di firma del codice da eliminare e quindi scegliere Delete (Elimina).
3. Per confermare, scegliere di nuovo Delete (Elimina).

Abilitazione della firma del codice per una funzione

Per abilitare la firma del codice per una funzione, è necessario associare una configurazione di firma del codice alla funzione.

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 Code signature.
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 concedere l'autorizzazione a un utente di accedere alle operazioni dell'API di firma del codice, allegare una o più istruzioni delle policy alla policy dell'utente. Per ulteriori informazioni sulle policy utente, consulta Utilizzo di politiche IAM basate sull'identità in Lambda.

La seguente istruzione di policy di esempio concede l'autorizzazione per creare, aggiornare e recuperare le configurazioni di firma del codice.

```
{
  "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 qualsiasi richiesta API `CreateFunction` se il relativo parametro `CodeSigningConfigArn` è mancante o non corrisponde al valore nella condizione.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowReferencingCodeSigningConfig",
      "Effect": "Allow",
      "Action": [
          "lambda:CreateFunction",
        ],
      "Resource": "*",
      "Condition": {
          "StringEquals": {
              "lambda:CodeSigningConfigArn":
                  “arn:aws:lambda:us-west-2:123456789012:code-signing-
config:csc-0d4518bd353a0a7c6”
          }
      }
    }
  ]
}
```

Configurazione di firma del codice con l'API Lambda

Per gestire le configurazioni di firma del codice con AWS CLI o AWS SDK, utilizza le seguenti operazioni API:

- ListCodeSigningConfigs
- CreateCodeSigningConfig

- GetCodeSigningConfig
- UpdateCodeSigningConfig
- DeleteCodeSigningConfig

Per gestire la configurazione di firma del codice per una funzione, utilizzare le seguenti operazioni API:

- CreateFunction
- GetFunctionCodeSigningConfig
- PutFunctionCodeSigningConfig
- DeleteFunctionCodeSigningConfig
- ListFunctionsByCodeSigningConfig

Uso dei tag sulle funzioni Lambda

Puoi aggiungere un tag alle funzioni AWS Lambda per attivare il controllo degli accessi basato su attributi (ABAC) e organizzarle in base al proprietario, al progetto o al reparto. I tag sono coppie chiave-valore a forma libera supportate dai servizi AWS per l'utilizzo in ABAC, per il filtraggio delle risorse e l'aggiunta di dettagli ai report di fatturazione.

I tag si applicano a livello di funzione, non a versioni o alias. I tag non fanno parte della configurazione specifica della versione di cui Lambda crea uno snapshot quando pubblichi una versione.

Sections

Autorizzazioni necessarie per lavorare con i tag

Concedi le autorizzazioni appropriate all'identità AWS Identity and Access Management IAM (utente, gruppo o ruolo) per la persona che lavora con la funzione:

- lambda: ListTags — Quando una funzione ha dei tag, concedi questa autorizzazione a chiunque abbia bisogno di chiamarla `GetFunction` o `ListTags` attivarla.
- lambda: TagResource — Concedi questa autorizzazione a chiunque abbia bisogno di chiamare `CreateFunction` o. `TagResource`

Per ulteriori informazioni, consulta Utilizzo di politiche IAM basate sull'identità in 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.

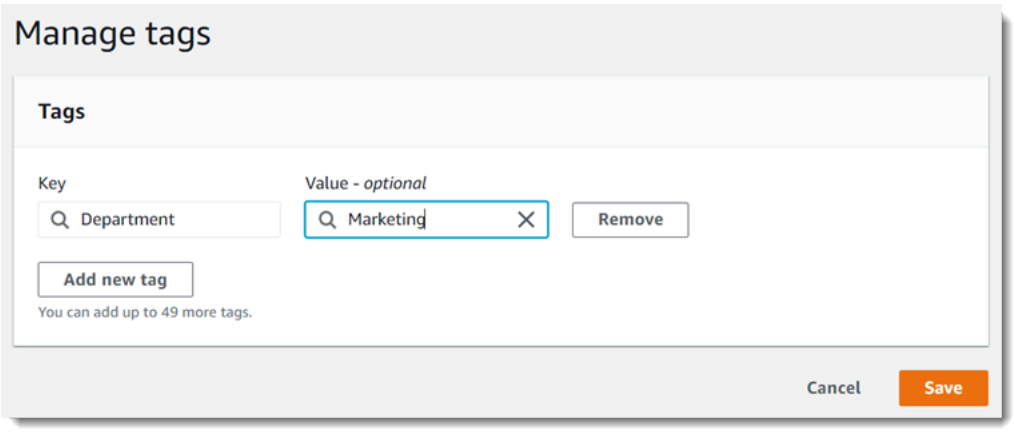
Per aggiungere tag durante la creazione di una funzione

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 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 l'architettura del set di istruzioni scelta.
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

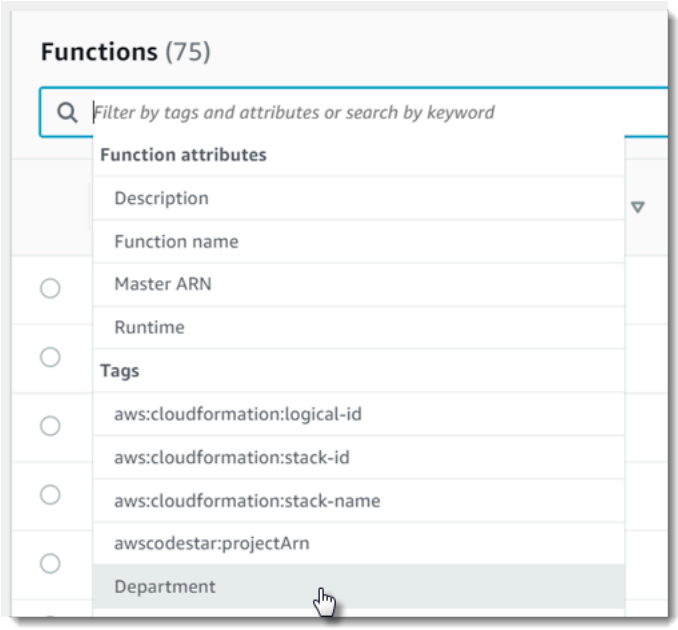
1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Scegli Configuration (Configurazione), quindi Tags (Tag).
4. In Tag, scegli Gestisci tag.
5. Scegli Add new tag (Aggiungi nuovo tag) e completa i campi Key (Chiave) e facoltativamente Value (Valore). Ripetere questa fase per aggiungere altri tag.



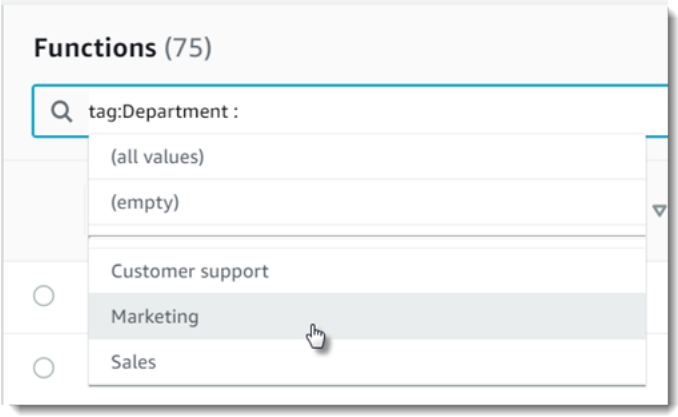
6. Selezionare Salva.

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 attributi di funzione e chiavi di tag.



3. Scegli una chiave di tag per visualizzare un elenco di valori in uso nella regione AWS corrente.
4. Scegliere un valore per visualizzare le funzioni con quel valore, oppure scegliere (tutti i valori) per visualizzare tutte le funzioni che hanno un tag con quella chiave.



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.

Utilizzo dei tag con la AWS CLI

Aggiunta e rimozione di tag

Per creare una nuova funzione Lambda con i tag, utilizza il comando create-function con l'opzione --tags.

```
aws lambda create-function --function-name my-function
--handler index.js --runtime nodejs20.x \
--role arn:aws:iam::123456789012:role/lambda-role \
--tags Department=Marketing,CostCenter=1234ABCD
```

Per aggiungere i tag alla funzione, usa il comando tag-resource.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:function:my-function \
--tags Department=Marketing,CostCenter=1234ABCD
```

Pr rimuovere i tag, utilizza il comando untag-resource.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:function:my-function \
--tag-keys Department
```

Visualizzazione dei tag su una funzione

Se desideri visualizzare i tag applicati a una funzione Lambda specifica, puoi utilizzare uno dei seguenti comandi della AWS CLI:

- ListTags— Per visualizzare un elenco dei tag associati a questa funzione, includi la funzione Lambda ARN (Amazon Resource Name):

  ```
  aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:function:my-function
  ```

- GetFunction— Per visualizzare un elenco dei tag associati a questa funzione, includi il nome della funzione Lambda:

  ```
  aws lambda get-function --function-name my-function
  ```

Funzioni di filtro per tag

È possibile utilizzare l'operazione AWS Resource Groups Tagging API GetResourcesAPI per filtrare le risorse in base ai 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.

Per ulteriori informazioni su AWS Resource Groups, consulta Cosa sono i gruppi di risorse? nella Guida per l'utente di AWS Resource Groups e tag.

Requisiti per i tag

I seguenti requisiti si applicano ai tag:

- numero massimo di tag per risorsa: 50
- Lunghezza massima della chiave: 128 caratteri Unicode in formato UTF-8
- Lunghezza massima del valore: 256 caratteri Unicode in formato UTF-8
- I valori e le chiavi dei tag rispettano la distinzione tra maiuscole e minuscole.
- Non utilizzare il prefisso `aws:` nei nomi o nei valori di tag perché è riservato per essere utilizzato in AWS. Non è possibile modificare né eliminare i nomi o i valori di tag con tale prefisso. I tag con questo prefisso non vengono conteggiati per il limite del numero di tag per risorsa.
- Se prevedi di utilizzare lo schema di aggiunta tag in più servizi e risorse, ricorda che in altri servizi possono essere presenti delle limitazioni sui caratteri consentiti. I caratteri consentiti sono in genere lettere, spazi e numeri rappresentabili in formato UTF-8, più i caratteri speciali: + - = . _ : / @.

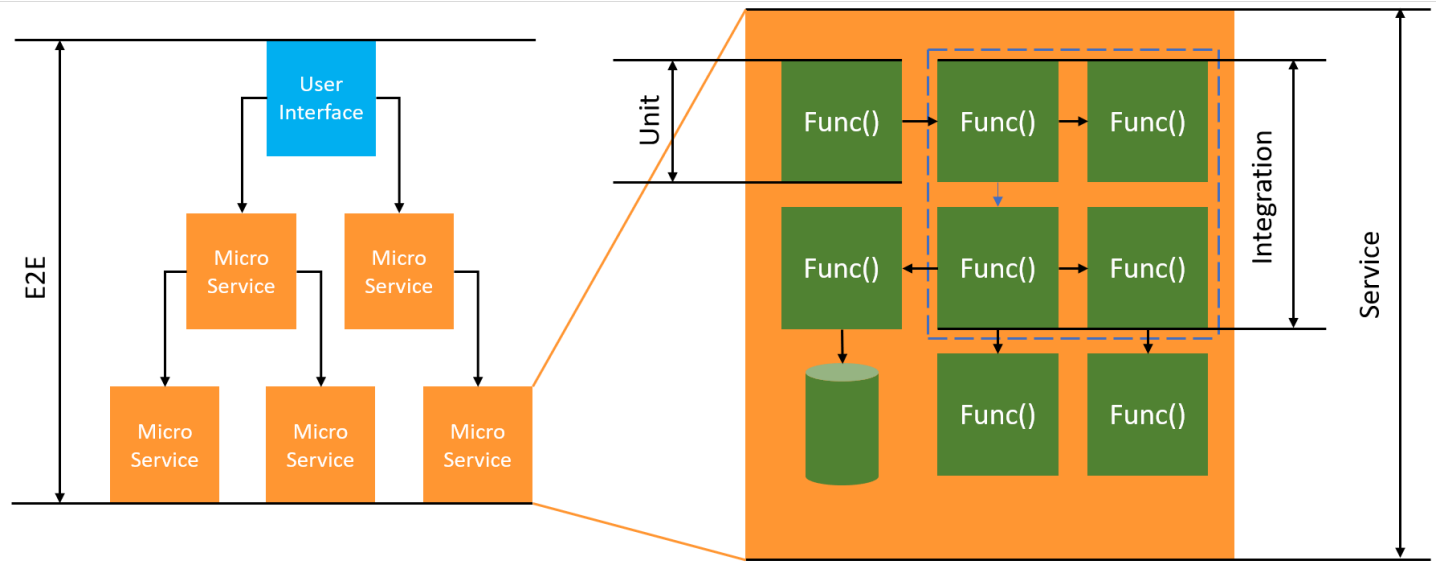
Come testare funzioni e applicazioni serverless

Sebbene il test delle funzioni serverless si basi su tipologie e tecniche di test consolidate, è importante tenere in considerazione la possibilità di effettuare test sulle applicazioni serverless nella loro interezza. I test basati sul cloud forniranno la misura più accurata della qualità sia delle funzioni sia 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 in ambito serverless, è necessario scrivere test unitari, test di integrazione e test 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.
- E nd-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

Le verifiche delle soluzioni serverless possono richiedere un po' più di tempo per configurare test in grado di analizzare le interazioni basate sugli eventi tra i servizi. Durante la lettura della presente guida, tieni a mente i seguenti aspetti pratici per l'azienda:

- Aumenta la qualità della tua applicazione
- Riduci il tempo necessario per creare funzionalità e correggere bug

La qualità di un'applicazione dipende dai test condotti per verificarne la funzionalità in vari scenari. Per migliorare la qualità della tua applicazione, è consigliabile valutare attentamente gli scenari aziendali e automatizzare i relativi test, eseguendoli sui servizi cloud.

Rilevare i bug del software e i problemi di configurazione durante un ciclo di sviluppo iterativo può avere un impatto minimo sui costi e sulla pianificazione. Se i problemi non vengono rilevati durante lo sviluppo, la ricerca e la risoluzione in fase di produzione richiede un maggiore impegno da parte di più persone.

Una strategia di test in ambito serverless ben pianificata aumenterà la qualità del software e migliorerà i tempi di iterazione, verificando che le applicazioni e le funzioni Lambda funzionino come previsto in un ambiente cloud.

Cosa testare

Ti consigliamo di adottare una strategia di test che verifichi i comportamenti dei servizi gestiti, la configurazione del cloud, le policy di sicurezza e l'integrazione con il codice per migliorare la qualità del software. I test comportamentali, noti anche come test della scatola nera, sono progettati per verificare il corretto funzionamento di un sistema senza la necessità di conoscere tutti i dettagli interni del software.

- 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.
- Verifica che un evento passi attraverso tutti i servizi previsti end-to-end in un flusso di lavoro.

Nell'architettura tradizionale basata su server, i team spesso definiscono l'ambito di test andando a includere solo il codice che viene eseguito sul server applicativo. Altri componenti, servizi o dipendenze spesso sono considerati esterni e non possono essere testati.

Le applicazioni serverless abitualmente sono costituite da piccole unità di lavoro, come le funzioni Lambda che recuperano prodotti da un database, elaborano elementi da una coda o ridimensionano un'immagine nell'archiviazione. Ogni componente viene eseguito nel proprio ambiente. Probabilmente, i team saranno responsabili di molte di queste piccole unità all'interno di una singola applicazione.

Alcune funzionalità delle applicazioni possono essere delegate interamente a servizi gestiti come Amazon S3 o create senza utilizzare alcun codice sviluppato internamente. Non è necessario testare questi servizi gestiti, ma è necessario testare l'integrazione con questi servizi.

Come testare le soluzioni serverless

Probabilmente hai familiarità con i test delle applicazioni implementate in locale: si scrivono test che vengono eseguiti su codice completamente in esecuzione sul proprio sistema operativo desktop o all'interno di container. Ad esempio, è possibile richiamare un componente del servizio Web locale con una richiesta e quindi formulare affermazioni sulla risposta.

Le soluzioni serverless vengono create a partire dal codice funzionale e dai servizi gestiti basati sul cloud, come code, database, bus di eventi e sistemi di messaggistica. Questi componenti sono tutti collegati tramite un'architettura basata sugli eventi, in cui i messaggi, chiamati eventi, fluiscono da una risorsa all'altra. Queste interazioni possono essere sincrone, ad esempio quando un servizio Web restituisce immediatamente i risultati, o un'azione asincrona che viene completata in un secondo momento, come l'inserimento di elementi in una coda o l'avvio di una fase del flusso di lavoro. La tua strategia di test deve includere entrambi gli scenari e testare le interazioni tra i servizi. Per le interazioni asincrone, potrebbe essere necessario rilevare effetti indesiderati nei componenti a valle che potrebbero non essere immediatamente osservabili.

La replica di un intero ambiente cloud, tra cui code, tabelle di database, bus di eventi, policy di sicurezza e altro ancora, non è un metodo pratico. Incontrerai inevitabilmente problemi dovuti alle differenze tra il tuo ambiente locale e gli ambienti implementati nel cloud. Le discrepanze tra i tuoi ambienti aumenteranno il tempo necessario per riprodurre e correggere i bug.

Nelle applicazioni serverless, i componenti dell'architettura solitamente esistono interamente nel cloud, quindi per sviluppare funzionalità e correggere bug è necessario eseguire test su codice e servizi nel cloud.

Tecniche di test

Nella realtà, per aumentare la qualità delle tue soluzioni, la tua strategia di test includerà probabilmente un mix di tecniche diverse. Utilizzerai test interattivi rapidi per eseguire il debug delle funzioni nella console, test unitari automatici per verificare la logica aziendale isolata, verifiche delle chiamate a servizi esterni con mock e test occasionali su emulatori che imitano un servizio.

- Test nel cloud: consente di implementare l'infrastruttura e il codice per eseguire test utilizzando effettivamente servizi, policy di sicurezza, configurazioni e parametri specifici dell'infrastruttura. I test basati sul cloud 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. È possibile scegliere da una raccolta di esempi di eventi di test o creare un evento personalizzato per testare una funzione in modo isolato. Tramite la console puoi anche condividere gli eventi di test con il tuo team.

  Per automatizzare i test nel ciclo di vita dello sviluppo e della realizzazione, dovrai eseguire i test all'esterno della console. Per le strategie e le risorse di automazione, consulta le sezioni relative ai test per linguaggi specifici nella presente guida.
- Test con mock (chiamati anche fake): i mock sono oggetti all'interno del codice che simulano e sostituiscono un servizio esterno. I mock forniscono un comportamento predefinito per verificare le chiamate e i parametri del servizio. Un fake (o "falso") è un'implementazione fittizia che utilizza scorciatoie per semplificare o migliorare le prestazioni. Ad esempio, un oggetto di accesso ai dati falso potrebbe restituire dati da un datastore in memoria. I mock possono simulare e semplificare dipendenze complesse, ma possono anche portare a più mock per sostituire le dipendenze nidificate.
- Test con emulatori: è possibile configurare applicazioni (a volte di terze parti) per simulare un servizio cloud nel proprio ambiente locale. Il loro punto di forza è la velocità, tuttavia la configurazione e l'equivalenza con i servizi in fase di produzione sono il loro punto debole. Consigliamo di utilizzare gli emulatori con parsimonia.

Test nel cloud

I test nel cloud sono utili per tutte le fasi del test, inclusi test unitari, test di integrazione e test. end-to-end Quando esegui test su codice basato sul cloud che interagisce anche con i servizi basati sul cloud, ottieni la misura più accurata della qualità del tuo codice.

Un modo conveniente per eseguire una funzione Lambda nel cloud è con un evento di test nella AWS Management 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 anche condividerlo con il tuo team per rendere i test più semplici e coerenti.

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 le risorse basate sul cloud. Queste risorse sono state distribuite utilizzando lo strumento a riga di AWS SAM comando AWS Serverless Application Model and. Il codice di test recupera innanzitutto gli output dello stack implementato, che includono l'endpoint API, la funzione ARN e il ruolo di sicurezza. Successivamente, il test invia una richiesta all'endpoint API, che risponde con un elenco di bucket Amazon S3. Questo test viene eseguito interamente su risorse basate sul cloud per verificare che tali risorse siano implementate e protette e che funzionino come previsto.

```
========================= 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-west-2.amazonaws.com/Prod/hello/
        > API Gateway endpoint URL for Prod stage for Hello World function

        PythonTestDemo
        = arn:aws:lambda:us-west-2:1234567890:function:testing-apigw-lambda-
PythonTestDemo-iSij8evaTdxl
        > Hello World Lambda Function ARN

        PythonTestDemoIamRole
        = arn:aws:iam::1234567890: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-west-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 nel cloud aumentano le probabilità che il codice venga eseguito correttamente in produzione.

I test nel cloud presentano alcuni svantaggi. L'aspetto negativo più evidente dei test nel cloud è che le implementazioni in ambienti cloud richiedono in genere più tempo rispetto alle implementazioni in ambienti desktop locali.

Fortunatamente, strumenti come AWS Serverless Application Model (AWS SAM) Accelerate, la modalità watch AWS Cloud Development Kit (AWS CDK) e SST (di terze parti) riducono la latenza associata alle iterazioni di implementazione nel cloud. Questi strumenti possono monitorare l'infrastruttura e il codice e implementare automaticamente aggiornamenti incrementali nell'ambiente cloud.

> **Note**
>
> Scopri come creare un'infrastruttura come codice nella Serverless Developer Guide per saperne di più su, e. AWS Serverless Application Model AWS CloudFormation AWS Cloud Development Kit (AWS CDK)

A differenza dei test locali, i test nel cloud richiedono risorse aggiuntive che possono comportare costi di servizio. La creazione di ambienti di test isolati può aumentare il carico di lavoro per i DevOps team, specialmente nelle organizzazioni con controlli rigorosi su account e infrastruttura. Tuttavia, quando si lavora con scenari infrastrutturali complessi, il costo in termini di tempo degli sviluppatori per configurare e gestire un ambiente locale complesso potrebbe essere simile (o più alto) rispetto all'utilizzo di ambienti di test usa e getta creati con gli strumenti di automazione 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, puoi scrivere un test che utilizza un mock del servizio Amazon S3 che restituisce una risposta specifica ogni volta che viene chiamato CreateObjectil metodo. Quando viene eseguito un test, il mock restituisce quella risposta programmata senza chiamare Amazon S3 o altri endpoint del servizio.

Gli oggetti fittizi (mock) sono spesso generati da un framework fittizio per ridurre gli sforzi necessari per lo sviluppo. Alcuni framework fittizi sono generici e altri sono progettati specificamente per gli AWS SDK, come Moto, una libreria Python per simulare servizi e risorse. AWS

Ricorda che gli oggetti fittizi (mock) differiscono dagli emulatori in quanto i mock vengono generalmente creati o configurati da uno sviluppatore come parte del codice di test, mentre gli emulatori sono applicazioni autonome che espongono le funzionalità allo stesso modo dei sistemi che emulano.

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 diventa disponibile. AWS

I test con mock presentano i seguenti svantaggi:

- Generalmente, i mock richiedono un notevole impegno per l'impostazione e la configurazione, in particolare quando si cerca di determinare i valori restituiti da servizi diversi al fine di 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 limitazioni delle quote o delle policy di AWS Identity and Access Management (IAM). Sebbene i mock siano più efficaci per simulare quando l'autorizzazione viene negata o quando viene superata una quota, i test non possono determinare quale risultato si verificherà effettivamente in un ambiente di produzione.

Test con emulazione

Gli emulatori sono in genere un'applicazione eseguita localmente che imita un servizio di produzione. AWS

Gli emulatori dispongono di API simili alle loro controparti cloud e forniscono valori di ritorno simili. Possono anche simulare cambiamenti di stato avviati dalle chiamate API. Ad esempio, è possibile AWS SAM eseguire una funzione con AWS SAM local per emulare il servizio Lambda in modo da poter richiamare rapidamente una funzione. Per ulteriori informazioni, consulta la sezione AWS SAM locale nella Guida per gli sviluppatori di AWS Serverless Application Model .

I vantaggi dei test con gli emulatori sono molteplici:

- Gli emulatori possono facilitare iterazioni e test di sviluppo locale rapidi.
- 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 n-tier, potresti disporre di un motore di database e un server Web (simili a quelli in esecuzione in produzione) in esecuzione sul tuo computer locale per fornire una capacità di test rapida, locale e isolata.
- 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.

I test con emulatori presentano i seguenti svantaggi:

- Gli emulatori possono essere difficili da configurare e replicare, soprattutto 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 restituire un esito negativo nel cloud a causa delle policy di sicurezza in fase di produzione, delle configurazioni tra servizi o del superamento delle quote Lambda.
- Molti AWS servizi non dispongono di emulatori disponibili. È importante considerare che, se ci si affida all'emulazione, potrebbe non essere disponibile un'opzione di test soddisfacente per alcune parti dell'applicazione.

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 di test nel contesto del cloud controllerà in modo completo non solo la logica aziendale, ma anche le policy di sicurezza, le configurazioni dei servizi, le quote, i valori restituiti e le firme delle API più aggiornati.

Struttura il tuo codice per la testabilità

Semplifica i test e le funzioni Lambda separando il codice specifico di Lambda dalla logica aziendale principale.

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 specifici di Lambda. 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 sottoposto a 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 presenti 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.

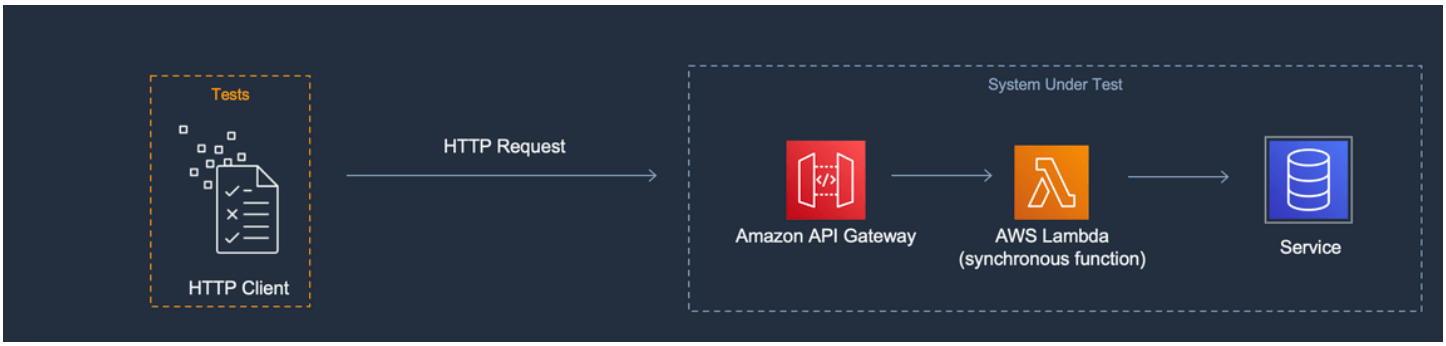
Concentrati sui test di integrazione

Quando si sviluppano applicazioni con Lambda, una buona pratica è quella di testare i componenti insieme.

I test eseguiti su due o più componenti architettonici sono chiamati test di integrazione. L'obiettivo dei test di integrazione è comprendere non solo come verrà eseguito il codice tra i componenti, ma anche come si comporterà l'ambiente che ospita il codice. nd-to-end I test E 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 pipeline CI/CD. 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, incluse le autorizzazioni IAM tra di loro. 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 Ciò eviterà i conflitti relativi alla denominazione delle risorse che possono verificarsi quando più sviluppatori lavorano a una base di codice condivisa, tentano di implementare risorse o richiamano 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 sam deploy o sam sync specifichino automaticamente uno stack con un prefisso univoco.

In alcuni casi, gli sviluppatori condividono un account. AWS Ciò può essere dovuto alla presenza di risorse nello stack che sono dispendiose in termini di gestione o di provisioning e configurazione. Ad esempio, un database può essere condiviso per semplificare la configurazione e il seeding dei dati in modo corretto.

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 del test, inclusi test unitari, test di integrazione e end-to-end test. 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.

Il vantaggio dei mock è minore per i test di integrazione, poiché il livello di impegno richiesto per implementare i mock necessari aumenta in proporzione al numero di punti di connessione. nd-to-end I test elettronici non dovrebbero utilizzare simulazioni, poiché questi test generalmente si occupano di stati e logiche complesse che non possono essere facilmente simulate 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, sarai in grado di creare e innovare con le funzionalità di servizio più recenti e le API aggiornate. Non rischierai di dover aspettare che i fornitori rilascino le versioni necessarie per raggiungere la parità di funzionalità. Ridurrai le spese iniziali e correnti per l'acquisto e la configurazione di più sistemi di sviluppo e macchine di compilazione. Inoltre, eviterai il problema che molti servizi cloud semplicemente non dispongono di emulatori. Pertanto, se la strategia di test dipende dall'emulazione, non si potranno utilizzare tali servizi (e, di conseguenza, occorreranno

soluzioni alternative potenzialmente più costose), oppure si produrranno codice e configurazioni non testati adeguatamente.

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 si utilizzano emulatori e chiamate fittizie per eseguire i test sul desktop locale, è possibile che si verifichino incongruenze nei test man mano che il codice passa da un ambiente all'altro nella pipeline CI/CD. I test unitari impiegati per convalidare la logica aziendale dell'applicazione sul desktop potrebbero non testare con precisione 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, è possibile chiamare l'CreateBucketAPI, ma è necessario tenere presente che l'identità che effettua la chiamata locale non avrà origine dal servizio Lambda. L'identità che effettua la chiamata non assumerà un ruolo di sicurezza come nel cloud, quindi verrà utilizzata invece un'autenticazione segnaposto, possibilmente con un ruolo o un'identità utente più permissivi che saranno diversi quando vengono eseguiti nel cloud.

Le configurazioni di mock ed emulazione verificheranno cosa farà la funzione Lambda se chiama Amazon S3; tuttavia, tali test non verificheranno 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, la funzione probabilmente genererà un errore qualora implementata 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 con mock sono generalmente impostati per eseguire direttamente il codice della funzione Lambda e per simulare l'integrazione con 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 con Amazon SQS verificheranno cosa farà la funzione Lambda quando viene chiamata da Amazon SQS con un determinato payload, ma non verificheranno che Amazon SQS richiami correttamente la funzione Lambda quando viene implementata 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 di 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 dal timeout e dalle impostazioni di memoria, il che potrebbe causare il fallimento della funzione se tali impostazioni non sono impostate correttamente. Le policy Lambda consentono anche 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 sono nel cloud e si connettono ad altre risorse, non si tratta di test di integrazione?

Definiamo test unitari quei test che operano su componenti architettonici isolati, ma ciò non impedisce di includere componenti che possono richiamare altri servizi o di 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 sul 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 policy IAM e le impostazioni di timeout e di 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 aggiornarti sulle ultime linee guida per i 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 di leggere anche i seguenti post del AWS blog:

- Accelerazione dello sviluppo serverless con AWS SAM Accelerate (AWS post sul blog)
- Aumentare la velocità di sviluppo con CDK Watch (AWS post sul blog)
- Simulazione delle integrazioni di 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

Compilazione di funzioni Lambda con Node.js

Puoi eseguire il JavaScript codice con Node.js in. AWS Lambda Lambda fornisce Runtime per Node.js che eseguono il tuo codice per elaborare gli eventi. Il codice viene eseguito in un ambiente che include AWS SDK for JavaScript, con le credenziali di un ruolo AWS Identity and Access Management (IAM) gestito dall'utente. Per ulteriori informazioni sulle versioni SDK incluse nei runtime di Node.js, consulta. the section called “Versioni SDK incluse in Runtime”

Lambda supporta i seguenti runtime di Node.js.

Node.js

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Node.js 20	`nodejs20.x`	Amazon Linux 2023			
Node.js 18	`nodejs18.x`	Amazon Linux 2			
Node.js 16	`nodejs16.x`	Amazon Linux 2	12 giugno 2024	28 febbraio 2025	31 marzo 2025

> **Note**
>
> I runtime di Node.js 18 e versioni successive utilizzano AWS SDK per la versione 3. JavaScript Per migrare una funzione da un runtime precedente, segui il workshop sulla migrazione su. GitHub Per ulteriori informazioni sull' AWS SDK per la JavaScript versione 3, consulta il post sul blog Modular AWS SDK for JavaScript is now general available.

Per creare una funzione Node.js.

1. Aprire la console Lambda.
2. Scegli Crea funzione.

3. Configurare le impostazioni seguenti:
   - Nome della funzione: inserisci il nome della funzione.
   - Runtime: scegli Node.js 20.x.
4. Scegli Crea funzione.
5. Per configurare un evento di test scegliere Test.
6. Per Event name (Nome evento) immettere **`test`**.
7. Seleziona Salvataggio delle modifiche.
8. Per invocare la funzione, scegliere Test (Testa).

La console crea una funzione Lambda con un singolo file di origine denominato `index.js` o `index.mjs`. È possibile modificare questo file e aggiungere altri file nell'editor di codice predefinito. Per salvare le modifiche, scegliere Save (Salva). Quindi, per eseguire il codice, scegliere Test (Testa).

> **Note**
>
> La console AWS Cloud9 Lambda fornisce un ambiente di sviluppo integrato nel browser. Puoi anche usarle AWS Cloud9 per sviluppare funzioni Lambda nel tuo ambiente. Per ulteriori informazioni, consulta Lavorare con AWS Lambda le funzioni utilizzando la Kit di strumenti AWS guida per l' AWS Cloud9 utente.

Il file `index.js` o `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 funzione Node.js 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.

> **Note**
>
> Per iniziare a sviluppare applicazioni nell'ambiente locale, implementate una delle applicazioni di esempio disponibili nell' GitHub archivio di questa guida.

Applicazioni Lambda di esempio in Node.js

- blank-nodejs — Una funzione Node.js 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.
- efs-nodejs - Una funzione che utilizza un file system Amazon EFS in un Amazon VPC. Questo esempio include un VPC, un file system, destinazioni di montaggio e un punto di accesso configurati per l'utilizzo con 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

Inizializzazione di Node.js

Node.js ha un modello di ciclo di eventi univoco che fa sì che il suo comportamento di inizializzazione sia diverso dagli altri tempi di esecuzione. In particolare, Node.js utilizza un modello di I/O senza blocchi che supporta operazioni asincrone. Questo modello consente a Node.js di funzionare in modo efficiente per la maggior parte dei carichi di lavoro. Ad esempio, se una funzione Node.js effettua una chiamata di rete, tale richiesta può essere designata come operazione asincrona e inserita in una coda di callback. La funzione può continuare a elaborare altre operazioni all'interno dello stack di chiamate principale senza essere bloccata in attesa che la chiamata di rete sia restituita. Una volta completata la chiamata di rete, viene eseguita la richiamata e quindi rimossa dalla coda di richiamata.

Alcune attività di inizializzazione possono essere eseguite in modo asincrono. L'esecuzione di queste attività asincrone potrebbe non essere completata prima di una chiamata. Ad esempio, il codice che effettua una chiamata di rete per recuperare un AWS parametro da Parameter Store potrebbe non essere completo nel momento in cui Lambda esegue la funzione di gestione. Di conseguenza, la variabile potrebbe essere null durante una chiamata. Per evitare ciò, assicurati che le variabili e altri codici asincroni siano completamente inizializzati prima di continuare con il resto della logica di business principale della funzione.

In alternativa, è possibile impostare il codice funzione come modulo ES, consentendo di utilizzare `await` al primo livello del file, al di fuori dell'ambito del gestore di funzioni. Quando `await` ogni `Promise`, il codice di inizializzazione asincrono viene completato prima delle chiamate del gestore, massimizzando l'efficacia della simultaneità con provisioning nella riduzione della latenza di avvio a freddo. Per ulteriori informazioni e un esempio, consulta Utilizzo di moduli ES Node.js e attesa di primo livello in AWS Lambda.

Impostazione di un gestore di funzioni come modulo ES

Per impostazione predefinita, Lambda tratta i file con il suffisso `.js` come moduli CommonJS. Facoltativamente, puoi designare il tuo codice come modulo ES. Ciò è possibile in due modi: specificando il `type` come `module` nel file `package.json` della funzione o utilizzando l'estensione del nome file `.mjs`. Nel primo scenario, il codice funzione tratta tutti i file `.js` come moduli ES, mentre nel secondo scenario solo il file specificato con `.mjs` è un modulo ES. È possibile combinare moduli ES e moduli CommonJS chiamandoli rispettivamente `.mjs` e `.cjs`, poiché i file `.mjs`sono sempre moduli ES e i file `.cjs` sono sempre moduli CommonJS.

Lambda cerca le cartelle nella variabile di `NODE_PATH` ambiente durante il caricamento dei moduli ES. Puoi caricare l' AWS SDK incluso nel runtime utilizzando le istruzioni del modulo ES. `import` È inoltre possibile caricare i moduli ES dai livelli.

Versioni SDK incluse in Runtime

La versione dell' AWS SDK inclusa nel runtime di Node.js dipende dalla versione di runtime e dalle tue. Regione AWS Per trovare la versione dell'SDK inclusa nel runtime che stai utilizzando, crea una funzione Lambda con il codice seguente.

> **Note**
>
> Il codice di esempio mostrato di seguito per le versioni 18 e successive di Node.js utilizza il formato CommonJS. Se crei la funzione nella console Lambda, assicurati di rinominare il file contenente il codice in. `index.js`

Example Node.js 18 e versioni successive

```
const { version } = require("@aws-sdk/client-s3/package.json");

exports.handler = async () => ({ version });
```

Ciò restituisce una risposta nel seguente formato:

```
{
  "version": "3.462.0"
}
```

Utilizzo di keep-alive per le connessioni TCP

L'agente HTTP/HTTPS Node.js predefinito crea una nuova connessione TCP per ogni nuova richiesta. Per evitare il costo della creazione di nuove connessioni, puoi `keepAlive: true` riutilizzare le connessioni che la tua funzione effettua utilizzando l' AWS SDK. JavaScript Il keep-alive può ridurre i tempi di richiesta per le funzioni Lambda che effettuano più chiamate API utilizzando l'SDK.

Nell' AWS SDK per JavaScript 3.x, incluso nei runtime Lambda `nodejs18.x` e successivi, keep-alive è abilitato per impostazione predefinita. Per disabilitare keep-alive, consulta Riutilizzo delle

connessioni con keep-alive in Node.js nella Guida per sviluppatori SDK per 3.x.AWS JavaScript Per ulteriori informazioni sull'utilizzo di keep-alive, consulta HTTP keep-alive è attivo di default nell'SDK modulare o sul blog Developer Tools. AWS JavaScript AWS

Caricamento di certificati CA

Per le versioni di runtime di Node.js fino a Node.js 18, Lambda carica automaticamente i certificati CA (autorità di certificazione) specifici di Amazon per semplificare la creazione di funzioni che interagiscono con altri 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 nella posizione `/var/runtime/ca-cert.pem`. Per ripristinare lo stesso comportamento da Node.js 18 e runtime precedenti, imposta la variabile di ambiente `NODE_EXTRA_CA_CERTS` su `/var/runtime/ca-cert.pem`.

Per prestazioni ottimali, consigliamo di raggruppare nel pacchetto di implementazione solo i certificati necessari e di caricarli tramite la variabile di ambiente `NODE_EXTRA_CA_CERTS`. 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`.

Definire il gestore della funzione 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.

La funzione di esempio seguente registra il contenuto dell'oggetto evento e restituisce la posizione dei registri.

> **Note**
>
> Questa pagina mostra esempi di gestori di moduli sia CommonJS sia ES. Per informazioni sulle differenze tra questi due tipi di gestori, consulta Impostazione di un gestore di funzioni come modulo ES.

ES module handler

Example

```
export const handler = async (event, context) => {
  console.log("EVENT: \n" + JSON.stringify(event, null, 2));
  return context.logStreamName;
};
```

CommonJS module handler

Example

```
exports.handler = async function (event, context) {
  console.log("EVENT: \n" + JSON.stringify(event, null, 2));
  return context.logStreamName;
};
```

Quando si configura una funzione, il valore dell'impostazione dell'handler è costituito dal nome del file e dal nome del metodo dell'handler esportato, separati da un punto. L'impostazione predefinita nella console e negli esempi in questa guida è `index.handler`. Questo indica il metodo `handler` che viene esportato dal file `index.js`.

Il runtime trasferisce gli argomenti al metodo del gestore. Il primo argomento è l'oggetto `event`, che contiene le informazioni sull'invoker. L'invoker trasferisce queste informazioni come stringa in formato JSON quando chiama Invoke e il runtime le converte in un oggetto. Quando un AWS servizio richiama la funzione, la struttura degli eventi varia in base al servizio.

Il secondo argomento è l'oggetto context, che contiene le informazioni sulla chiamata, la funzione, e l'ambiente di esecuzione. In questo esempio, la funzione ottiene il nome del flusso di log dall'oggetto context e lo restituisce all'invoker.

Inoltre puoi utilizzare un argomento di richiamo, ovvero una funzione che puoi richiamare nei gestori non asincroni per inviare una risposta. È consigliato l'utilizzo di async/await, anziché dei richiami. Async/Await fornisce una migliore leggibilità, gestione degli errori ed efficienza. Per ulteriori informazioni sulla differenza tra async/await e richiami, consulta Utilizzo dei richiami.

Denominazione

Quando si configura una funzione, il valore dell'impostazione dell'handler è costituito dal nome del file e dal nome del metodo dell'handler esportato, separati da un punto. L'impostazione predefinita per le funzioni create nella console e per gli esempi in questa guida è. `index.handler` Indica il `handler` metodo che viene esportato dal `index.mjs` file `index.js` or.

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. Selezionare Salva.

Utilizzo di async/await

Se il codice esegue un'attività asincrona, usa lo schema aync/await per assicurarti che il gestore termini l'esecuzione. Async/await è un modo conciso e leggibile per scrivere codice asincrono in Node.js, senza la necessità di richiami nidificati o di concatenamento di promesse. Con async/await puoi scrivere codice che si legga come codice sincrono, pur rimanendo asincrono e privo di blocchi.

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`.

> **Note**
>
> Assicurati di attendere il completamento degli eventi asincroni. Se la funzione restituisce un risultato prima del completamento degli eventi asincroni, potrebbe avere esito negativo o causare un comportamento imprevisto nell'applicazione. Ciò può accadere quando un ciclo `forEach` contiene un evento asincrono. I loop `forEach` prevedono una chiamata sincrona. Per ulteriori informazioni, consulta la sezione Array.prototype.forEach() nella documentazione di Mozilla.

ES module handler

Example — Richiesta HTTP con async/await

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

export const handler = async(event) => {
    try {
        // fetch is available in Node.js 18 and later runtimes
        const res = await fetch(url);
        console.info("status", res.status);
        return res.status;
    }
    catch (e) {
        console.error(e);
        return 500;
    }
};
```

CommonJS module handler

Example — Richiesta HTTP con async/await

```
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;
};
```

Nell'esempio successivo async/await è utilizzato per elencare i bucket Amazon Simple Storage Service.

> **Note**
>
> Prima di utilizzare questo esempio, assicurati che il ruolo di esecuzione della tua funzione disponga delle autorizzazioni di lettura di Amazon S3.

ES module handler

Example — AWS SDK v3 con async/await

Questo esempio utilizza la versione AWS SDK for JavaScript v3, disponibile nei runtime e nelle versioni successive. `nodejs18.x`

```
import {S3Client, ListBucketsCommand} from '@aws-sdk/client-s3';
const s3 = new S3Client({region: 'us-east-1'});

export const handler = async(event) => {
  const data = await s3.send(new ListBucketsCommand({}));
  return data.Buckets;

};
```

CommonJS module handler

Example — AWS SDK v3 con async/await

Questo esempio utilizza la versione AWS SDK for JavaScript v3, disponibile nei runtime e nelle versioni successive. `nodejs18.x`

```
const { S3Client, ListBucketsCommand } = require('@aws-sdk/client-s3');
const s3 = new S3Client({ region: 'us-east-1' });

exports.handler = async (event) => {
  const data = await s3.send(new ListBucketsCommand({}));
  return data.Buckets;
};
```

Utilizzo dei richiami

Per dichiarare il gestore della funzione è consigliato l'utilizzo di async/await, anziché di quello dei richiami. Async/await rappresenta una scelta migliore per diverse ragioni:

- Leggibilità: il codice async/await è più facile da leggere e comprendere rispetto al codice con richiami, che, invece, può di colpo diventare difficile da seguire e provocare problemi con i richiami.
- Debug e gestione degli errori: il debug del codice basato su richiami può essere difficile. La pila delle chiamate può diventare difficile da seguire e gli errori possono essere facilmente ingeriti. Con async/await, puoi utilizzare i blocchi try/catch per gestire gli errori.
- Efficienza: i richiami richiedono spesso il passaggio tra diverse parti del codice. Async/await può ridurre il numero dei passaggi di contesto, realizzando un codice più efficiente.

Quando utilizzi i richiami nel gestore, la funzione continua ad essere eseguita finché il ciclo di eventi è vuoto o la funzione va in timeout. 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 WaitsForEmptyEventcontext.callback Loop su false.

La funzione di callback richiede due argomenti, un `Error` e una risposta. L'oggetto risposta deve essere compatibile con `JSON.stringify`.

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

ES module handler

Example – richiesta HTTP con callback

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

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

CommonJS module handler

Example – richiesta HTTP con callback

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

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

Nell'esempio che segue, la risposta da Amazon S3 viene restituita all'invoker non appena disponibile. Il timeout in esecuzione sul ciclo di eventi viene bloccato e l'esecuzione continua la volta successiva in cui la funzione viene richiamata.

> **Note**
>
> Prima di utilizzare questo esempio, assicurati che il ruolo di esecuzione della tua funzione disponga delle autorizzazioni di lettura di Amazon S3.

ES module handler

Example — AWS SDK v3 con Loop callbackWaitsFor EmptyEvent

Questo esempio utilizza la versione AWS SDK for JavaScript v3, disponibile nei runtime `nodejs18.x` e nelle versioni successive.

```
import AWS from "@aws-sdk/client-s3";
const s3 = new AWS.S3({});

export const handler = function (event, context, callback) {
  context.callbackWaitsForEmptyEventLoop = false;
  s3.listBuckets({}, callback);
  setTimeout(function () {
    console.log("Timeout complete.");
  }, 5000);
};
```

CommonJS module handler

Example — AWS SDK v3 con Loop callbackWaitsFor EmptyEvent

Questo esempio utilizza la versione AWS SDK for JavaScript v3, disponibile nei runtime `nodejs18.x` e nelle versioni successive.

```
const AWS = require("@aws-sdk/client-s3");
const s3 = new AWS.S3({});

exports.handler = function (event, context, callback) {
  context.callbackWaitsForEmptyEventLoop = false;
  s3.listBuckets({}, callback);
  setTimeout(function () {
    console.log("Timeout complete.");
  }, 5000);
};
```

Distribuisci funzioni Lambda in Node.js con archivi di file .zip

Il codice della AWS Lambda funzione comprende un file .js o .mjs contenente il codice del gestore della funzione, insieme a tutti i 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 Node.js, consulta la pagina Implementazione di funzioni Lambda in Node.js 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 versione di Ubuntu e Bash integrata con Windows.

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

Per le funzioni Lambda che utilizzano il runtime di Node.js, una dipendenza può essere qualsiasi modulo Node.js. Il runtime di Node.js include una serie di librerie comuni, oltre a una versione di AWS SDK for JavaScript. Il runtime `nodejs16.x` Lambda include la versione 2.x dell'SDK. Le versioni di runtime `nodejs18.x` e successive includono la versione 3 dell'SDK. Per utilizzare la versione 2 dell'SDK con le versioni di runtime `nodejs18.x` e successive, aggiungi l'SDK al pacchetto di implementazione del file .zip. Se il runtime scelto include la versione dell'SDK che stai utilizzando,

non è necessario includere la libreria SDK nel tuo file .zip. Per scoprire quale versione dell'SDK è inclusa nel runtime che stai utilizzando, consulta. the section called “Versioni SDK incluse in Runtime”

Lambda aggiorna periodicamente le librerie SDK nel runtime di Node.js 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 versione particolare 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 delle funzioni Lambda di Node.js utilizzando file .zip.

Creazione di un pacchetto di implementazione .zip senza dipendenze

Se il codice della funzione non ha dipendenze oltre alle librerie incluse nel runtime Lambda, il file .zip contiene solo il file `index.js` o `index.mjs` 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 della funzione non si trova nella directory principale del file .zip, Lambda non è 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 Node.js 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 runtime Node.js di Lambda, 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. Per istruzioni sull'inclusione di dipendenze in un livello, consulta the section called “Creazione di un livello Node.js per 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 .
   ```

Creazione di un livello Node.js per 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”.

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 `PATH` variabile raccolga il contenuto del layer, il file Layer .zip deve avere le sue dipendenze nei seguenti percorsi di cartella:

- `nodejs/node_modules`
- `nodejs/node16/node_modules (NODE_PATH)`
- `nodejs/node18/node_modules (NODE_PATH)`
- `nodejs/node20/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 runtime di Node.js include una serie di librerie comuni, oltre a una versione di AWS SDK for 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 utilizzi un'istruzione `import` o `require` nel codice, il runtime di Node.js cerca nelle directory nel percorso `NODE_PATH` 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 tuo pacchetto di implementazione, questa versione avrà 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. Per 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 implementazione, il raggruppamento del codice comporta un miglioramento delle prestazioni perché si riducono le operazioni di I/O.

Creazione e aggiornamento delle funzioni Lambda di Node.js 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 AWS Command Line Interface e l'API Lambda. Puoi anche creare e aggiornare le funzioni Lambda usando AWS Serverless Application Model (AWS SAM) e AWS 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 Linux, Lambda richiede 644 permessi per i file non eseguibili (rw-r--r--) e 755 permessi () per le directory e i file eseguibili. rwxr-xr-x

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 eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 755 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta Set, View, Change, or Remove Permissions on an Object nella documentazione di Microsoft Windows.

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 AWS Management Console, consulta la Guida introduttiva ad Amazon S3. Per caricare file utilizzando la AWS CLI, consulta Move objects nella Guida per l'AWS CLI utente.

> **Note**
>
> Non è possibile modificare il tipo di pacchetto di distribuzione (.zip o immagine del contenitore) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine del contenitore 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. Selezionare 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, Node.js o Ruby)
- Il pacchetto di implementazione della funzione deve avere dimensioni inferiori a 3 MB.

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. Quando hai finito di modificare il codice, scegli Implementa per salvare le modifiche e aggiornare la 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. Usa la funzione create-function e update-function-codei comandi per distribuire 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 su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

> **Note**
>
> Se carichi il tuo file.zip da un bucket Amazon S3 utilizzando AWS CLI il, il bucket deve trovarsi nella stessa posizione della Regione AWS tua funzione.

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 nodejs20.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 nodejs20.x --handler index.handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
```

```
--code S3Bucket=DOC-EXAMPLE-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 DOC-EXAMPLE-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

Il 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 utilizzi 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 eventuali dipendenze specificate. Per ulteriori informazioni sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda, consulta la Guida introduttiva AWS Serverless Application Model alla AWS SAM Developer Guide.

È inoltre possibile utilizzare 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 tua macchina di compilazione. Per istruzioni su come caricare un file su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.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 del codice della funzione, sul percorso della cartella locale o sull'oggetto FunctionCode
- `Runtime`: imposta il runtime prescelto

Inoltre AWS SAM, se il tuo file.zip è più grande di 50 MB, non è necessario caricarlo prima in un bucket Amazon S3. AWS SAM puoi caricare pacchetti.zip fino alla dimensione massima consentita di 250 MB (decompressi) da una posizione sulla macchina di compilazione locale.

Per ulteriori informazioni sulla distribuzione delle funzioni utilizzando il file.zip in AWS SAM, consulta la Guida per gli sviluppatori. AWS::Serverless::FunctionAWS SAM

Creazione e aggiornamento di funzioni con file.zip utilizzando AWS CloudFormation

È possibile utilizzare AWS 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 su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

Nel AWS 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 che AWS CloudFormation genera non può superare i 4 MB. Per ulteriori informazioni sulla distribuzione delle funzioni utilizzando il file.zip in AWS CloudFormation, AWS::Lambda::Functionconsultate la Guida per l'utente.AWS CloudFormation

Distribuisci funzioni Lambda in Node.js con immagini di container

Esistono tre modi per creare un'immagine di container per una funzione Lambda in Node.js:

- Utilizzo di un'immagine di base per Node.js AWS

  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.
- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  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 Node.js nell'immagine.
- Utilizzo di un'immagine non di base AWS

  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 Node.js 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

- 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	Definizione come obsoleto
20	Node.js 20	Amazon Linux 2023	Dockerfile per Node.js 2.0 e versioni successive GitHub	
18	Node.js 18	Amazon Linux 2	Dockerfile per Node.js 18 in poi GitHub	
16	Node.js 16	Amazon Linux 2	Dockerfile per Node.js 16 in poi GitHub	12 giugno 2024

Repository Amazon ECR: gallery.ecr.aws/lambda/nodejs

Le immagini di base Node.js 20 e successive si basano sull'immagine minima del contenitore Amazon Linux 2023. Le immagini di base precedenti utilizzavano 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`.

Le immagini basate su AL2023 utilizzano `microdnf` (symlinked as `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 immagini basate su AL2023, 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 la sezione Introduzione al runtime di Amazon Linux 2023 AWS Lambda sul AWS Compute Blog.

> **Note**
>
> Per eseguire immagini basate su AL2023 localmente, incluso with AWS Serverless Application Model (AWS SAM), devi usare la versione Docker 20.10.10 o successiva.

Utilizzo di un'immagine di base per Node.js AWS

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- Docker (versione minima 20.10.10 per Node.js 20 e immagini di base successive)
- Node.js

Creazione di un'immagine da un'immagine di base

Per creare un'immagine del contenitore da un'immagine di AWS base per Node.js

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
cd example
```

2. Crea un nuovo progetto Node.js 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 for 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 variabile di `{LAMBDA_TASK_ROOT}` ambiente definita da Lambda.
- Imposta l'argomento `CMD` specificando il gestore della funzione Lambda.

Example Dockerfile

```
FROM public.ecr.aws/lambda/nodejs:20

# 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`.

```
docker build --platform linux/amd64 -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 comando seguente: `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"
   ```

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:
   - Sostituisci `docker-image:test` con il nome e il tag della tua immagine Docker.
   - 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 image in un digest di immagini specifico. Ciò significa che se punti il tag immagine utilizzato per distribuire 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.

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 di runtime Lambda, che gestisce l'interazione tra Lambda e il codice della funzione.

Installa il client di interfaccia di runtime per Node.js utilizzando il gestore di pacchetti npm:

```
npm install aws-lambda-ric
```

È inoltre possibile scaricare il client dell'interfaccia di runtime Node.js da GitHub. Il client di interfaccia di runtime attualmente supporta le seguenti versioni di Node.js:

- 14.x
- 16.x
- 18.x
- 20.x

L'esempio seguente dimostra come creare un'immagine contenitore per Node.js utilizzando un'immagine non di AWS base. Il Dockerfile di esempio utilizza l'immagine di base `buster`. Il Dockerfile include il client di interfaccia di runtime.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- Docker
- Node.js

Creazione di un'immagine da un'immagine di base alternativa

Per creare un'immagine del contenitore da un'immagine non di AWS base

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
cd example
```

2. Crea un nuovo progetto Node.js 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 `buster` 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. Nella prima fase viene creata un'immagine della build, che è un ambiente Node.js 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.

   Example Dockerfile

```
# Define custom function directory
```

```
ARG FUNCTION_DIR="/function"

FROM node:20-buster 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-buster-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`.

```
docker build --platform linux/amd64 -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. È possibile creare l'emulatore nell'immagine o utilizzare la seguente procedura per installarlo sul 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. Tieni 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 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:
   - Sostituisci `docker-image:test` con il nome e il tag della tua immagine Docker.
   - 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 image in un digest di immagini specifico. Ciò significa che se punti il tag immagine utilizzato per distribuire 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.

Oggetto contesto AWS Lambda in 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`: imposta su false per inviare immediatamente la risposta quando viene eseguito il callback anziché attendere che il ciclo di eventi Node.js sia vuoto. Se impostato su false, tutti gli eventi in sospeso rimarranno in esecuzione durante la successiva chiamata.

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
}
```

AWS Lambda registrazione delle funzioni in Node.js

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della 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 Utilizzo dei CloudWatch log di Amazon con AWS Lambda.

Questa pagina descrive come produrre un output di registro dal codice della funzione Lambda o accedere ai log utilizzando AWS Command Line Interface la console Lambda o la console. CloudWatch

Sections

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.

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: 200 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 runtime di Node.js registra le voci `START`, `END` e `REPORT` 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.
- 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 segmento X-Ray.

- Campionato – Per le richieste tracciate, il risultato del campionamento.

Puoi visualizzare i log nella console Lambda, nella CloudWatch console Logs o dalla riga di comando.

Utilizzo dei controlli di registrazione avanzati di Lambda con Node.js

Per avere 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 Node.js supportati:

- Formato di log: scegli tra i formati di testo normale e JSON strutturato per i log della funzione
- Livello di log: per i log in formato JSON, scegli il livello di dettaglio dei log che Lambda invia ad CloudWatch Amazon, come ERROR, DEBUG o INFO
- Gruppo di log: scegli il gruppo di log a cui la CloudWatch 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 la funzione Lambda”.

Per utilizzare le opzioni del formato di log e del livello di log con le funzioni Lambda in Node.js, consulta le istruzioni nelle sezioni seguenti.

Utilizzo di log JSON strutturati con Node.js

Se si seleziona JSON per il formato di registro della funzione, Lambda invierà l'output dei log utilizzando i metodi della console `console.trace` di`console.debug`,,, `console.log` `console.info`, `console.error`, `console.warn` e CloudWatch a 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 che sono già codificati in JSON, quindi i log delle applicazioni della funzione continueranno a essere acquisiti come prima.

Per ulteriori informazioni sull'utilizzo del pacchetto Powertools for AWS Lambda logging per creare log strutturati JSON nel runtime Node.js, vedere. the section called “Registrazione”

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 registro della funzione su JSON.

Il primo esempio utilizza il metodo `console.error` per generare una stringa semplice.

Example Codice di registrazione di Node.js

```
export const handler = async (event) => {
  console.error("This is a warning message");
  ...
}
```

Example Record di log JSON

```
{
    "timestamp":"2023-11-01T00:21:51.358Z",
    "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 CloudWatch ai log non è stringato.

Example Codice di registrazione di Node.js

```
export const handler = async (event) => {
  console.log({data: 12.3, flag: false});
  ...
}
```

Example Record di log JSON

```
{
    "timestamp": "2023-12-08T23:21:04.664Z",
    "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 Codice di registrazione di Node.js

```
export const handler = async (event) => {
  console.log('Some object - ', {data: 12.3, flag: false});
  ...
}
```

Example Record di log JSON

```
{
    "timestamp": "2023-12-08T23:21:04.664Z",
    "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 ai log utilizzando i metodi. CloudWatch `console` 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 Codice di registrazione di Node.js

```
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": "2023-12-08T23:21:04.632Z",
    "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": "2023-12-08T23:21:04.646Z",
    "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 per Node.js che è possibile utilizzare per creare log in formato metrico incorporato (EMF). 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 tuo codice.

Se attualmente il codice emette i log EMF direttamente utilizzando `console.log` o utilizzando Powertools for AWS Lambda (TypeScript), non CloudWatch sarà inoltre in grado di analizzarli se modificate il formato di registro della funzione in JSON.

> **Important**
>
> Per assicurarvi che i log EMF delle vostre funzioni continuino ad essere analizzati correttamente CloudWatch, aggiornate le librerie 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 parametri incorporati a **`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 filtraggio a livello di log con Node.js

Per AWS Lambda filtrare i log delle applicazioni in base al loro livello di registro, la funzione deve utilizzare log 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 di 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 log della funzione, consulta la pagina the section called “Configurazione dei controlli di registrazione avanzati per la funzione Lambda”.
- Utilizza un'altra libreria o metodo di registrazione per creare nel codice dei log JSON strutturati che includono una coppia chiave-valore "livello" che definisce il livello dell'output log. Ad esempio, puoi utilizzare Powertools per generare output AWS Lambda di log strutturati JSON dal tuo codice. Per ulteriori informazioni sull'utilizzo di Powertools con il runtime Node.js, consulta la pagina the section called “Registrazione”.

  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.

Quando configuri la tua funzione per utilizzare il filtraggio a livello di log, selezioni il livello di log che desideri inviare AWS Lambda a Logs tra le 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 funzionam ento della funzione
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'appl icazione

Lambda invia i log del livello selezionato e inferiore a. CloudWatch Ad esempio, se configuri un livello di log WARN, Lambda invierà i log corrispondenti ai livelli WARN, ERROR e FATAL.

Uso della 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 del codice incorporato, troverai i log nei risultati dell’esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l’output del log nella sezione Dettagli.

Utilizzo della 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. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegliere il gruppo di log della funzione (/aws/lambda/***function-name***).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Nuovi flussi di log vengono visualizzati quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire più 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 il flusso di log nella traccia.

AWS Command Line InterfaceAWS CLI Usando il ()

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- AWS CLI — Configurazione rapida con `aws configure`

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":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "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-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.

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 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.

```
#!/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 seguente comando 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.

Strumentazione del codice Node.js in AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, che possono includere funzioni Lambda e altri servizi AWS .

Per inviare dati di tracciamento a X-Ray, è possibile utilizzare una delle due librerie SDK:

- AWS Distro for OpenTelemetry (ADOT): una distribuzione sicura, pronta per la produzione e supportata dell'SDK (Otel AWS). OpenTelemetry
- SDK AWS X-Ray per Node.js: un SDK per generare e inviare i dati di traccia su X-Ray.

Ciascun SDK offre dei modi per inviare i dati di telemetria al servizio X-Ray. Puoi quindi utilizzare X-Ray per visualizzare, filtrare e analizzare le metriche delle prestazioni dell'applicazione per identificare i problemi e le opportunità di ottimizzazione.

> **Important**
>
> 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 end-to-end tracciamento in X-Ray utilizzando entrambe le soluzioni. Per ulteriori informazioni sulla scelta più adatta, consulta Scegliere tra le AWS Distro per OpenTelemetry e SDK X-Ray.

Sections

Utilizzo di ADOT per strumentare le 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 runtime Node.js, puoi aggiungere il livello Lambda gestito da AWS per ADOT Javascript per strumentare automaticamente le tue funzioni. Per istruzioni dettagliate su come aggiungere questo layer, consulta AWS Distro for OpenTelemetry Lambda JavaScript Support nella documentazione ADOT.

Utilizzo dell'SDK X-Ray per strumentare le 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-nodejs/package.json

```
{
  "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 strumentare i client AWS SDK nella versione AWS SDK for JavaScript v3, avvolgete l'istanza del client con il metodo. `captureAWSv3Client`

Example blank-nodejs/function/index.js — Tracciamento di un client AWS 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'SDK X-Ray. 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 “Configurazione delle variabili d'ambiente”.

Dopo aver aggiunto le dipendenze corrette e aver apportato le modifiche necessarie al codice, attivare 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. Scegli Modifica.
5. In X-Ray, attivare Active tracing (Tracciamento attivo).
6. Selezionare Salva.

Attivazione del tracciamento con l'API Lambda

Configura il tracciamento sulla tua funzione Lambda con AWS o SDK, utilizza AWS CLI le seguenti operazioni API:

- UpdateFunctionConfigurazione
- GetFunctionConfigurazione
- CreateFunction

Il AWS CLI comando di esempio seguente abilita il tracciamento attivo 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 AWS CloudFormation

Per attivare il tracciamento su una `AWS::Lambda::Function` risorsa in un AWS 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), utilizzate la `Tracing` proprietà.

Example template.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretazione di una traccia X-Ray

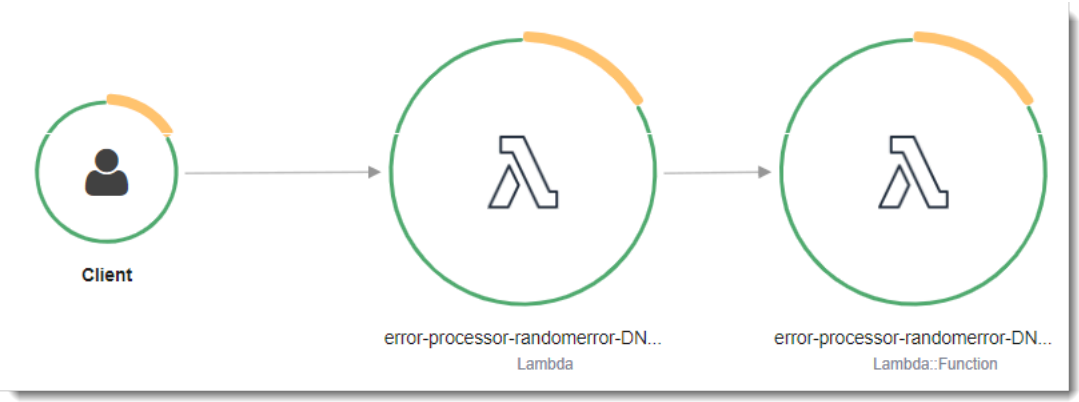
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al ruolo di esecuzione della funzione. Altrimenti, aggiungete la AWSXRayDaemonWriteAccesspolitica al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico dei servizi X-Ray mostra informazioni sull'applicazione e tutti i suoi componenti. L'immagine seguente 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 compaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon Logs. CloudWatch

X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste.

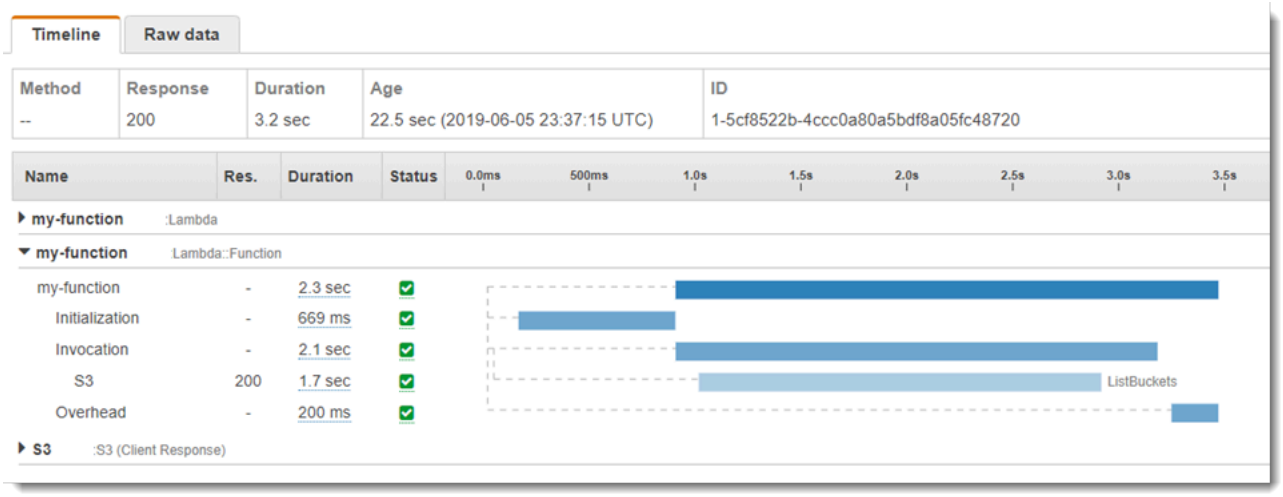
> **Note**
>
> La frequenza di campionamento di X-Ray non può essere configurata per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, il che crea 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 denominati my-function, ma uno ha un'origine di `AWS::Lambda` e l'altro ha un'origine di. `AWS::Lambda::Function` Se il `AWS::Lambda` segmento mostra un errore, il servizio Lambda presentava un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione presentava un problema.



Questo esempio espande il `AWS::Lambda::Function` segmento per mostrarne i tre 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.

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 tracciamento X-Ray gratuitamente ogni mese fino a un determinato limite come parte del AWS piano gratuito. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consultare Prezzi di AWS X-Ray.

Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray)

Se utilizzate X-Ray SDK per strumentare i client AWS SDK del codice della funzione, il pacchetto di distribuzione può diventare piuttosto grande. Per evitare di caricare dipendenze di runtime ogni volta che si aggiorna il codice della funzione, includere l'SDK X-Ray in un livello 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:
        - nodejs16.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 layer per le dipendenze richiede modifiche alla compilazione per generare l'archivio dei layer prima della distribuzione. Per un esempio funzionante, vedere l'applicazione di esempio blank-nodejs .

Creazione di funzioni Lambda con TypeScript

È possibile utilizzare il runtime Node.js per eseguire il TypeScript codice. AWS Lambda Poiché Node.js non esegue il TypeScript codice in modo nativo, è necessario prima traspilare il TypeScript codice in. JavaScript Quindi, usa i JavaScript file per distribuire il codice della funzione in Lambda. Il codice viene eseguito in un ambiente che include l' AWS SDK for JavaScript, con le credenziali di un ruolo AWS Identity and Access Management (IAM) che gestisci. Per ulteriori informazioni sulle versioni SDK incluse nei runtime di Node.js, consulta. the section called “Versioni SDK incluse in Runtime”

Lambda supporta i seguenti runtime di Node.js.

Node.js

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Node.js 20	`nodejs20.x`	Amazon Linux 2023			
Node.js 18	`nodejs18.x`	Amazon Linux 2			
Node.js 16	`nodejs16.x`	Amazon Linux 2	12 giugno 2024	28 febbraio 2025	31 marzo 2025

Argomenti

Configurazione di un ambiente di TypeScript sviluppo

Utilizzate un ambiente di sviluppo integrato locale (IDE), un editor di testo o AWS Cloud9scrivete il codice TypeScript della funzione. Non puoi creare TypeScript codice sulla console Lambda.

Per trascrivere il TypeScript codice, configura un compilatore come esbuild o il TypeScript compiler (`tsc`) di Microsoft, fornito in bundle con la distribuzione. TypeScript Puoi usare il AWS Serverless Application Model (AWS SAM) o il per semplificare la creazione e la distribuzione del codice AWS Cloud Development Kit (AWS CDK). TypeScript Entrambi gli strumenti utilizzano esbuild per trascrivere il codice TypeScript in. JavaScript

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 runtime di Node.js che si intende utilizzare. Per ulteriori informazioni, consulta Target nella documentazione esbuild. Per un esempio di file tsconfig.json che dimostra come indirizzare una versione specifica di Node.js supportata da Lambda, fai riferimento al 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 per non emettere file. `tsc` JavaScript 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"]
}
```

Definisci 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.

Example TypeScript gestore

La seguente funzione di esempio registra il contenuto dell'oggetto evento e restituisce la posizione dei log. Tieni presente quanto segue:

- Prima di utilizzare questo codice in una funzione Lambda, devi aggiungere il pacchetto @types/aws-lambda come dipendenza di sviluppo. Questo pacchetto contiene le definizioni dei tipi per Lambda. Quando è installato `@types/aws-lambda`, l'istruzione `import` (`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
- Il gestore in questo esempio è un modulo ES e deve essere designato come tale nel file `package.json` o utilizzando l'estensione del file `.mjs`. Per ulteriori informazioni, consulta le Impostazione di un gestore di funzioni come modulo ES.

```
import { Handler } from 'aws-lambda';

export const handler: Handler = async (event, context) => {
    console.log('EVENT: \n' + JSON.stringify(event, null, 2));
    return context.logStreamName;
};
```

Il runtime trasferisce gli argomenti al metodo del gestore. Il primo argomento è l'oggetto `event`, che contiene le informazioni sull'invoker. L'invoker trasferisce queste informazioni come stringa in formato JSON quando chiama Invoke e il runtime le converte in un oggetto. Quando un AWS servizio richiama la tua funzione, la struttura degli eventi varia in base al servizio. Con TypeScript, consigliamo di utilizzare annotazioni di tipo per l'oggetto evento. Per ulteriori informazioni, consulta Utilizzo di tipi per l'oggetto evento.

Il secondo argomento è l'oggetto context, che contiene le informazioni sulla chiamata, la funzione, e l'ambiente di esecuzione. In questo esempio, la funzione ottiene il nome del flusso di log dall'oggetto context e lo restituisce all'invoker.

Inoltre puoi utilizzare un argomento di richiamo, ovvero una funzione che puoi richiamare nei gestori non asincroni per inviare una risposta. È consigliato l'utilizzo di async/await, anziché dei richiami. Async/Await fornisce una migliore leggibilità, gestione degli errori ed efficienza. Per ulteriori informazioni sulla differenza tra async/await e richiami, consulta Utilizzo dei richiami.

Utilizzo di async/await

Se il codice esegue un'attività asincrona, usa lo schema aync/await per assicurarti che il gestore termini l'esecuzione. Async/await è un modo conciso e leggibile per scrivere codice asincrono in Node.js, senza la necessità di richiami nidificati o di concatenamento di promesse. Con async/await puoi scrivere codice che si legga come codice sincrono, pur rimanendo asincrono e privo di blocchi.

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`.

Example TypeScript funzione: asincrona

In questo esempio viene utilizzato `fetch`, che è disponibile nel runtime di `nodejs18.x`. Tieni presente quanto segue:

- Prima di utilizzare questo codice in una funzione Lambda, devi aggiungere il pacchetto @types/aws-lambda come dipendenza di sviluppo. Questo pacchetto contiene le definizioni dei tipi per Lambda. Quando è installato `@types/aws-lambda`, l'istruzione `import` (`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
- Il gestore in questo esempio è un modulo ES e deve essere designato come tale nel file `package.json` o utilizzando l'estensione del file `.mjs`. Per ulteriori informazioni, consulta le Impostazione di un gestore di funzioni come modulo ES.

```
import { APIGatewayProxyEvent, APIGatewayProxyResult } from 'aws-lambda';
const url = 'https://aws.amazon.com/';
export const lambdaHandler = async (event: APIGatewayProxyEvent):
 Promise<APIGatewayProxyResult> => {
    try {
        // fetch is available with Node.js 18
        const res = await fetch(url);
        return {
            statusCode: res.status,
```

```
            body: JSON.stringify({
                message: await res.text(),
            }),
        };
    } catch (err) {
        console.log(err);
        return {
            statusCode: 500,
            body: JSON.stringify({
                message: 'some error happened',
            }),
        };
    }
};
```

Utilizzo dei richiami

Per dichiarare il gestore della funzione è consigliato l'utilizzo di async/await, anziché di quello dei richiami. Async/await rappresenta una scelta migliore per diverse ragioni:

- Leggibilità: il codice async/await è più facile da leggere e comprendere rispetto al codice con richiami, che, invece, può di colpo diventare difficile da seguire e provocare problemi con i richiami.
- Debug e gestione degli errori: il debug del codice basato su richiami può essere difficile. La pila delle chiamate può diventare difficile da seguire e gli errori possono essere facilmente ingeriti. Con async/await, puoi utilizzare i blocchi try/catch per gestire gli errori.
- Efficienza: i richiami richiedono spesso il passaggio tra diverse parti del codice. Async/await può ridurre il numero dei passaggi di contesto, realizzando un codice più efficiente.

Quando utilizzi i richiami nel gestore, la funzione continua ad essere eseguita finché il ciclo di eventi è vuoto o la funzione va in timeout. 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. Puoi configurare il runtime per inviare immediatamente la risposta impostando WaitsForEmptyEventcontext.callback Loop su false.

La funzione di callback richiede due argomenti, un `Error` e una risposta. L'oggetto risposta deve essere compatibile con `JSON.stringify`.

Example TypeScript funzione con callback

Questa funzione di esempio riceve un evento da Gateway Amazon API, registra l'evento e gli oggetti di contesto e quindi restituisce una risposta a Gateway API. Tieni presente quanto segue:

- Prima di utilizzare questo codice in una funzione Lambda, devi aggiungere il pacchetto @types/aws-lambda come dipendenza di sviluppo. Questo pacchetto contiene le definizioni dei tipi per Lambda. Quando è installato `@types/aws-lambda`, l'istruzione `import` (`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
- Il gestore in questo esempio è un modulo ES e deve essere designato come tale nel file `package.json` o utilizzando l'estensione del file `.mjs`. Per ulteriori informazioni, consulta le Impostazione di un gestore di funzioni come modulo ES.

```
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 di tipi per l'oggetto evento

Consigliamo di non utilizzare il tipo any per gli argomenti del gestore e il tipo return perché si perde la possibilità di controllare i tipi. Genera invece un evento utilizzando il comando AWS Serverless Application Model CLI sam local generate-event o usa una definizione open source dal pacchetto @types /aws-lambda.

Generazione di un evento utilizzando il comando sam local generate-event

1. Generare un evento proxy Amazon Simple Storage Service (Amazon S3).

```
sam local generate-event s3 put >> S3PutEvent.json
```

2. Utilizzate l'utilità quicktype per generare definizioni dei tipi dal file S3 .json. PutEvent

```
npm install -g quicktype
quicktype S3PutEvent.json -o S3PutEvent.ts
```

3. Usare i tipi generati nel codice.

```
import { S3PutEvent } from './S3PutEvent';

export const lambdaHandler = async (event: S3PutEvent): Promise<void> => {
  event.Records.map((record) => console.log(record.s3.object.key));
};
```

Generazione di un evento utilizzando una definizione open-source dal pacchetto @types/aws-lambda

1. Aggiungere il pacchetto @types/aws-lambda come dipendenza dallo sviluppo.

```
npm install -D @types/aws-lambda
```

2. Usare i tipi nel codice.

```
import { S3Event } from "aws-lambda";

export const lambdaHandler = async (event: S3Event): Promise<void> => {
  event.Records.map((record) => console.log(record.s3.object.key));
};
```

Implementa codice trascritto TypeScript in Lambda con archivi di file.zip

Prima di poter distribuire il TypeScript codice inAWS Lambda, devi trasporlo in. JavaScript Questa pagina spiega tre modi per creare e distribuire TypeScript codice in Lambda con archivi di file.zip:

- Utilizzo di AWS Serverless Application Model (AWS SAM)
- Utilizzando AWS Cloud Development Kit (AWS CDK)
- Utilizzando AWS Command Line Interface(AWS CLI) e esbuild

AWS SAMe AWS CDK semplifica la creazione e l'implementazione di 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. AWS CDK consente di creare applicazioni affidabili, scalabili e convenienti nel cloud con la notevole potenza espressiva di un linguaggio di programmazione. AWS CDK è destinato a utenti AWS da moderatamente a altamente esperti. Sia gli che AWS CDK i AWS SAM usano esbuild per trascrivere il codice TypeScript in. JavaScript

Utilizzo AWS SAM per distribuire TypeScript codice in 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 SAMusa esbuild per creare le funzioni Lambda di Node.js TypeScript dal codice. il supporto di esbuild è attualmente in anteprima pubblica. Durante l'anteprima pubblica, il supporto di esbuild potrebbe essere soggetto a modifiche incompatibili con il passato.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2
- Versione della CLI AWS SAM 1.75 o successiva
- Node.js 18.x

Implementare un'applicazione AWS SAM di esempio

1. Inizializza l'applicazione utilizzando il modello Hello World. TypeScript

```
sam init --app-template hello-world-typescript --name sam-app --package-type Zip --runtime nodejs18.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 il TypeScript codice in 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 18.x
- Docker o esbuild

Implementare un'applicazione AWS CDK di esempio

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. Dovresti vedere 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'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. Apri hello-world.ts e aggiungi il seguente codice al file. Contiene il NodejsFunction costrutto, che crea la funzione Lambda, e il costrutto, che crea LambdaRestApi 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 costrutto `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 la sezione relativa alla configurazione di esbuild nella guida di riferimento dell'API AWS CDK.

7. Apri .ts. hello-world-stack 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. AWS CDK crea e impacchetta la funzione Lambda usando esbuild e quindi implementa la funzione al runtime di 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 TypeScript codice in Lambda

L'esempio seguente dimostra come traspilare e distribuire codice in TypeScript 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 18.x
- 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 progetto Node.js con npm o un gestore di pacchetti di 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. 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 risorse API Gateway.

> **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',
      }),
   };
};
```

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 "nodejs18.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 della funzione

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- Docker
- Node.js 18.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.

Example Dockerfile

Il seguente Dockerfile 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.

```
FROM public.ecr.aws/lambda/nodejs:18 as builder
WORKDIR /usr/app
COPY package.json index.ts  ./
RUN npm install
RUN npm run build

FROM public.ecr.aws/lambda/nodejs:18
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`.

```
docker build --platform linux/amd64 -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:
   - Sostituisci `docker-image:test` con il nome e il tag della tua immagine Docker.
   - 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 image in un digest di immagini specifico. Ciò significa che se punti il tag immagine utilizzato per distribuire 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.

AWS Lambdaoggetto di contesto in TypeScript

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`: imposta su false per inviare immediatamente la risposta quando viene eseguito il callback anziché attendere che il ciclo di eventi Node.js sia vuoto. Se impostato su false, tutti gli eventi in sospeso rimarranno in esecuzione durante la successiva chiamata.

Puoi usare il pacchetto npm @types/aws-lambda per lavorare con l'oggetto di contesto.

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 come dipendenza di sviluppo. Questo pacchetto contiene le definizioni dei tipi per Lambda. Quando è installato `@types/aws-lambda`, l'istruzione `import` (`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

```
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;
};
```

AWS Lambda funzione di accesso TypeScript

AWS Lambda monitora automaticamente le funzioni Lambda e invia le voci di registro ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della 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 registri, consulta. Utilizzo dei CloudWatch log di Amazon con AWS Lambda

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

- Strumenti e librerie
- Utilizzo di Powertools per AWS Lambda () e per la registrazione TypeScript strutturata AWS SAM
- Utilizzo di Powertools for AWS Lambda (TypeScript) e the AWS CDK per la registrazione strutturata
- Uso della console Lambda
- Utilizzo della console CloudWatch

Strumenti 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, 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 () e per la registrazione TypeScript strutturata AWS SAM

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 back-end 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:

- Node.js 18.x 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, consulta Aggiornamento della CLI. AWS SAM

Distribuisci un'applicazione di esempio AWS SAM

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 nodejs18.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 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 <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:

```
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.552000
 START RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb Version: $LATEST
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.594000
 2022-08-31T09:33:10.557Z 70693159-7e94-4102-a2af-98a6343fb8fb
 INFO {"_aws":{"Timestamp":1661938390556,"CloudWatchMetrics":
[{"Namespace":"sam-app","Dimensions":[["service"]],"Metrics":
[{"Name":"ColdStart","Unit":"Count"}]}]},"service":"helloWorld","ColdStart":1}
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.595000
 2022-08-31T09:33:10.595Z 70693159-7e94-4102-a2af-98a6343fb8fb INFO
 {"level":"INFO","message":"This is an INFO log - sending HTTP 200 - hello world
 response","service":"helloWorld","timestamp":"2022-08-31T09:33:10.594Z"}
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.655000
 2022-08-31T09:33:10.655Z 70693159-7e94-4102-a2af-98a6343fb8fb INFO
```

```
 {"_aws":{"Timestamp":1661938390655,"CloudWatchMetrics":[{"Namespace":"sam-
app","Dimensions":[["service"]],"Metrics":[]}]},"service":"helloWorld"}
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.754000 END
 RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.754000
 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 configurare la conservazione dei log, aggiungi quanto segue al tuo modello: AWS SAM

```
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 for AWS Lambda (TypeScript) e the 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. AWS CDK

Questa applicazione implementa un back-end 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:

- Node.js 18.x 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, consulta Aggiornamento della CLI. AWS SAM

Distribuisci un'applicazione di esempio AWS CDK

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 costrutto 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 costrutto, 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:

```
2023/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2022-08-31T14:48:37.047000
 START RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c Version: $LATEST
2023/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2022-08-31T14:48:37.050000 {
"level": "INFO",
"message": "This is an INFO log - sending HTTP 200 - hello world response",
"service": "helloWorld",
"timestamp": "2022-08-31T14:48:37.048Z",
"xray_trace_id": "1-630f74c4-2b080cf77680a04f2362bcf2"
}
2023/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2022-08-31T14:48:37.082000 END
 RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c
2023/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2022-08-31T14:48:37.082000
 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
```

Uso della 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 del codice incorporato, troverai i log nei risultati dell’esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l’output del log nella sezione Dettagli.

Utilizzo della 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. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegliere il gruppo di log della funzione (/aws/lambda/***function-name***).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Nuovi flussi di log vengono visualizzati quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire più 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 il 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 utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, che possono includere funzioni Lambda e altri servizi AWS.

Per inviare dati di tracciamento a X-Ray, è possibile utilizzare una delle tre librerie SDK:

- AWSDistro for OpenTelemetry (ADOT): una distribuzione sicura, pronta per la produzione e supportata dell'SDK (OtelAWS). OpenTelemetry
- SDK AWS X-Ray per Node.js: un SDK per generare e inviare i dati di traccia su X-Ray.
- Powertools for AWS Lambda (TypeScript): un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori.

Ciascun SDK offre dei modi per inviare i dati di telemetria al servizio X-Ray. Puoi quindi utilizzare X-Ray per visualizzare, filtrare e analizzare le metriche delle prestazioni dell'applicazione per identificare i problemi e le opportunità di ottimizzazione.

> **Important**
>
> Gli SDK X-Ray e Powertools per AWS Lambda 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 end-to-end tracciamento in X-Ray utilizzando entrambe le soluzioni. Per ulteriori informazioni sulla scelta più adatta, consulta Scegliere tra le AWS Distro per OpenTelemetry e SDK X-Ray.

Sections

Utilizzo di Powertools per AWS Lambda (TypeScript) e per il tracciamento AWS SAM

Segui i passaggi seguenti per scaricare, creare e distribuire un' TypeScript applicazione Hello World di esempio con i moduli Powertools for AWS Lambda (TypeScript) integrati utilizzando il. AWS SAM Questa applicazione implementa un back-end 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:

- Node.js 18.x o versione successiva
- AWS CLI versione 2
- CLI AWS SAM versione 1.75 o successiva. Se disponi di una versione precedente della CLI AWS SAM, consulta Aggiornamento della CLI AWS SAM.

Implementare un'applicazione AWS SAM di esempio

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 nodejs18.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 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 <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 nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste.

> **Note**
>
> La frequenza di campionamento di X-Ray non può essere configurata per le funzioni.

Utilizzo di Powertools for AWS Lambda (TypeScript) e the 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 back-end 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:

- Node.js 18.x o versione successiva
- AWS CLI versione 2
- AWS CDK versione 2
- CLI AWS SAM versione 1.75 o successiva. Se disponi di una versione precedente della CLI AWS SAM, consulta Aggiornamento della CLI AWS SAM.

Implementare un'applicazione AWS Cloud Development Kit (AWS CDK) di esempio

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. Dovresti vedere un file chiamato .ts. hello-world-stack 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 costrutto 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 costrutto, 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:

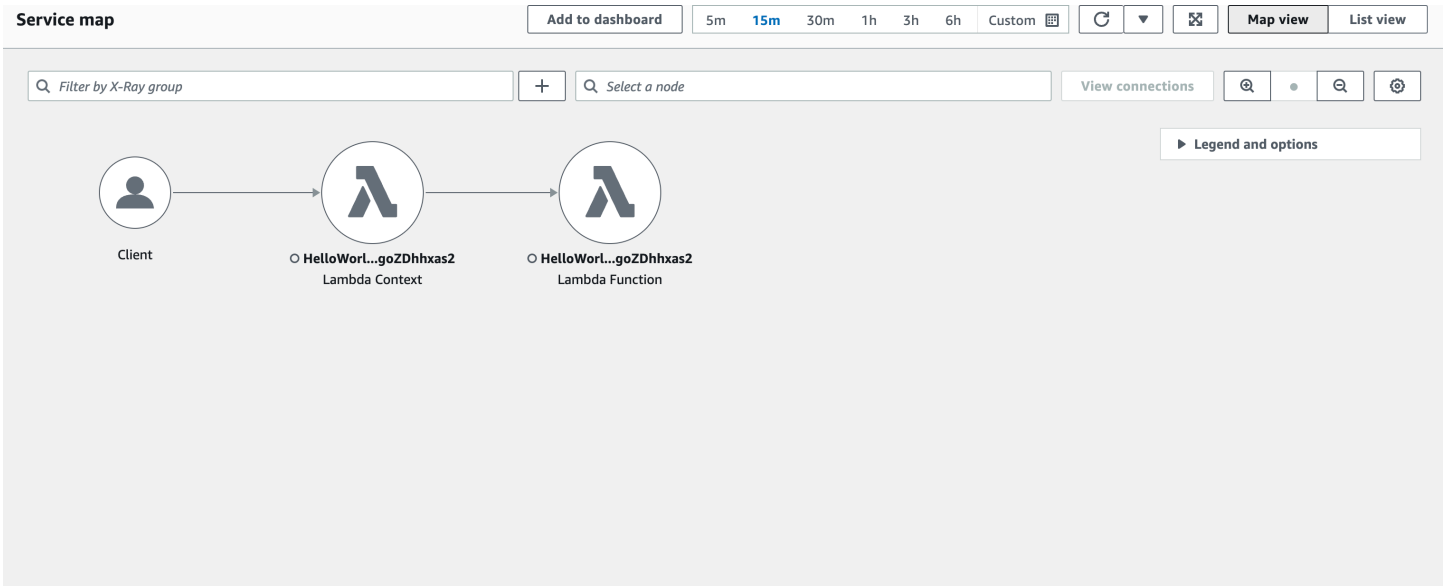
```
XRay Event [revision 1] at (2023-01-31T11:50:06.997000) with id
 (1-11a2222-111a222222cb33de3b95daf9) and duration (0.449s)
  - 0.350s - HelloWorldStack-helloworldfunction111A2BCD-Xyzv11a1bcde [HTTP: 200]
  - 0.157s - HelloWorldStack-helloworldfunction111A2BCD-Xyzv11a1bcde
    - 0.169s - Initialization
    - 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 di una traccia X-Ray

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. La mappa del tracciamento X-Ray fornisce informazioni sull'applicazione e su tutti i relativi 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 for Python (Boto3), con le credenziali AWS Identity and Access Management di un ruolo (IAM) che gestisci. Per ulteriori informazioni sulle versioni SDK incluse nei runtime Python, consulta. the section called “Versioni SDK incluse in runtime”

Lambda supporta i seguenti runtime di Python.

Python

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Python 3.12	`python3.12`	Amazon Linux 2023			
Python 3.11	`python3.11`	Amazon Linux 2			
Python 3.10	`python3.10`	Amazon Linux 2			
Python 3.9	`python3.9`	Amazon Linux 2			
Python 3.8	`python3.8`	Amazon Linux 2	14 ottobre 2024	28 febbraio 2025	31 marzo 2025

> **Note**
>
> Le informazioni sul runtime in questa tabella sono sottoposte ad aggiornamenti continui. Per ulteriori informazioni sull'utilizzo degli AWS SDK in Lambda, consulta AWS Gestione degli SDK nelle funzioni Lambda in Serverless Land.

Per creare una funzione Python

1. Aprire la console Lambda.
2. Scegli Crea funzione.
3. Configurare le impostazioni seguenti:
   - Nome della funzione: inserisci il nome della funzione.
   - Runtime: scegli Python 3.12.
4. Scegli Crea funzione.
5. Per configurare un evento di test scegliere Test.
6. Per Event name (Nome evento) immettere **`test`**.
7. Seleziona Salvataggio delle modifiche.
8. Per invocare la funzione, scegliere Test (Testa).

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. Per salvare le modifiche, scegliere Save (Salva). Quindi, per eseguire il codice, scegliere Test (Testa).

> **Note**
>
> La console AWS Cloud9 Lambda fornisce un ambiente di sviluppo integrato nel browser. Puoi anche usarle AWS Cloud9 per sviluppare funzioni Lambda nel tuo ambiente. Per ulteriori informazioni, consulta Lavorare con AWS Lambda le funzioni utilizzando la Kit di strumenti AWS guida per l' AWS Cloud9 utente.

> **Note**
>
> Per iniziare a sviluppare applicazioni nell'ambiente locale, implementate una delle applicazioni di esempio disponibili nell' GitHub archivio di questa guida.
>
> 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

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

Versioni SDK incluse in runtime

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 inclusa 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__}')
```

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, 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 Ambiente di esecuzione Lambda. Per esempi di come utilizzare Graceful Shutdown con le estensioni, consulta il repository Samples.AWS GitHub

Definisci il gestore di funzioni 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.

È possibile utilizzare la seguente sintassi generale quando si crea un gestore di funzioni in Python:

```
def handler_name(event, context):
    ...
    return some_value
```

Denominazione

Il nome del gestore di funzioni Lambda specificato al momento della creazione di una funzione Lambda deriva da quanto segue:

- il nome del file in cui si trova la funzione del gestore Lambda.
- il nome della funzione del gestore Python.

Un gestore di funzioni può essere qualsiasi nome; tuttavia, il nome predefinito sulla console Lambda è `lambda_function.lambda_handler`. Questo nome del gestore funzioni riflette il nome della funzione (`lambda_handler`) e il file in cui è memorizzato il codice del gestore è memorizzato (`lambda_function.py`).

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. Selezionare Salva.

Come funziona

Quando il gestore di funzioni viene richiamato da Lambda, il Runtime Lambda passa due argomenti al gestore di funzioni:

- Il primo argomento è l'oggetto evento. Un evento è un documento in formato JSON formattato che contiene i dati che una funzione Lambda deve elaborare. Il runtime Lambda converte l'evento in un oggetto e lo passa al codice della funzione. Di solito è del tipo Python `dict`. Può essere anche di tipo `list`, `str`, `int`, `float` o `NoneType`.

  L'oggetto evento contiene informazioni dal servizio di invocazione. Quando si invoca una funzione, si determina la struttura e il contenuto dell'evento. Quando un AWS servizio richiama la tua funzione, definisce la struttura dell'evento. Per ulteriori informazioni sugli eventi generati dai AWS servizi, vedere. Richiamare Lambda con eventi di altri servizi AWS
- Il secondo argomento è l'oggetto contesto. Un oggetto contesto viene passato alla funzione da Lambda in runtime. Questo oggetto fornisce i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di runtime.

Restituzione di un valore

È inoltre possibile che il gestore restituisca un valore. Ciò che accade al valore restituito dipende dal tipo di chiamata e dal servizio che ha invocato la funzione. Per esempio:

- Se si utilizza il tipo di `RequestResponse` invocazione, ad esempioInvocazione sincrona, AWS Lambda restituisce il risultato della chiamata della funzione Python al client che richiama la funzione Lambda (nella risposta HTTP alla richiesta di chiamata, serializzata in JSON). Ad esempio, la console AWS Lambda utilizza il tipo di invocazione `RequestResponse`; quindi, quando si invoca la funzione mediante la console, in quest'ultima verrà visualizzato 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.

> **Note**
>
> In Python 3.9 e nelle versioni successive, Lambda include il requestId dell'invocazione nella risposta di errore.

Esempi

La sezione seguente mostra esempi di funzioni Python che è possibile utilizzare con Lambda. Se si utilizza la console Lambda per creare la funzione, non è necessario allegare un file di archivio .zip per eseguire le funzioni di questa sezione. Queste funzioni utilizzano le librerie Python standard che sono incluse nel runtime Lambda selezionato. Per ulteriori informazioni, consulta Pacchetti di implementazione Lambda.

Restituzione di un messaggio

L'esempio seguente mostra una funzione denominata `lambda_handler`. La funzione accetta l'input dell'utente di un nome e cognome e restituisce un messaggio contenente i dati dell'evento ricevuto come input.

```
def lambda_handler(event, context):
    message = 'Hello {} {}!'.format(event['first_name'], event['last_name'])
    return {
        'message' : message
    }
```

È possibile utilizzare i seguenti dati evento per invocare la funzione:

```
{
    "first_name": "John",
    "last_name": "Smith"
}
```

La risposta mostra i dati dell'evento passati come input:

```
{
    "message": "Hello John Smith!"
}
```

Analisi di una risposta

L'esempio seguente mostra una funzione denominata `lambda_handler`. La funzione utilizza i dati degli eventi passati da Lambda al runtime. Analizza la variabile di ambiente in `AWS_REGION` restituita nella risposta JSON.

```
import os
import json

def lambda_handler(event, context):
    json_region = os.environ['AWS_REGION']
    return {
        "statusCode": 200,
        "headers": {
            "Content-Type": "application/json"
        },
        "body": json.dumps({
            "Region ": json_region
        })
    }
```

È possibile utilizzare qualsiasi dato evento per invocare la funzione:

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

I runtime Lambda impostano diverse variabili di ambiente durante l'inizializzazione. Per ulteriori informazioni sulle variabili di ambiente restituite nella risposta al runtime, consulta Usa le variabili di ambiente Lambda per configurare i valori nel codice.

La funzione in questo esempio dipende da una risposta riuscita (in `200`) dall'API Invoke. Per ulteriori informazioni sullo stato dell'API Invoke, consulta Sintassi della risposta di Invoke.

Rimozione di un calcolo

L'esempio seguente mostra una funzione denominata `lambda_handler`. La funzione accetta l'input dell'utente e restituisce un calcolo all'utente. Per ulteriori informazioni su questo esempio, consultate il repository. aws-doc-sdk-examples GitHub

```
import logging

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

def lambda_handler(event, context):
    ...
    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
```

È possibile utilizzare i seguenti dati evento per invocare la funzione:

```
{
    "action": "increment",
    "number": 3
}
```

Utilizzo di archivi di file .zip per le funzioni Lambda in Python

Il codice della AWS Lambda funzione comprende un file.py contenente il codice del gestore della funzione, insieme a tutti i pacchetti e i 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 versione di Ubuntu e Bash integrata con Windows.

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

Per le funzioni Lambda che utilizzano il runtime Python, una dipendenza può essere qualsiasi pacchetto o modulo Python. Quando si distribuisce la funzione utilizzando un archivio.zip, è possibile aggiungere queste dipendenze al file.zip con il codice della funzione o utilizzare un layer Lambda. Un livello è un file .zip separato che può contenere codice aggiuntivo o altri contenuti. Per saperne di più sull'utilizzo dei layer Lambda in Python, consulta. the section called “Livelli”

I runtime Lambda Python includono e le relative dipendenze. AWS SDK for 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 inline in AWS Serverless Application Model () o modelli.AWS SAM AWS 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, si consiglia di aggiungere tutte le dipendenze della funzione al pacchetto di implementazione, anche se le relative versioni sono incluse nel runtime Lambda. Ciò include l'SDK Boto3.

Per scoprire quale versione dell'SDK for Python (Boto3) è inclusa nel runtime che stai utilizzando, consulta. the section called “Versioni SDK incluse in 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 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 si trova nella directory principale 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 codice della funzione dipende da pacchetti o moduli aggiuntivi, puoi aggiungere queste dipendenze al tuo 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. 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 si trova nella directory principale del file .zip, Lambda non sarà in grado di 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$ python3.12 -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 può essere denominata `site-packages` oppure `dist-packages`. Questa cartella può trovarsi nella directory `lib/python3.x` oppure `lib64/python3.x` (dove python3.x rappresenta la versione di Python che stai utilizzando).

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/python3.12/site-packages`.

```
~/my_function$ cd my_virtual_env/lib/python3.12/site-packages
~/my_function/my_virtual_env/lib/python3.12/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/python3.12/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 tuo pacchetto di implementazione, questa versione avrà 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 e nell'immagine di base di Python 3.11, l' AWS SDK e le sue 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 rispettive versioni sono già 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 librerie Python popolari, tra cui NumPy e Pandas, non sono Python puro e contengono codice scritto in C o C++. Quando aggiungi librerie contenenti codice C/C++ al pacchetto di implementazione, devi creare il pacchetto correttamente per assicurarti che sia compatibile con l'ambiente di esecuzione Lambda.

La maggior parte dei pacchetti disponibili nel Python Package Index (PyPI) sono disponibili come "wheel" (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 implementazione compatibile con Lambda, è necessario installare il wheel per i sistemi operativi Linux e l'architettura del set di istruzioni della funzione.

Alcuni pacchetti possono essere disponibili solo come distribuzioni di origine. Per questi pacchetti, è necessario compilare e creare personalmente 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 un wheel compatibile con Lambda, utilizza l'opzione `--platform` in 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 di origine, la creazione delle librerie C/C++ spetta a te. Per rendere il pacchetto compatibile con l'ambiente di esecuzione Lambda, devi crearlo in un ambiente che utilizza lo stesso sistema operativo Amazon Linux 2. Puoi farlo creando il tuo pacchetto in un'istanza Linux di Amazon EC2.

Per ulteriori informazioni sull'avvio e la connessione a un'istanza Linux di Amazon EC2, consulta il Tutorial: Nozioni di base sulle istanze Linux di Amazon EC2 nella Guida per l'utente di Amazon EC2 per le istanze Linux.

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 AWS Command Line Interface e l'API Lambda. Puoi anche creare e aggiornare le funzioni Lambda usando AWS Serverless Application Model (AWS SAM) e AWS 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 eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 755 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta Set, View, Change, or Remove Permissions on an Object nella documentazione di Microsoft Windows.

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 AWS Management Console, consulta la Guida introduttiva ad Amazon S3. Per caricare file utilizzando la AWS CLI, consulta Move objects nella Guida per l'AWS CLI utente.

> **Note**
>
> Non è possibile modificare il tipo di pacchetto di distribuzione (.zip o immagine del contenitore) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine del contenitore 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. Selezionare 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, Node.js o Ruby)
- Il pacchetto di implementazione della funzione deve avere dimensioni inferiori a 3 MB.

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. Quando hai finito di modificare il codice, scegli Implementa per salvare le modifiche e aggiornare la funzione.

Creazione e aggiornamento di funzioni con file.zip utilizzando il 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 su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

> **Note**
>
> Se carichi il tuo file.zip da un bucket Amazon S3 utilizzando AWS CLI il, il bucket deve trovarsi nella stessa posizione della Regione AWS tua funzione.

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 python3.12 --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 python3.12 --handler lambda_function.lambda_handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--code S3Bucket=DOC-EXAMPLE-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 DOC-EXAMPLE-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
- UpdateFunctionCodice

Creazione e aggiornamento di funzioni con file.zip utilizzando AWS SAM

Il 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 utilizzi 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 eventuali dipendenze specificate. Per ulteriori informazioni sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda, consulta la Guida introduttiva AWS Serverless Application Model alla AWS SAM Developer Guide.

È inoltre possibile utilizzare 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 tua macchina di compilazione. Per istruzioni su come caricare un file su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.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 del codice della funzione, sul percorso della cartella locale o sull'oggetto FunctionCode
- `Runtime`: imposta il runtime prescelto

Inoltre AWS SAM, se il tuo file.zip è più grande di 50 MB, non è necessario caricarlo prima in un bucket Amazon S3. AWS SAM puoi caricare pacchetti.zip fino alla dimensione massima consentita di 250 MB (decompressi) da una posizione sulla macchina di compilazione locale.

Per ulteriori informazioni sulla distribuzione delle funzioni utilizzando il file.zip in AWS SAM, consulta la Guida per gli sviluppatori. AWS::Serverless::FunctionAWS SAM

Creazione e aggiornamento di funzioni con file.zip utilizzando AWS CloudFormation

È possibile utilizzare AWS 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 su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

Per i runtime di Node.js e Python, puoi anche fornire codice sorgente in linea nel tuo modello. AWS CloudFormation AWS CloudFormation quindi crea un file.zip contenente il codice quando crei la funzione.

Utilizzo di un file .zip esistente

Nel AWS 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

È possibile dichiarare semplici funzioni scritte in Python o Node.js in linea in un modello. AWS CloudFormation Poiché il codice è incorporato in YAML o JSON, non puoi aggiungere dipendenze esterne al tuo pacchetto di implementazione. Ciò significa che la 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 non cambia frequentemente.

Per creare un file .zip dal codice inline per i runtime Node.js e 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 AWS CloudFormation generato non può superare i 4 MB. Per ulteriori informazioni sulla distribuzione delle funzioni utilizzando il file.zip in AWS CloudFormation, AWS::Lambda::Functionconsultate la Guida per l'utente.AWS 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:

- Usare un'immagine AWS 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.
- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  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 per implementare un runtime personalizzato. Per rendere l'immagine compatibile con Lambda, devi includere il client di interfaccia di runtime per Python nell'immagine.
- Utilizzo di un'immagine non di base AWS

  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

- Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

AWS immagini base per Python

AWS fornisce le seguenti immagini di base per Python:

Tag	Runtime	Sistema operativo	Dockerfile	Definizione come obsoleto
3.12	Python 3.12	Amazon Linux 2023	Dockerfile per Python 3.12 e versioni successive GitHub	
3.11	Python 3.11	Amazon Linux 2	Dockerfile per Python 3.11 e versioni successive GitHub	
3.10	Python 3.10	Amazon Linux 2	Dockerfile per Python 3.10 e versioni successive GitHub	
3.9	Python 3.9	Amazon Linux 2	Dockerfile per Python 3.9 e versioni successive GitHub	
3.8	Python 3.8	Amazon Linux 2	Dockerfile per Python 3.8 e versioni successive GitHub	14 ottobre 2024

Repository Amazon ECR: gallery.ecr.aws/lambda/python

Le immagini di base di Python 3.12 e versioni successive si basano sull'immagine contenitore minima di Amazon Linux 2023. Le immagini di base di Python 3.8-3.11 sono basate sull'immagine Amazon Linux 2. Le immagini basate su AL2023 offrono diversi vantaggi rispetto ad Amazon Linux 2, tra cui un ingombro di implementazione ridotto e versioni aggiornate di librerie come. `glibc`

Le immagini basate su AL2023 utilizzano `microdnf` (symlinked as`dnf`) come gestore di pacchetti anzichéyum, 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 immagini basate su AL2023, 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 la sezione Introduzione al runtime di Amazon Linux 2023 AWS Lambda sul AWS Compute Blog.

> **Note**
>
> Per eseguire immagini basate su AL2023 localmente, incluso with AWS Serverless Application Model (AWS SAM), devi usare Docker versione 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 incluse nel runtime e le 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 incluse nel runtime sono installate nella directory `/var/runtime`. Le librerie installate da pip 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)
```

Usare un'immagine AWS base per Python

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- Docker (versione minima 20.10.10 per Python 3.12 e immagini base successive)
- Python

Creazione di un'immagine da un'immagine di base

Per creare un'immagine contenitore da un'immagine di AWS 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 for 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 variabile di `{LAMBDA_TASK_ROOT}` ambiente definita da Lambda.
- Imposta l'argomento `CMD` specificando il gestore della funzione Lambda.

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`.

```
docker build --platform linux/amd64 -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 comando seguente: `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"
```

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:
   - Sostituisci `docker-image:test` con il nome e il tag della tua immagine Docker.
   - 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 image in un digest di immagini specifico. Ciò significa che se punti il tag immagine utilizzato per distribuire 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.

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 di runtime Lambda, 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 awslambdaric
```

Puoi anche scaricare il client dell'interfaccia di runtime Python da. GitHub

L'esempio seguente dimostra come creare un'immagine contenitore per Python usando 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 Command Line Interface (AWS CLI) versione 2
- Docker
- Python

Creazione di un'immagine da un'immagine di base alternativa

Per creare un'immagine del contenitore da un'immagine non di AWS 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 for 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.

   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} \
        awslambdaric

# 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", "awslambdaric" ]
# 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`.

```
docker build --platform linux/amd64 -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. È possibile creare l'emulatore nell'immagine o utilizzare la seguente procedura per installarlo sul 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. Tieni presente quanto segue:

   - `docker-image` è il nome dell'immagine e `test` è il tag.
   - `/usr/local/bin/python -m awslambdaric 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 awslambdaric 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 awslambdaric 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 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:

   - Sostituisci `docker-image:test` con il nome e il tag della tua immagine Docker.

- 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 image in un digest di immagini specifico. Ciò significa che se punti il tag immagine utilizzato per distribuire 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.

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 .

Lavorare con i livelli per le funzioni Python Lambda

Un layer Lambda è un archivio di file.zip che contiene codice o dati supplementari. 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. Package del contenuto del layer. Ciò significa creare un archivio di file.zip che contenga le dipendenze da utilizzare nelle funzioni.
2. Crea il livello in Lambda.
3. Aggiungi il layer alle tue funzioni.

Questo argomento contiene passaggi e indicazioni su come impacchettare e creare correttamente un layer Python Lambda con dipendenze di librerie esterne.

Argomenti

Prerequisiti

Per seguire i passaggi di questa sezione, è necessario disporre di quanto segue:

- Python 3.11 e il programma di installazione del pacchetto pip
- AWS Command Line Interface () versione 2 AWS CLI

In questo argomento, facciamo riferimento all'applicazione di `layer-python`esempio nel repository awsdocs GitHub . Questa applicazione contiene script che scaricano le dipendenze e generano i livelli. L'applicazione contiene anche le funzioni corrispondenti che utilizzano le dipendenze dai livelli. Dopo aver creato un livello, potete implementare e richiamare la funzione corrispondente per

verificare che tutto funzioni correttamente. Poiché si utilizza il runtime Python 3.11 per le funzioni, i livelli devono essere compatibili anche con Python 3.11.

Nell'applicazione di `layer-python` esempio, ci sono due esempi:

- Il primo esempio prevede il pacchetto della `requests`libreria in un livello Lambda. La `layer/` directory contiene gli script per generare il layer. La `function/` directory contiene una funzione di esempio per verificare il funzionamento del layer. La maggior parte di questo tutorial spiega come creare e impacchettare questo layer.
- Il secondo esempio riguarda il pacchetto della numpylibreria in un livello Lambda. La `layer-numpy/` directory contiene gli script per generare il layer. La `function-numpy/` directory contiene una funzione di esempio per verificare il funzionamento del layer. Per un esempio di come creare e impacchettare questo layer, vedithe section called “Utilizzo delle distribuzioni `manylinux` delle ruote”.

Compatibilità a livello Python con Amazon Linux

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.

In Python, la maggior parte dei pacchetti sono disponibili come ruote (`.whl`file) oltre alla distribuzione dei sorgenti. Ogni ruota è un tipo di distribuzione integrata che supporta una combinazione specifica di versioni di Python, sistemi operativi e set di istruzioni della macchina.

Le ruote sono utili per garantire che il tuo layer sia compatibile con Amazon Linux. Quando scarichi le tue dipendenze, scarica la ruota universale, se possibile. (Per impostazione predefinita, `pip` installa la ruota universale, se disponibile). La ruota universale contiene `any` un tag della piattaforma, che indica che è compatibile con tutte le piattaforme, incluso Amazon Linux.

Nell'esempio che segue, impacchettate la `requests` libreria in un livello Lambda. La `requests` libreria è un esempio di pacchetto disponibile come ruota universale.

Non tutti i pacchetti Python sono distribuiti come ruote universali. Ad esempio, numpyha più distribuzioni di ruote, ognuna delle quali supporta un diverso set di piattaforme. Per tali pacchetti, scarica la `manylinux` distribuzione per garantire la compatibilità con Amazon Linux. Per istruzioni dettagliate su come impacchettare tali livelli, consultathe section called “Utilizzo delle distribuzioni `manylinux` delle ruote”.

In rari casi, un pacchetto Python potrebbe non essere disponibile come ruota. Se esiste solo la distribuzione del codice sorgente (sdist), ti consigliamo di installare e impacchettare le dipendenze in un ambiente Docker basato sull'immagine del contenitore di base di Amazon Linux 2023. Consigliamo questo approccio anche se desideri includere le tue librerie personalizzate scritte in altri linguaggi come C/C++. Questo approccio imita l'ambiente di esecuzione Lambda in Docker e garantisce che le dipendenze dei pacchetti non Python siano compatibili con Amazon Linux.

Percorsi di livello per i runtime di Python

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 `PATH` variabile raccolga il contenuto del layer, il file layer.zip dovrebbe avere le sue dipendenze nei seguenti percorsi di cartella:

- `python`
- `python/lib/python3.x/site-packages`

Ad esempio, il file.zip del layer risultante creato in questo tutorial ha la seguente struttura di directory:

```
layer_content.zip
# python
    # lib
        # python3.11
            # site-packages
                # requests
                # <other_dependencies> (i.e. dependencies of the requests package)
                # ...
```

La `requests`libreria è posizionata correttamente nella `python/lib/python3.11/site-packages` directory. Ciò garantisce che Lambda possa localizzare la libreria durante le chiamate delle funzioni.

Imballaggio del contenuto del livello

In questo esempio, impacchettate la `requests` libreria Python in un file.zip di livello. Completate i seguenti passaggi per installare e impacchettare il contenuto del layer.

Per installare e impacchettare il contenuto del layer

1. Clona il aws-lambda-developer-guide GitHub repository, che contiene il codice di esempio necessario nella `sample-apps/layer-python` directory.

```
git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
```

2. Vai alla `layer` directory dell'app di `layer-python` esempio. Questa directory contiene gli script che utilizzate per creare e impacchettare correttamente il layer.

```
cd aws-lambda-developer-guide/sample-apps/layer-python/layer
```

3. Esaminate il `requirements.txt`file. Questo file definisce le dipendenze da includere nel layer, vale a dire la `requests` libreria. È possibile aggiornare questo file per includere tutte le dipendenze che si desidera includere nel proprio layer.

   Example requirements.txt

```
requests==2.31.0
```

4. Assicuratevi di disporre delle autorizzazioni per eseguire entrambi gli script.

```
chmod 744 1-install.sh && chmod 744 2-package.sh
```

5. Esegui lo `1-install.sh`script utilizzando il seguente comando:

```
./1-install.sh
```

   Questo script utilizza `venv` per creare un ambiente virtuale Python denominato. `create_layer` Quindi installa tutte le dipendenze richieste nella directory. `create_layer/lib/python3.11/site-packages`

   Example 1-install.sh

```
python3.11 -m venv create_layer
source create_layer/bin/activate
pip install -r requirements.txt
```

6. Esegui lo `2-package.sh`script utilizzando il seguente comando:

```
./2-package.sh
```

Questo script copia il contenuto dalla `create_layer/lib` directory in una nuova directory denominata`python`. Quindi comprime il contenuto della `python` directory in un file denominato`layer_content.zip`. Questo è il file.zip per il livello. È possibile decomprimere il file e verificare che contenga la struttura di file corretta, come mostrato nella the section called “Percorsi di livello per i runtime di Python” sezione.

Example 2-package.sh

```
mkdir python
cp -r create_layer/lib python/
zip -r layer_content.zip python
```

Creare il livello

In questa sezione, prendi il `layer_content.zip` file che hai generato nella sezione precedente e lo carichi come layer Lambda. È possibile caricare un layer utilizzando AWS Management Console o l'API Lambda tramite AWS Command Line Interface ()AWS CLI. Quando caricate il file Layer .zip, nel PublishLayerVersion AWS CLI comando seguente, specificate `python3.11` come runtime compatibile e `arm64` come architettura compatibile.

```
aws lambda publish-layer-version --layer-name python-requests-layer \
    --zip-file fileb://layer_content.zip \
    --compatible-runtimes python3.11 \
    --compatible-architectures "arm64"
```

Dalla risposta, nota il`LayerVersionArn`, che `arn:aws:lambda:us-east-1:`*`123456789012`*`:layer:python-requests-layer:1` assomiglia a. Avrai bisogno di questo Amazon Resource Name (ARN) nel passaggio successivo di questo tutorial, quando aggiungi il layer alla tua funzione.

Aggiungere il layer alla tua funzione

In questa sezione, si distribuisce una funzione Lambda di esempio che utilizza `requests` la libreria nel relativo codice funzione, quindi si collega il layer. Per distribuire la funzione, è necessario un. the section called “Ruolo di esecuzione (autorizzazioni per le funzioni di accedere ad altre risorse)”

Se non disponi di un ruolo di esecuzione esistente, segui i passaggi nella sezione comprimibile. Altrimenti, passa alla sezione successiva per distribuire la funzione.

(Facoltativo) Crea un ruolo di esecuzione

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 AWSLambdaBasicExecutionRolepolitica dispone delle autorizzazioni necessarie alla funzione per scrivere i log in Logs. CloudWatch

Per distribuire la funzione Lambda

1. Passa alla directory `function/`. Se ti trovi attualmente nella `layer/` directory, esegui il seguente comando:

   ```
   cd ../function
   ```

2. Controlla il codice della funzione. La funzione importa la `requests` libreria, effettua una semplice richiesta HTTP GET e quindi restituisce il codice di stato e il corpo.

   ```
   import requests

   def lambda_handler(event, context):
       print(f"Version of requests library: {requests.__version__}")
       request = requests.get('https://api.github.com/')
       return {
           'statusCode': request.status_code,
           'body': request.text
       }
   ```

3. Crea un pacchetto di distribuzione di file.zip utilizzando il seguente comando:

```
zip my_deployment_package.zip lambda_function.py
```

4. Distribuisci la funzione. Nel AWS CLI comando seguente, sostituite il `--role` parametro con il vostro ruolo di esecuzione ARN:

```
aws lambda create-function --function-name python_function_with_layer \
    --runtime python3.11 \
    --architectures "arm64" \
    --handler lambda_function.lambda_handler \
    --role arn:aws:iam::123456789012:role/lambda-role \
    --zip-file fileb://my_deployment_package.zip
```

(Facoltativo) Richiamate la funzione senza collegare un livello

A questo punto, puoi facoltativamente provare a richiamare la tua funzione prima di collegare il livello. Se provate a farlo, dovreste ricevere un errore di importazione perché la funzione non può fare riferimento al pacchetto. `requests` Per richiamare la tua funzione, usa il seguente AWS CLI comando:

```
aws lambda invoke --function-name python_function_with_layer \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "key": "value" }' response.json
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

Per visualizzare l'errore specifico, aprite il `response.json` file di output. Dovresti vedere un messaggio `ImportModuleError` con il seguente messaggio di errore:

```
"errorMessage": "Unable to import module 'lambda_function': No module named 'requests'"
```

Quindi, collega il livello alla tua funzione. Nel AWS CLI comando seguente, sostituite il `--layers` parametro con la versione del layer ARN che avete notato in precedenza:

```
aws lambda update-function-configuration --function-name python_function_with_layer \
    --cli-binary-format raw-in-base64-out \
    --layers "arn:aws:lambda:us-east-1:123456789012:layer:python-requests-layer:1"
```

Infine, provate a richiamare la vostra funzione usando il seguente comando: AWS CLI

```
aws lambda invoke --function-name python_function_with_layer \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "key": "value" }' response.json
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Il `response.json` file di output contiene dettagli sulla risposta.

(Facoltativo) Pulisci le tue 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 ai tuoi Account AWS.

Per eliminare il layer Lambda

1. Apri la pagina Layers (Livelli) nella console Lambda.
2. Seleziona il livello che hai creato.
3. Scegliete Elimina, quindi scegliete nuovamente 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. Inserisci **`delete`** nel campo di immissione del testo, quindi scegli Elimina.

Utilizzo delle distribuzioni **manylinux** delle ruote

A volte, un pacchetto che si desidera includere come dipendenza non ha una ruota universale (in particolare, non ha `any` come tag della piattaforma). In questo caso, scarica `manylinux` invece la ruota che supporta. Ciò garantisce che le tue librerie di livelli siano compatibili con Amazon Linux.

numpyè un pacchetto che non dispone di una ruota universale. Se si desidera includere il `numpy` pacchetto nel layer, è possibile completare i seguenti passaggi di esempio per installare e impacchettare correttamente il layer.

Per installare e impacchettare il contenuto del layer

1. Clona il `aws-lambda-developer-guide` GitHub repository, che contiene il codice di esempio necessario nella `sample-apps/layer-python` directory.

   ```
   git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
   ```

2. Vai alla `layer-numpy` directory dell'app di `layer-python` esempio. Questa directory contiene gli script che utilizzate per creare e impacchettare correttamente il layer.

   ```
   cd aws-lambda-developer-guide/sample-apps/layer-python/layer-numpy
   ```

3. Esaminate il `requirements.txt`file. Questo file definisce le dipendenze da includere nel layer, vale a dire la `numpy` libreria. Qui, specifichi l'URL della distribuzione `manylinux` wheel compatibile con Python 3.11, Amazon Linux e il `x86_64` set di istruzioni:

   Example requirements.txt

   ```
   https://files.pythonhosted.org/packages/3a/d0/
   edc009c27b406c4f9cbc79274d6e46d634d139075492ad055e3d68445925/numpy-1.26.4-cp311-
   cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
   ```

4. Assicurati di disporre delle autorizzazioni per eseguire entrambi gli script.

   ```
   chmod 744 1-install.sh && chmod 744 2-package.sh
   ```

5. Esegui lo `1-install.sh`script utilizzando il seguente comando:

   ```
   ./1-install.sh
   ```

Questo script utilizza `venv` per creare un ambiente virtuale Python denominato. `create_layer` Quindi installa tutte le dipendenze richieste nella directory. `create_layer/lib/python3.11/site-packages` Il `pip` comando è diverso in questo caso, perché è necessario specificare il `--platform` tag come. `manylinux2014_x86_64` Questo indica `pip` di installare la `manylinux` ruota corretta, anche se il computer locale utilizza macOS o Windows.

Example 1-install.sh

```
python3.11 -m venv create_layer
source create_layer/bin/activate
pip install -r requirements.txt --platform=manylinux2014_x86_64 --only-binary=:all:
 --target ./create_layer/lib/python3.11/site-packages
```

6. Esegui lo `2-package.sh`script utilizzando il seguente comando:

```
./2-package.sh
```

Questo script copia il contenuto dalla `create_layer/lib` directory in una nuova directory denominata`python`. Quindi comprime il contenuto della `python` directory in un file denominato`layer_content.zip`. Questo è il file.zip per il livello. È possibile decomprimere il file e verificare che contenga la struttura di file corretta, come mostrato nella the section called "Percorsi di livello per i runtime di Python" sezione.

Example 2-package.sh

```
mkdir python
cp -r create_layer/lib python/
zip -r layer_content.zip python
```

Per caricare questo layer su Lambda, usa il seguente comando: PublishLayerVersion AWS CLI

```
aws lambda publish-layer-version --layer-name python-numpy-layer \
    --zip-file fileb://layer_content.zip \
    --compatible-runtimes python3.11 \
    --compatible-architectures "x86_64"
```

Dalla risposta, nota il`LayerVersionArn`, che `arn:aws:lambda:us-east-1:123456789012:layer:python-numpy-layer:1` assomiglia a. Per verificare che il layer funzioni come previsto, distribuite la funzione Lambda nella `function-numpy` directory.

Per distribuire la funzione Lambda

1. Passa alla directory `function-numpy/`. Se ti trovi attualmente nella `layer-numpy/` directory, esegui il seguente comando:

```
cd ../function-numpy
```

2. Controlla il codice della funzione. La funzione importa la `numpy` libreria, crea un `numpy` array semplice e quindi restituisce un codice di stato e un corpo fittizi.

```
import json
import numpy as np

def lambda_handler(event, context):

    x = np.arange(15, dtype=np.int64).reshape(3, 5)
    print(x)

    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

3. Crea un pacchetto di distribuzione di file.zip utilizzando il seguente comando:

```
zip my_deployment_package.zip lambda_function.py
```

4. Distribuisci la funzione. Nel AWS CLI comando seguente, sostituite il `--role` parametro con il vostro ruolo di esecuzione ARN:

```
aws lambda create-function --function-name python_function_with_numpy \
    --runtime python3.11 \
    --handler lambda_function.lambda_handler \
    --role arn:aws:iam::123456789012:role/lambda-role \
    --zip-file fileb://my_deployment_package.zip
```

(Facoltativo) Richiamate la funzione senza collegare un livello

Facoltativamente, potete provare a richiamare la funzione prima di collegare il livello. Se provate a farlo, dovreste ricevere un errore di importazione perché la funzione non può fare riferimento al pacchetto. numpy Per richiamare la tua funzione, usa il seguente AWS CLI comando:

```
aws lambda invoke --function-name python_function_with_numpy \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "key": "value" }' response.json
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

Per visualizzare l'errore specifico, aprite il `response.json` file di output. Dovresti vedere un messaggio `ImportModuleError` con il seguente messaggio di errore:

```
"errorMessage": "Unable to import module 'lambda_function': No module named 'numpy'"
```

Quindi, collega il livello alla tua funzione. Nel AWS CLI comando seguente, sostituite il `--layers` parametro con la versione del layer ARN:

```
aws lambda update-function-configuration --function-name python_function_with_numpy \
    --cli-binary-format raw-in-base64-out \
    --layers "arn:aws:lambda:us-east-1:123456789012:layer:python-requests-layer:1"
```

Infine, provate a richiamare la vostra funzione usando il seguente comando: AWS CLI

```
aws lambda invoke --function-name python_function_with_numpy \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "key": "value" }' response.json
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{
```

```
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

È possibile esaminare i log delle funzioni per verificare che il codice stampi l'numpyarray in modo standard out.

Oggetto contesto AWS Lambda in 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 Definisci il gestore di funzioni 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.

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.

AWS Lambda registrazione delle funzioni in Python

AWS Lambda monitora automaticamente le funzioni Lambda e invia le voci di registro ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della 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 registri, consulta. Utilizzo dei CloudWatch log di Amazon con AWS Lambda

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 del flusso di log CloudWatch Logs e dell'oggetto evento.

Nota che se la tua funzione genera log usando istruzioni Python`print`, Lambda può inviare output di log a Logs solo in formato testo semplice. CloudWatch 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: $LATEST
## ENVIRONMENT VARIABLES
/aws/lambda/my-function
2023/08/31/[$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: 16 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 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 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.
- 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 segmento X-Ray.
- 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 di registro 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: $LATEST
[INFO]  2023-08-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    ##
 ENVIRONMENT VARIABLES
[INFO]  2023-08-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    /aws/
lambda/my-function
[INFO]  2023-08-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    2023/01/31/
[$LATEST]1bbe51xmplb34a2788dbaa7433b0aa4d
[INFO]  2023-08-31T22:12:58.535Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    ## EVENT
[INFO]  2023-08-31T22:12:58.535Z    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: 3 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 avere 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 Python di Lambda supportati:

- Formato di log: scegli tra i formati di testo normale e JSON strutturato per i log della funzione
- Livello di log: per i log in formato JSON, scegli il livello di dettaglio dei log che Lambda invia ad CloudWatch Amazon, come ERROR, DEBUG o INFO
- Gruppo di log: scegli il gruppo di log a cui la CloudWatch 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 la funzione 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 per il formato di log della tua funzione, Lambda invierà i log in uscita dalla libreria di registrazione standard Python in un formato 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 di log generati utilizzando la libreria `logging` Python vengono acquisiti CloudWatch in Logs quando si configura il formato di log della funzione come JSON.

Tieni presente che se usi il metodo `print` per generare output log di base come descritto in the section called “Stampa nel log”, Lambda acquisirà questi output come testo normale anche se configuri il formato di registrazione della funzione come JSON.

Output log JSON standard che utilizzano la libreria di registrazione di Python

Il seguente frammento di codice e l'output di registro 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":"2023-10-27T19:17:45.586Z",
    "level":"INFO",
    "message":"Inside the handler function",
    "logger": "root",
    "requestId":"79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Registrazione di parametri aggiuntivi in JSON

Quando il formato di log della funzione è impostato su JSON, puoi anche registrare parametri aggiuntivi con la libreria `logging` Python standard usando `extra` la parola chiave per passare un dizionario Python all'output del 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": "2023-11-02T15:26:28Z",
  "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": "2023-11-02T16:18:57Z",
  "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 AWS Lambda, per produrre log strutturati JSON, non è necessario apportare alcuna modifica. AWS Lambda non codifica due volte i log che sono già codificati in JSON. Anche se configuri la tua funzione per utilizzare il formato di registro JSON, i tuoi output di registrazione vengono visualizzati nella struttura JSON che definisci. CloudWatch

L'esempio seguente mostra come gli output di log generati utilizzando il pacchetto Powertools for AWS Lambda vengono acquisiti in Logs. CloudWatch Il formato di questo output 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, vedere e AWS Lambdathe section called “Utilizzo di Powertools per AWS Lambda (Python) AWS SAM e per la registrazione strutturata” the section called “Utilizzo di Powertools per AWS Lambda (Python) AWS CDK e per la registrazione strutturata”

Example Frammento di codice di registrazione Python (usando Powertools for) 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": "2023-10-31 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 filtraggio a livello di registro, puoi scegliere di inviare solo i log di 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 “Filtraggio a livello di log”.

Per AWS Lambda filtrare i log delle applicazioni in base al relativo livello di registro, la funzione deve utilizzare log in formato JSON. Puoi farlo in due modi:

- Crea output log utilizzando la libreria standard `logging` di Python e configura la funzione affinché utilizzi JSON come formato di log. Successivamente, AWS Lambda filtra gli output log utilizzando la coppia chiave-valore "livello" nell'oggetto JSON descritto in the section called “Utilizzo di log JSON strutturati con Python”. Per informazioni su come configurare il formato di log della funzione, consulta la pagina the section called “Configurazione dei controlli di registrazione avanzati per la funzione Lambda”.
- Utilizza un'altra libreria o metodo di registrazione per creare nel codice dei log JSON strutturati che includono una coppia chiave-valore "livello" che definisce il livello dell'output log. Ad esempio, puoi utilizzare Powertools per AWS Lambda generare output 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 invierà 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 assegnerà al log il livello INFO e aggiungerà un timestamp per tuo conto.

Visualizzazione dei log nella console di 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 del codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l'output del log 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. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegliere il gruppo di log della funzione (/aws/lambda/***function-name***).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Nuovi flussi di log vengono visualizzati quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire più 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 il flusso di log nella traccia.

Visualizzazione dei log con AWS CLI

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- AWS CLI — Configurazione rapida con `aws configure`

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":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "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-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.

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 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.

```
#!/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 seguente comando 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.

Strumenti 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, 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 (Python) AWS SAM e 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 back-end 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 SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS SAM

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 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 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:

```
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04
 2023-02-03T14:59:50.371000 INIT_START Runtime Version:
 python:3.9.v16    Runtime Version ARN: arn:aws:lambda:us-
east-1::runtime:07a48df201798d627f2b950f03bb227aab4a655a1d019c3296406f95937e2525
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.112000
 START RequestId: d455cfc4-7704-46df-901b-2a5cce9405be Version: $LATEST
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.114000 {
  "level": "INFO",
  "location": "hello:23",
  "message": "Hello world API - HTTP 200",
  "timestamp": "2023-02-03 14:59:51,113+0000",
  "service": "PowertoolsHelloWorld",
  "cold_start": true,
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYtOc9j",
  "function_memory_size": "128",
  "function_arn": "arn:aws:lambda:us-east-1:111122223333:function:sam-app-
HelloWorldFunction-YBg8yfYtOc9j",
  "function_request_id": "d455cfc4-7704-46df-901b-2a5cce9405be",
  "correlation_id": "e73f8aef-5e07-436e-a30b-63e4b23f0047",
  "xray_trace_id": "1-63dd2166-434a12c22e1307ff2114f299"
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "function_name",
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
```

```
        ]
      }
    ]
  },
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYtOc9j",
  "service": "PowertoolsHelloWorld",
  "ColdStart": [
    1.0
  ]
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "HelloWorldInvocations",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "service": "PowertoolsHelloWorld",
  "HelloWorldInvocations": [
    1.0
  ]
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.128000 END
 RequestId: d455cfc4-7704-46df-901b-2a5cce9405be
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.128000
 REPORT RequestId: d455cfc4-7704-46df-901b-2a5cce9405be    Duration: 16.33 ms
 Billed Duration: 17 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 configurare la conservazione dei log, aggiungi quanto segue al tuo modello: AWS SAM

```
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) AWS CDK e per la registrazione strutturata

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Hello World Python di esempio con i moduli Powertools for AWS Lambda (Python) integrati utilizzando il. AWS CDK Questa applicazione implementa un back-end 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, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS CDK

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 costruttore 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, o 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_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:

```
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04
 2023-02-03T14:59:50.371000 INIT_START Runtime Version:
 python:3.9.v16    Runtime Version ARN: arn:aws:lambda:us-
east-1::runtime:07a48df201798d627f2b950f03bb227aab4a655a1d019c3296406f95937e2525
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.112000
 START RequestId: d455cfc4-7704-46df-901b-2a5cce9405be Version: $LATEST
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.114000 {
  "level": "INFO",
  "location": "hello:23",
    "message": "Hello world API - HTTP 200",
  "timestamp": "2023-02-03 14:59:51,113+0000",
  "service": "PowertoolsHelloWorld",
  "cold_start": true,
```

```
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYtOc9j",
  "function_memory_size": "128",
  "function_arn": "arn:aws:lambda:us-east-1:111122223333:function:sam-app-
HelloWorldFunction-YBg8yfYtOc9j",
  "function_request_id": "d455cfc4-7704-46df-901b-2a5cce9405be",
  "correlation_id": "e73f8aef-5e07-436e-a30b-63e4b23f0047",
  "xray_trace_id": "1-63dd2166-434a12c22e1307ff2114f299"
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "function_name",
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYtOc9j",
  "service": "PowertoolsHelloWorld",
  "ColdStart": [
    1.0
  ]
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "service"
```

```
          ]
        ],
        "Metrics": [
          {
            "Name": "HelloWorldInvocations",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "service": "PowertoolsHelloWorld",
  "HelloWorldInvocations": [
    1.0
  ]
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.128000 END
 RequestId: d455cfc4-7704-46df-901b-2a5cce9405be
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.128000
 REPORT RequestId: d455cfc4-7704-46df-901b-2a5cce9405be    Duration: 16.33 ms
 Billed Duration: 17 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
```

Test delle funzioni AWS Lambda in Python

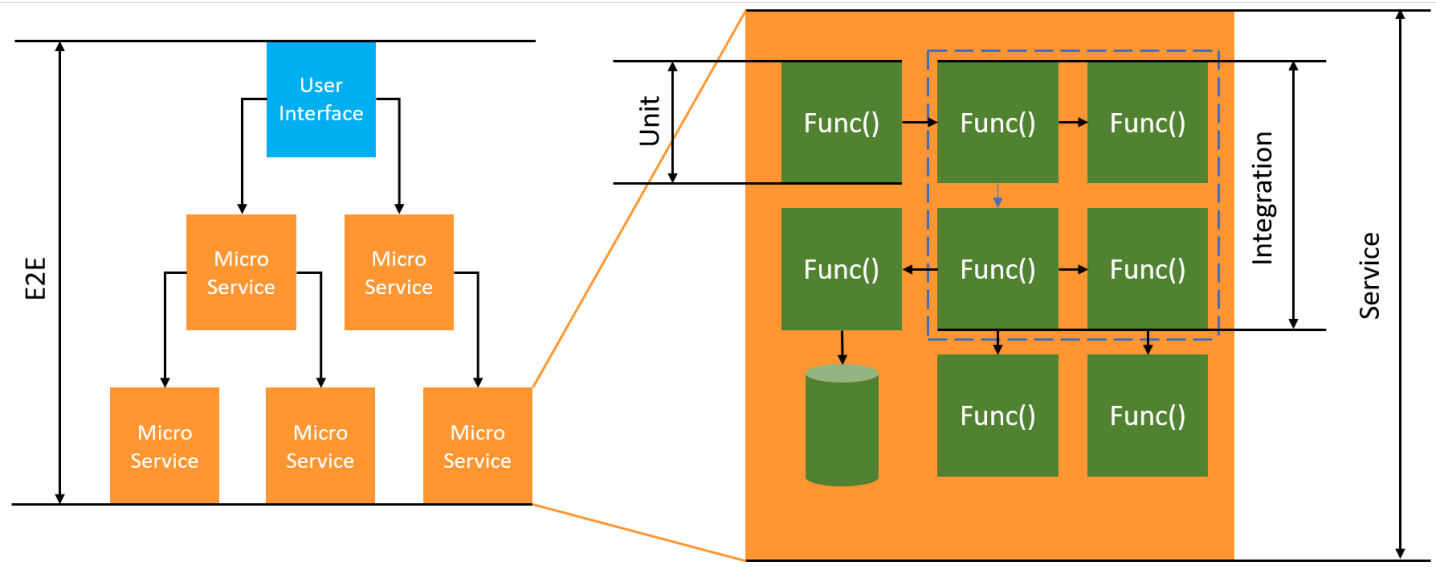
> **Note**
>
> Consulta il capitolo sul Test delle funzioni per un'introduzione completa alle tecniche e alle best practice per testare soluzioni serverless.

Sebbene il test delle funzioni serverless si basi su tipologie e tecniche di test consolidate, è importante tenere in considerazione la possibilità di effettuare test sulle applicazioni serverless nella loro interezza. I test basati sul cloud forniranno la misura più accurata della qualità sia delle funzioni sia 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 senza server, continuerai a scrivere unità, integrazioni e end-to-endtest.

- 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.
- E nd-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

Generalmente, utilizzerai una combinazione di approcci per testare il codice dell'applicazione serverless, inclusi test nel cloud, test con mock e occasionalmente test con emulatori.

Test nel cloud

I test nel cloud sono utili per tutte le fasi del test, inclusi test unitari, test di integrazione e end-to-end test. 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 servizi e risorse AWS, in modo da poter testare le funzioni con poche o nessuna modifica utilizzando decoratori per intercettare e simulare le risposte.

La funzione di convalida di Powertools per AWS Lambda (Python) fornisce decoratori che consentono di convalidare gli eventi di input e le risposte di output dalle funzioni Python.

Per ulteriori informazioni, leggi il post del blog Test unitari di Lambda con Python e servizi AWS fittizi.

Per ridurre la latenza associata alle iterazioni di implementazione del cloud, consulta le sezioni AWS Serverless Application Model (AWS SAM) Accelerate e AWS Cloud Development Kit (AWS CDK) watch mode. 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 utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, che possono includere funzioni Lambda e altri servizi AWS .

Per inviare dati di tracciamento a X-Ray, è possibile utilizzare una delle tre librerie SDK:

- AWS Distro for OpenTelemetry (ADOT): una distribuzione sicura, pronta per la produzione e supportata dell'SDK (Otel AWS). OpenTelemetry
- SDK AWS X-Ray per Python: un SDK per generare e inviare i dati di traccia su X-Ray.
- Powertools for AWS Lambda (Python): un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori.

Ciascun SDK offre dei modi per inviare i dati di telemetria al servizio X-Ray. Puoi quindi utilizzare X-Ray per visualizzare, filtrare e analizzare le metriche delle prestazioni dell'applicazione per identificare i problemi e le opportunità di ottimizzazione.

> **Important**
>
> 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 end-to-end tracciamento in X-Ray utilizzando entrambe le soluzioni. Per ulteriori informazioni sulla scelta più adatta, consulta Scegliere tra le AWS Distro per OpenTelemetry e SDK X-Ray.

Sections

- Interpretazione di una traccia X-Ray
- Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray)

Usare Powertools per AWS Lambda (Python) AWS SAM e per tracciare

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Hello World Python di esempio con i moduli Powertools for AWS Lambda (Python) integrati utilizzando il. AWS SAM Questa applicazione implementa un back-end 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, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS SAM

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 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-YBg8yfYtOc9j -
 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-YBg8yfYtOc9j
 - 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-YBg8yfYtOc9j - 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-YBg8yfYtOc9j [HTTP: 200]
 - 0.016s - sam-app-HelloWorldFunction-YBg8yfYtOc9j
   - 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 nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste.

> **Note**
>
> La frequenza di campionamento di X-Ray non può essere configurata per le funzioni.

Usare Powertools per AWS Lambda (Python) e AWS CDK per tracciare

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Hello World Python di esempio con i moduli Powertools for AWS Lambda (Python) integrati utilizzando il. AWS CDK

Questa applicazione implementa un back-end 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 CDK versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS CDK

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 costruttore 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, o 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_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 -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-YBg8yfYtOc9j -
 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-YBg8yfYtOc9j
 - 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-YBg8yfYtOc9j - 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-YBg8yfYtOc9j [HTTP: 200]
 - 0.016s - sam-app-HelloWorldFunction-YBg8yfYtOc9j
   - 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 layer, consulta AWS Distro for OpenTelemetry Lambda Support for Python nella documentazione ADOT.

Utilizzo dell'SDK X-Ray per strumentare le 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 strumentare i client AWS SDK applicando una patch alla libreria con il modulo. `boto3` `aws_xray_sdk.core`

Example funzione — Tracciamento di un client SDK AWS

```
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 aver apportato le modifiche necessarie al codice, attivare 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. Scegli Modifica.
5. In X-Ray, attivare Active tracing (Tracciamento attivo).
6. Selezionare Salva.

Attivazione del tracciamento con l'API Lambda

Configura il tracciamento sulla tua funzione Lambda con AWS o SDK, utilizza AWS CLI le seguenti operazioni API:

- UpdateFunctionConfigurazione
- GetFunctionConfigurazione
- CreateFunction

Il AWS CLI comando di esempio seguente abilita il tracciamento attivo 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 AWS CloudFormation

Per attivare il tracciamento su una `AWS::Lambda::Function` risorsa in un AWS 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), utilizzate la `Tracing` proprietà.

Example template.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretazione di una traccia X-Ray

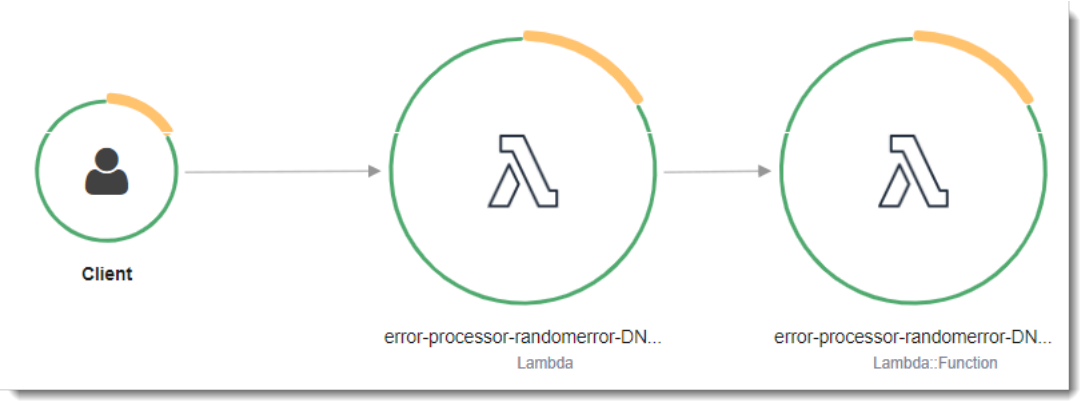
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al ruolo di esecuzione della funzione. Altrimenti, aggiungete la AWSXRayDaemonWriteAccesspolitica al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico dei servizi X-Ray mostra informazioni sull'applicazione e tutti i suoi componenti. L'immagine seguente 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 compaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon Logs. CloudWatch

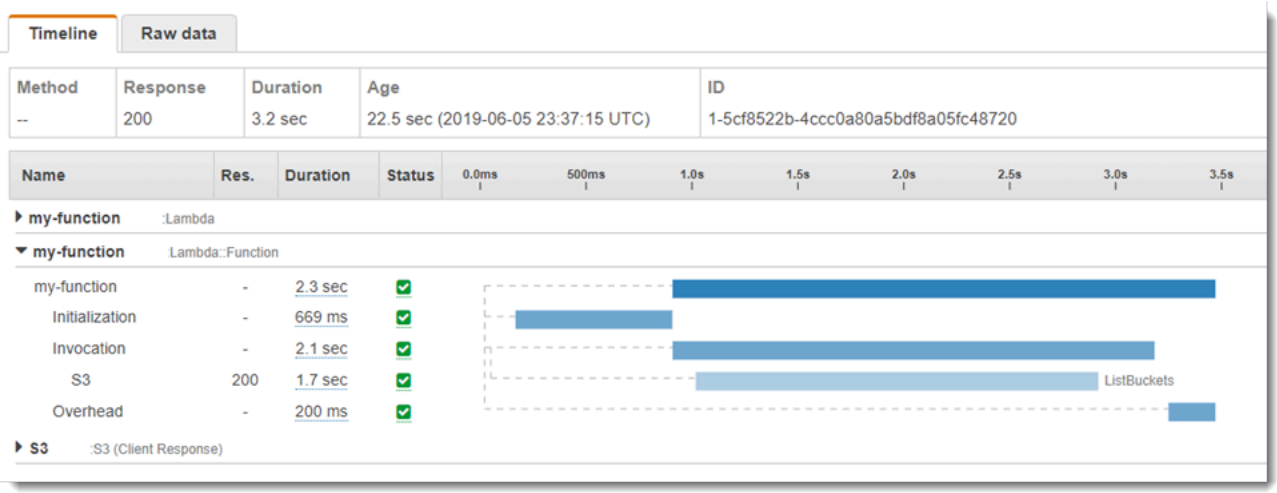
X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste.

> **Note**
>
> La frequenza di campionamento di X-Ray non può essere configurata per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, il che crea 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 denominati my-function, ma uno ha un'origine di `AWS::Lambda` e l'altro ha un'origine di. `AWS::Lambda::Function` Se il `AWS::Lambda` segmento mostra un errore, il servizio Lambda presentava un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione presentava un problema.



Questo esempio espande il `AWS::Lambda::Function` segmento per mostrarne i tre 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.

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 tracciamento X-Ray gratuitamente ogni mese fino a un determinato limite come parte del AWS piano gratuito. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consultare Prezzi di AWS X-Ray.

Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray)

Se utilizzate X-Ray SDK per strumentare i client AWS SDK del codice della funzione, il pacchetto di distribuzione può diventare piuttosto grande. Per evitare di caricare dipendenze di runtime ogni volta che si aggiorna il codice della funzione, includere l'SDK X-Ray in un livello 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:
        - python3.8
```

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 layer per le dipendenze richiede modifiche alla compilazione per generare l'archivio dei layer 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 for Ruby, con le credenziali di un ruolo AWS Identity and Access Management (IAM) che gestisci. Per saperne di più sulle versioni SDK incluse nei runtime di Ruby, consulta. the section called “Versioni SDK incluse in runtime”

Lambda supporta i seguenti runtime di Ruby.

Ruby

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Ruby 3.3	`ruby3.3`	Amazon Linux 2023			
Ruby 3.2	`ruby3.2`	Amazon Linux 2			

Per creare una funzione Ruby

1. Aprire la console Lambda.
2. Scegli Crea funzione.
3. Configurare le impostazioni seguenti:
   - Nome della funzione: inserisci il nome della funzione.
   - Runtime: scegli Ruby 3.2.
4. Scegli Crea funzione.
5. Per configurare un evento di test scegliere Test.
6. Per Event name (Nome evento) immettere **`test`**.
7. Seleziona Salvataggio delle modifiche.
8. Per invocare la funzione, scegliere Test (Testa).

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. Per salvare le modifiche, scegliere Save (Salva). Quindi, per eseguire il codice, scegliere Test (Testa).

> **Note**
>
> La console AWS Cloud9 Lambda fornisce un ambiente di sviluppo integrato nel browser. Puoi anche usarle AWS Cloud9 per sviluppare funzioni Lambda nel tuo ambiente. Per ulteriori informazioni, consulta Lavorare con AWS Lambda le funzioni utilizzando la Kit di strumenti AWS guida per l' AWS Cloud9 utente.

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.

> **Note**
>
> Per iniziare a sviluppare applicazioni nell'ambiente locale, implementate una delle applicazioni di esempio disponibili nell' GitHub archivio di questa guida.
>
> Applicazioni Lambda di esempio in Ruby
>
> - blank-ruby — Una funzione Ruby che mostra l'uso della registrazione, delle variabili di ambiente, del AWS X-Ray tracciamento, dei livelli, dei test unitari e dell'SDK. AWS
> - Esempi di codice Ruby per AWS Lambda — Esempi di codice scritti in Ruby che dimostrano come interagire con Lambda. AWS

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

Versioni SDK incluse in runtime

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 il numero di versione di un particolare service gem incluso nel runtime che stai utilizzando, crea una funzione Lambda con codice nel seguente formato. Sostituisci `aws-sdk-s3` e `Aws::S3` con il nome dei service gem utilizzati 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)

Il runtime di Ruby 3.2 supporta 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 3.2, imposta la variabile di ambiente `RUBY_YJIT_ENABLE` su `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
```

Definisci il gestore 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.

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 Ambiente di esecuzione Lambda.

Utilizzo di archivi di file .zip per le funzioni Lambda in Ruby

Il codice della tua AWS Lambda funzione comprende un file.rb contenente il codice del gestore della 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 versione di Ubuntu e Bash integrata con Windows.

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

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

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 maggiori informazioni sull'uso dei livelli Lambda, consulta Livelli Lambda.

Il runtime di Ruby include. AWS SDK for 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 scoprire quale versione dell'SDK for Ruby è inclusa nella tua versione di runtime, consulta. the section called “Versioni SDK incluse in 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 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 si trova nella directory principale 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

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”.

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 `PATH` variabile raccolga il contenuto del layer, il file.zip del layer dovrebbe avere le sue dipendenze nei seguenti percorsi di cartella:

- `ruby/gems/2.7.0` (`GEM_PATH`)
- `ruby/lib` (`RUBYLIB`)

Ad esempio, la struttura del file .zip del livello potrebbe essere simile alla seguente:

```
json.zip
# ruby/gems/2.7.0/
               | build_info
               | cache
               | doc
               | extensions
```

```
                | gems
                | # json-2.1.0
                # specifications
                  # json-2.1.0.gemspec
```

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.

Creazione di un pacchetto di implementazione .zip con librerie native

Molte gemme Ruby comuni, come `nokogiri`, `nio4r` e `mysql`, contengono estensioni native scritte in C. Quando aggiungi librerie contenenti codice C al pacchetto di implementazione, devi creare il pacchetto correttamente per assicurarti che sia compatibile con l'ambiente di esecuzione Lambda.

Per le applicazioni di produzione, consigliamo di creare e distribuire il codice utilizzando (). AWS Serverless Application Model AWS SAM AWS SAM Usa l'`sam build --use-container`opzione per creare la tua funzione all'interno di un contenitore Docker simile a Lambda. Per saperne di più sull'utilizzo AWS SAM per distribuire il codice della funzione, consulta Building applications nella Developer Guide.AWS SAM

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#
   ```

4. 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
```

5. 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
```

6. 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 Lambda AWS Command Line Interface e l'API Lambda. Puoi anche creare e aggiornare le funzioni Lambda usando AWS Serverless Application Model (AWS SAM) e AWS 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 Linux, Lambda necessita di 644 autorizzazioni per i file non eseguibili (rw-r--r--) e 755 permessi () per le directory e i file eseguibili. rwxr-xr-x

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 eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 755 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta Set, View, Change, or Remove Permissions on an Object nella documentazione di Microsoft Windows.

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 AWS Management Console, consulta la Guida introduttiva ad Amazon S3. Per caricare file utilizzando la AWS CLI, consulta Move objects nella Guida per l'AWS CLI utente.

> **Note**
>
> Non è possibile modificare il tipo di pacchetto di distribuzione (.zip o immagine del contenitore) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine del contenitore 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. Selezionare 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, Node.js o Ruby)
- Il pacchetto di implementazione della funzione deve avere dimensioni inferiori a 3 MB.

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. Quando hai finito di modificare il codice, scegli Implementa per salvare le modifiche e aggiornare la 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. Usa la funzione create-function e update-function-codei comandi per distribuire 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 su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

> **Note**
>
> Se carichi il tuo file.zip da un bucket Amazon S3 utilizzando AWS CLI il, il bucket deve trovarsi nella stessa posizione della Regione AWS tua funzione.

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=DOC-EXAMPLE-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 DOC-EXAMPLE-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

Il 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 utilizzi 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 eventuali dipendenze specificate. Per ulteriori informazioni sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda, consulta la Guida introduttiva AWS Serverless Application Model alla AWS SAM Developer Guide.

È inoltre possibile utilizzare 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 tua macchina di compilazione. Per istruzioni su come caricare un file su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.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 del codice della funzione, sul percorso della cartella locale o sull'oggetto FunctionCode
- `Runtime`: imposta il runtime prescelto

Inoltre AWS SAM, se il tuo file.zip è più grande di 50 MB, non è necessario caricarlo prima in un bucket Amazon S3. AWS SAM puoi caricare pacchetti.zip fino alla dimensione massima consentita di 250 MB (decompressi) da una posizione sulla macchina di compilazione locale.

Per ulteriori informazioni sulla distribuzione delle funzioni utilizzando il file.zip in AWS SAM, consulta la Guida per gli sviluppatori. AWS::Serverless::FunctionAWS SAM

Creazione e aggiornamento di funzioni con file.zip utilizzando AWS CloudFormation

È possibile utilizzare AWS 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 su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

Nel AWS 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 che AWS CloudFormation genera non può superare i 4 MB. Per ulteriori informazioni sulla distribuzione delle funzioni utilizzando il file.zip in AWS CloudFormation, AWS::Lambda::Functionconsultate la Guida per l'utente.AWS 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.
- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  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 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 base AWS

  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

- 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	Definizione come obsoleto
3.3	Ruby 3.3	Amazon Linux 2023	Dockerfile per Ruby 3.3 su GitHub	
3.2	Ruby 3.2	Amazon Linux 2	Dockerfile per Ruby 3.2 su GitHub	

Repository Amazon ECR: gallery.ecr.aws/lambda/ruby

Usare un'immagine di AWS base per Ruby

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- 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 for 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.
   - Usa il comando COPY per copiare il codice della funzione e le dipendenze di runtime in una variabile di `{LAMBDA_TASK_ROOT}` ambiente definita da Lambda.
   - Imposta l'argomento `CMD` specificando il gestore della funzione Lambda.

   Example Dockerfile

```
FROM public.ecr.aws/lambda/ruby:3.2

# 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`.

```
docker build --platform linux/amd64 -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 comando seguente: `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"
   ```

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:
   - Sostituisci `docker-image:test` con il nome e il tag della tua immagine Docker.
   - 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 image in un digest di immagini specifico. Ciò significa che se punti il tag immagine utilizzato per distribuire 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.

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 di runtime Lambda, che gestisce l'interazione tra Lambda e il codice della funzione.

Installa il client dell'interfaccia di runtime Lambda per Ruby utilizzando il gestore di RubyGems pacchetti.org:

```
gem install aws_lambda_ric
```

Puoi anche scaricare il client dell'interfaccia di runtime Ruby da. GitHub Il client di interfaccia di runtime supporta le versioni di Ruby da 2.5.x a 2.7.x.

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 Command Line Interface (AWS CLI) versione 2
- 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 for 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.
   - Create una directory per il codice della funzione e una variabile di ambiente che punti a quella directory. In questo esempio, la directory is`/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 base. AWS
   - 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.

   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`.

```
docker build --platform linux/amd64 -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 utilizzare la seguente procedura per 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. Tieni presente quanto segue:

- `docker-image` è il nome dell'immagine e `test` è il tag.
- `aws_lambda_ric 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_ric 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_ric 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 commando 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:

   - Sostituisci `docker-image:test` con il nome e il tag della tua immagine Docker.
   - 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 image in un digest di immagini specifico. Ciò significa che se punti il tag immagine utilizzato per distribuire 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.

Oggetto contesto AWS Lambda in 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.

AWS Lambda registrazione delle funzioni in Ruby

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della 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 Utilizzo dei CloudWatch log di Amazon con AWS Lambda.

Questa pagina descrive come produrre un output di registro dal codice della funzione Lambda o accedere ai log utilizzando AWS Command Line Interface la console Lambda o la console. CloudWatch

Sections

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.

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: 16 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 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.
- 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 segmento X-Ray.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Per ottenere log più dettagliati, utilizza la the section called “Libreria di registrazione”.

Uso della 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 del codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l'output del log nella sezione Dettagli.

Utilizzo della 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. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegliere il gruppo di log della funzione (/aws/lambda/***`function-name`***).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Nuovi flussi di log vengono visualizzati quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire più 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 il flusso di log nella traccia.

AWS Command Line InterfaceAWS CLI Usando il ()

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- AWS CLI — Configurazione rapida con `aws configure`

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":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "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-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.

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 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.

```
#!/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 seguente comando 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.

Libreria di registrazione

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: 3 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 consentirti 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, è possibile registrare errori e latenza per ogni chiamata effettuata dalla funzione a un servizio. AWS

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico dei servizi X-Ray mostra informazioni sull'applicazione e tutti i suoi componenti. L'immagine seguente 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 compaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon Logs. CloudWatch

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. Scegli Modifica.

5. In X-Ray, attivare Active tracing (Tracciamento attivo).
6. Selezionare Salva.

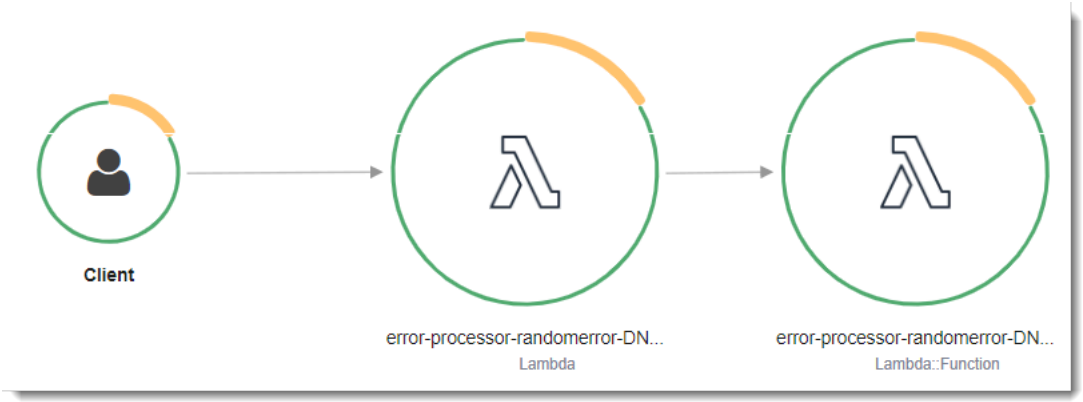
> **Prezzi**
>
> Puoi utilizzare il tracciamento X-Ray gratuitamente ogni mese fino a un determinato limite come parte del AWS piano gratuito. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consultare Prezzi di AWS X-Ray.

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 AWSXRayDaemonWriteAccesspolicy al ruolo di esecuzione.

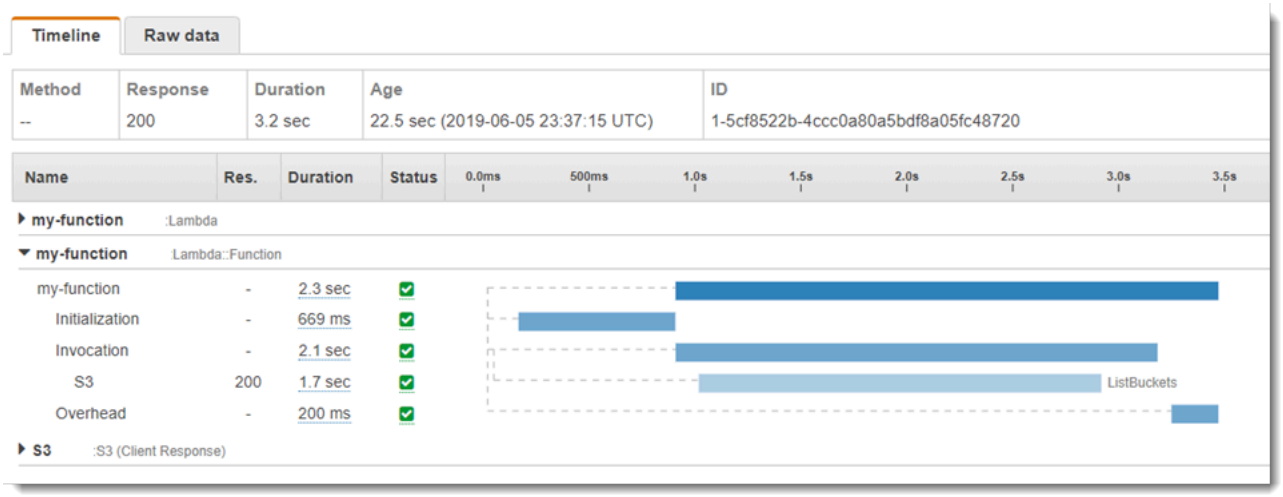
X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste.

> **Note**
>
> La frequenza di campionamento di X-Ray non può essere configurata per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, il che crea 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 denominati my-function, ma uno ha un'origine di `AWS::Lambda` e l'altro ha un'origine di. `AWS::Lambda::Function` Se il `AWS::Lambda` segmento mostra un errore, il servizio Lambda presentava un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione presentava un problema.



Questo esempio espande il `AWS::Lambda::Function` segmento per mostrarne i tre 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.

È possibile strumentare il codice del gestore per registrare i metadati e tracciare le chiamate a valle. Per registrare dettagli sulle chiamate effettuate dal gestore ad altre risorse e servizi, utilizzare l'SDK for Ruby X-Ray. Per ottenere l'SDK, aggiungere il pacchetto `aws-xray-sdk` alle dipendenze dell'applicazione.

Example blank-ruby/function/Gemfile

```
# Gemfile
source 'https://rubygems.org'

gem 'aws-xray-sdk', '0.11.4'
```

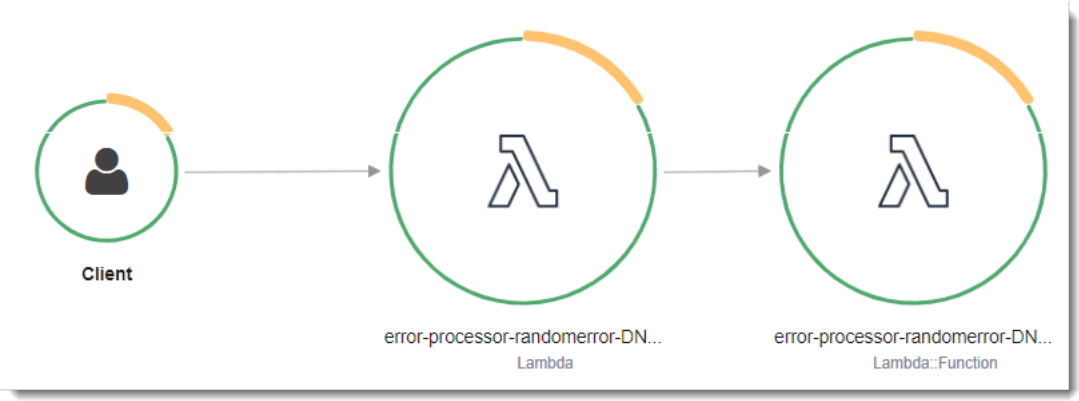
```
gem 'aws-sdk-lambda', '1.39.0'
gem 'test-unit', '3.3.5'
```

Per utilizzare i client AWS SDK, richiedete il `aws-xray-sdk/lambda` modulo dopo aver creato un client nel codice di inizializzazione.

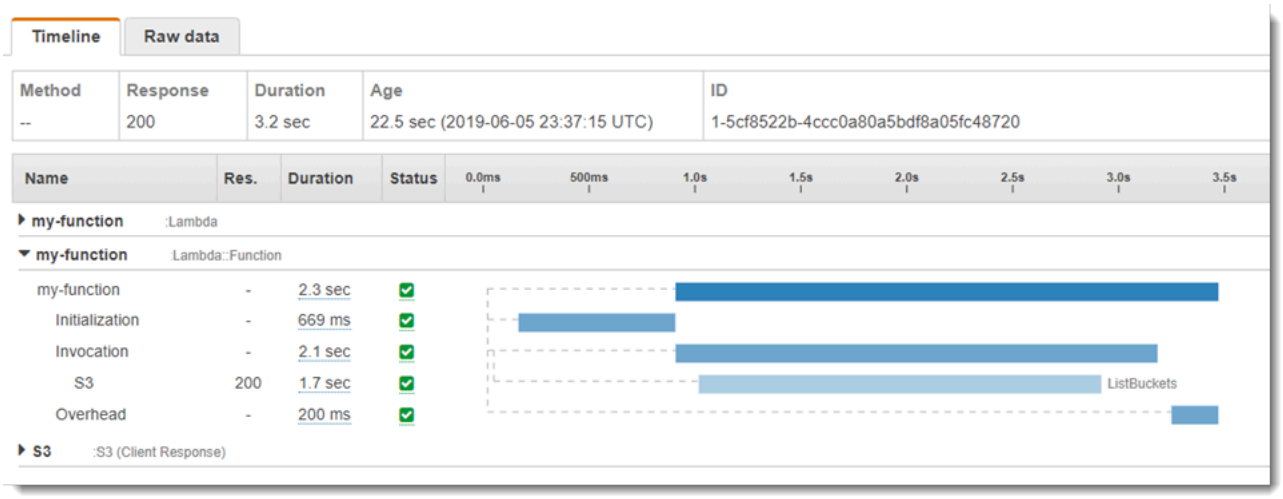
Example blank-ruby/function/lambda_function.rb — Tracciamento di un client SDK AWS

```
# 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 informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, il che crea 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 denominati my-function, ma uno ha un'origine di `AWS::Lambda` e l'altro ha un'origine di. `AWS::Lambda::Function` Se il `AWS::Lambda` segmento mostra un errore, il servizio Lambda presentava un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione presentava un problema.



Questo esempio espande il `AWS::Lambda::Function` segmento per mostrarne i tre 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.

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta X-Ray SDK for Ruby nella Developer Guide. AWS X-Ray

Sections

- Abilitazione del tracciamento attivo con l'API Lambda
- Abilitazione del tracciamento attivo con AWS 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 o AWS SDK, utilizza le seguenti operazioni API:

- UpdateFunctionConfigurazione

- GetFunctionConfigurazione
- CreateFunction

Il AWS CLI comando di esempio seguente abilita il tracciamento attivo 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 del tracciamento attivo con AWS CloudFormation

Per attivare il tracciamento su una `AWS::Lambda::Function` risorsa in un AWS 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), utilizzate la `Tracing` proprietà.

Example template.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
```

```
      ...
```

Memorizzazione delle dipendenze di runtime in un layer

Se utilizzi X-Ray SDK per strumentare i client AWS SDK del codice della funzione, il pacchetto di distribuzione può diventare piuttosto grande. Per evitare di caricare dipendenze di runtime ogni volta che si aggiorna il codice della funzione, includere l'SDK X-Ray in un livello Lambda.

L'esempio seguente mostra una risorsa `AWS::Serverless::LayerVersion` che memorizza l'SDK for Ruby X-Ray.

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 layer per le dipendenze richiede modifiche alla compilazione per generare l'archivio dei layer 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.

Java

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Java 21	`java21`	Amazon Linux 2023			
Java 17	`java17`	Amazon Linux 2			
Java 11	`java11`	Amazon Linux 2			
Java 8	`java8.al2`	Amazon Linux 2			

Lambda fornisce le seguenti librerie per le funzioni Java:

- 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 for Java 2.0 — L'SDK AWS ufficiale per il linguaggio di programmazione Java.

> **Important**
>
> Non utilizzare componenti privati dell'API JDK, come campi, metodi o classi di tipo privato. I componenti dell'API non pubblici possono cambiare o essere rimossi in qualsiasi aggiornamento e causare l'interruzione dell'applicazione.

Per creare una funzione Java

1. Aprire la console Lambda.
2. Scegli Crea funzione.
3. Configurare le impostazioni seguenti:
   - Nome della funzione: inserisci il nome della funzione.
   - Runtime: scegli Java 21.
4. Scegli Crea funzione.
5. Per configurare un evento di test scegliere Test.
6. Per Event name (Nome evento) immettere **`test`**.
7. Seleziona Salvataggio delle modifiche.
8. Per invocare la funzione, scegliere Test (Testa).

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 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

- Definisci il gestore di funzioni Lambda in Java
- Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR
- Distribuisci funzioni Java Lambda con immagini di container
- Utilizzo dei livelli per le funzioni Java Lambda
- Migliorare le prestazioni di avvio con Lambda SnapStart
- Impostazioni di personalizzazione della funzione Lambda in Java
- AWS Lambda oggetto di contesto in Java
- AWS Lambda registrazione delle funzioni in Java
- Strumentazione del codice Java in AWS Lambda
- Esempi di applicazioni Java per AWS Lambda

Definisci il gestore di funzioni 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.

Il GitHub repository di questa guida fornisce applicazioni di easy-to-deploy esempio che illustrano una varietà di tipi di gestori. Per informazioni dettagliate, consulta la parte finale di questo argomento.

Sections

Gestore di esempio: runtime di Java 17

Nell'esempio seguente di Java 17, una classe denominata `HandlerIntegerJava17` definisce un metodo del gestore denominato `handleRequest`. Il metodo del gestore accetta i seguenti input:

- Un `IntegerRecord`, che è un record Java personalizzato che rappresenta i dati degli eventi. In questo esempio, definiamo `IntegerRecord` come segue:

```
record IntegerRecord(int x, int y, String message) {
}
```

- Un oggetto di contesto, che fornisce i metodi e le proprietà che forniscono le informazioni sull'invocazione, sulla funzione e sull'ambiente di esecuzione.

Supponiamo di voler scrivere una funzione che registri il `message` dall'`IntegerRecord` di input e restituisca la somma di `x` e `y`. Di seguito è riportato il codice della funzione:

Example HandlerIntegerJava17.java

```
package example;
```

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;

// Handler value: example.HandlerInteger
public class HandlerIntegerJava17 implements RequestHandler<IntegerRecord, Integer>{

  @Override
  /*
   * Takes in an InputRecord, which contains two integers and a String.
   * Logs the String, then returns the sum of the two Integers.
   */
  public Integer handleRequest(IntegerRecord event, Context context)
  {
    LambdaLogger logger = context.getLogger();
    logger.log("String found: " + event.message());
    return event.x() + event.y();
  }
}

record IntegerRecord(int x, int y, String message) {
}
```

Specifica quale metodo Lambda deve richiamare impostando il parametro del gestore sulla configurazione della funzione. È possibile specificare il gestore nei seguenti formati:

- *package*.*Class*::*method*: formato completo . Ad esempio: `example.Handler::handleRequest`.
- *package*.*Class*: formato abbreviato per le classi che implementano un'interfaccia del gestore. Ad esempio: `example.Handler`.

Quando Lambda richiama il gestore, il runtime Lambda riceve un evento come stringa in formato JSON e lo converte in un oggetto. Nell'esempio precedente, un evento di esempio potrebbe essere simile al seguente:

Example event.json

```
{
  "x": 1,
  "y": 20,
```

```
  "message": "Hello World!"
}
```

Puoi salvare questo file e testare la tua funzione localmente con il seguente comando AWS Command Line Interface (CLI):

```
aws lambda invoke --function-name function_name --payload file://event.json out.json
```

Gestore di esempio: runtime di Java 11 e versioni precedenti

Lambda supporta i record nei runtime di Java 17 e successivi. In tutti i runtime di Java, è possibile utilizzare una classe per rappresentare i dati degli eventi. L'esempio seguente utilizza un elenco di numeri interi e un oggetto di contesto come input e restituisce la somma di tutti i numeri interi nell'elenco.

Example Handler.java

Nell'esempio seguente, una classe denominata `Handler` definisce un metodo del gestore denominato `handleRequest`. Il metodo del gestore accetta un oggetto evento e contesto come input e restituisce una stringa.

Example HandlerList.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.List;

// Handler value: example.HandlerList
public class HandlerList implements RequestHandler<List<Integer>, Integer>{

  @Override
  /*
   * Takes a list of Integers and returns its sum.
   */
  public Integer handleRequest(List<Integer> event, Context context)
  {
    LambdaLogger logger = context.getLogger();
```

```
    logger.log("EVENT TYPE: " + event.getClass().toString());
    return event.stream().mapToInt(Integer::intValue).sum();
  }
}
```

Per altri esempi, consulta la pagina Codice del gestore di esempio.

Codice di inizializzazione

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 un'invocazione e l'altra e possono essere riutilizzate dal gestore migliaia di volte. È possibile aggiungere il codice di inizializzazione al di fuori del metodo del gestore per risparmiare tempo di calcolo e riutilizzare le risorse tra più invocazioni.

Nell'esempio seguente, il codice di inizializzazione del client non rientra nel metodo del gestore principale. Il runtime inizializza il client prima che la funzione esegua il suo primo evento. Gli eventi successivi sono molto più veloci perché Lambda non ha bisogno di inizializzare nuovamente il client.

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;

import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.GetAccountSettingsResponse;
import software.amazon.awssdk.services.lambda.model.LambdaException;

// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String,String>, String> {

    private static final LambdaClient lambdaClient = LambdaClient.builder().build();

    @Override
    public String handleRequest(Map<String,String> event, Context context) {

        LambdaLogger logger = context.getLogger();
```

```
        logger.log("Handler invoked");

        GetAccountSettingsResponse response = null;
        try {
            response = lambdaClient.getAccountSettings();
        } catch(LambdaException e) {
            logger.log(e.getMessage());
        }
        return response != null ? "Total code size for your account is " +
 response.accountLimit().totalCodeSize() + " bytes" : "Error";
    }
}
```

Scelta dei tipi di input e di output

Specificare il tipo di oggetto a cui l'evento si mappa nella firma del metodo del gestore. Nell'esempio precedente, il runtime di Java deserializza l'evento in un tipo che implementa l'interfaccia `Map<String,String>`. tring-to-string Le mappe S funzionano per eventi flat come i seguenti:

Example Event.json – Dati meteo

```
{
  "temperatureK": 281,
  "windKmh": -3,
  "humidityPct": 0.55,
  "pressureHPa": 1020
}
```

Tuttavia, il valore di ogni campo deve essere una stringa o un numero. Se l'evento include un campo con un oggetto come valore, il runtime non può deserializzarlo e restituisce un errore.

Scegliere un tipo di input che funzioni con i dati degli eventi elaborati dalla funzione. È possibile utilizzare un tipo di base, un tipo generico o un tipo ben definito.

Tipi di input

- `Integer`, `Long`, `Double` e così via. – L' evento è un numero senza formattazione aggiuntiva, ad esempio `3.5`. Il runtime 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 (senza virgolette) in un oggetto `String`.

- *Type*, `Map<String,`*Type*`>` e così via. – L'evento è un oggetto JSON. Il runtime lo deserializza in un oggetto del tipo o dell'interfaccia specificati.
- `List<Integer>`, `List<String>`, `List<Object>` e così via. – 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 dai AWS servizi, usa i tipi nella libreria aws-lambda-java-events.

Se si definisce il proprio tipo di input, dovrebbe essere un vecchio oggetto Java (POJO) deserializzabile e mutabile, con un costruttore predefinito e proprietà per ogni campo nell'evento. Le chiavi nell'evento che non si mappano a una proprietà e le proprietà che non sono incluse nell'evento vengono eliminate senza errori.

Il tipo di output può essere un oggetto o `void`. Il runtime serializza i valori restituiti in testo. Se l'output è un oggetto con campi, il runtime lo serializza in un documento JSON. Se è un tipo che esegue il wrapping di un valore primitivo, il runtime restituisce una rappresentazione testuale di tale valore.

Interfacce del gestore

La libreria aws-lambda-java-core definisce due interfacce per i metodi del gestore. Utilizzare le interfacce fornite per semplificare la configurazione del gestore e convalidare la firma del metodo del gestore in fase di compilazione.

- com.amazonaws.services.lambda.runtime. RequestHandler
- com.amazonaws.services.lambda.runtime. RequestStreamGestore

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. Quando si utilizza questa interfaccia, il runtime Java deserializza l'evento in un 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.

Example Handler.java – Interfaccia del gestore

```
// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String,String>, String>{
```

```
  @Override
  public String handleRequest(Map<String,String> event, Context context)
```

Per utilizzare la propria serializzazione, 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.

Nell'esempio seguente vengono utilizzati tipi di lettore e di writer memorizzati per utilizzare i flussi di input e output.

Example HandlerStream.java

```
import com.amazonaws.services.lambda.runtime.Context
import com.amazonaws.services.lambda.runtime.LambdaLogger
import com.amazonaws.services.lambda.runtime.RequestStreamHandler
...
// Handler value: example.HandlerStream
public class HandlerStream implements RequestStreamHandler {
  @Override
  /*
   * Takes an InputStream and an OutputStream. Reads from the InputStream,
   * and copies all characters to the OutputStream.
   */
  public void handleRequest(InputStream inputStream, OutputStream outputStream, Context
 context) throws IOException
  {
    LambdaLogger logger = context.getLogger();
    BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream,
 Charset.forName("US-ASCII")));
    PrintWriter writer = new PrintWriter(new BufferedWriter(new
 OutputStreamWriter(outputStream, Charset.forName("US-ASCII"))));
    int nextChar;
    try {
      while ((nextChar = reader.read()) != -1) {
        outputStream.write(nextChar);
      }
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      reader.close();
      String finalString = writer.toString();
      logger.log("Final string result: " + finalString);
      writer.close();
```

```
    }
  }
}
```

Codice di esempio del gestore

L' GitHub archivio di questa guida include applicazioni di esempio che dimostrano l'uso di vari tipi di gestori e interfacce. Ogni applicazione di esempio include script per facilitare la distribuzione e la pulizia, un AWS SAM modello e risorse di supporto.

Applicazioni Lambda di esempio in Java

- java17-examples: una funzione Java che dimostra come utilizzare un record Java per rappresentare un oggetto di dati dell'evento di input.
- 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' AWS SDK 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.
- Utilizza API Gateway per richiamare una funzione Lambda: una funzione Java che esegue la scansione di una tabella Amazon DynamoDB che contiene informazioni sui dipendenti. Quindi utilizza Amazon Simple Notification Service per inviare un messaggio di testo ai dipendenti per festeggiare i loro anniversari di lavoro. Questo esempio usa API Gateway per richiamare la funzione.

Le `s3-java` applicazioni `java-events` and accettano un evento AWS di servizio come input e restituiscono una stringa. L'applicazione `java-basic` include vari tipi di gestori:

- Handler.java – Accetta input `Map<String,String>`.
- HandlerInteger.java — Accetta un input `Integer` come.
- HandlerList.java — Accetta un input `List<Integer>` come input.
- HandlerStream.java — Accetta un `InputStream` e `OutputStream` come input.
- HandlerString.java — Accetta un `String` come input.

- HandlerWeatherData.java: accetta un tipo personalizzato come input.

Per testare diversi tipi di gestore, è sufficiente modificare il valore del gestore nel AWS SAM modello. Per istruzioni dettagliate, consultare il file readme dell'applicazione di esempio.

Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR

Il codice della 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, quindi utilizzare il pacchetto di distribuzione per distribuire il codice della funzione su (). AWS Lambda AWS Command Line Interface AWS CLI

Sections

Prerequisiti

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- AWS CLI — Configurazione rapida con `aws configure`

Strumenti e librerie

Lambda fornisce le seguenti librerie per le funzioni Java:

- com.amazonaws: aws-lambda-java-core (obbligatorio) — Definisce le interfacce del metodo del gestore e l'oggetto di contesto 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 provenienti da servizi che richiamano funzioni Lambda.
- com.amazonaws: aws-lambda-java-log 4j2 — Una libreria di appender per Apache Log4j 2 che puoi utilizzare per aggiungere l'ID della richiesta per la chiamata corrente ai log delle funzioni.
- AWS SDK for Java 2.0 — L'SDK AWS ufficiale per il linguaggio di programmazione Java.

Queste librerie sono disponibili tramite Maven Central Repository. Aggiungile 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'
}
```

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>
  </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 Livelli Lambda.

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
```

Creare un livello Java per dipendenze

> **Note**
>
> L'utilizzo di livelli con funzioni in un linguaggio compilato come Java potrebbe non offrire gli stessi vantaggi di un linguaggio interpretato come Python. Siccome Java è un linguaggio compilato, 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. È preferibile, invece, includere qualunque codice condiviso in fase di compilazione per sfruttare le ottimizzazioni integrate del compilatore.

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 distribuzione con Gradle" o the section called "Creazione di un pacchetto di distribuzione con Maven".

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 `PATH` variabile raccolga il contenuto del layer, il file Layer .zip dovrebbe avere le sue dipendenze nei seguenti percorsi di cartella:

- `java/lib` (CLASSPATH)

Ad esempio, la struttura del file .zip del livello potrebbe essere simile alla seguente:

```
jackson.zip
# java/lib/jackson-core-2.2.3.jar
```

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.

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. In caso contrario, Lambda potrebbe non decomprimere correttamente il file JAR, facendo sì che il file `MANIFEST.MF` 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'appender 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>
        <dependencies>
          <dependency>
            <groupId>com.github.edwgiz</groupId>
            <artifactId>maven-shade-plugin.log4j2-cachefile-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 AWS Management Console, consulta la Guida introduttiva ad Amazon S3. Per caricare file utilizzando la AWS CLI, consulta Move objects nella Guida per l'AWS CLI utente.

> **Note**
>
> Non è possibile modificare il tipo di pacchetto di distribuzione (.zip o immagine del contenitore) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine del contenitore 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. Selezionare 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. Usa la funzione create-function e update-function-codei comandi per distribuire 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 su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

> **Note**
>
> Se carichi il tuo file.zip o JAR da un bucket Amazon S3 utilizzando AWS CLI il, il bucket deve trovarsi nella stessa posizione della Regione AWS tua funzione.

Per creare una nuova funzione utilizzando un file.zip o JAR 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 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 java21 --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 java21 --handler example.handler \
--role arn:aws:iam::123456789012:role/service-role/my-lambda-role \
--code S3Bucket=DOC-EXAMPLE-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 DOC-EXAMPLE-BUCKET --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Caricamento di un pacchetto di distribuzione con AWS SAM

È possibile utilizzarlo AWS SAM per automatizzare le distribuzioni del codice funzionale, della configurazione e delle dipendenze. AWS SAM è un'estensione AWS 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: java21
      Description: Java function
      MemorySize: 512
      Timeout: 10
      # Function's execution role
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambda_ReadOnlyAccess
        - AWSXrayWriteOnlyAccess
        - AWSLambdaVPCAccessExecutionRole
      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 AWS 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

- java17-examples: una funzione Java che dimostra come utilizzare un record Java per rappresentare un oggetto di dati dell'evento di input.
- 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 l'ultima versione della aws-lambda-java-eventslibreria (3.0.0 e successive). Questi esempi non richiedono l' AWS SDK 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.
- Utilizza API Gateway per richiamare una funzione Lambda: una funzione Java che esegue la scansione di una tabella Amazon DynamoDB che contiene informazioni sui dipendenti. Quindi utilizza Amazon Simple Notification Service per inviare un messaggio di testo ai dipendenti per festeggiare i loro anniversari di lavoro. Questo esempio usa API Gateway per richiamare la funzione.

Distribuisci funzioni Java Lambda con immagini di container

Esistono tre modi per creare un'immagine di container per una funzione Lambda in Java:

- Utilizzo di un'immagine di base per Java AWS

  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.
- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  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 per implementare un runtime personalizzato. Per rendere l'immagine compatibile con Lambda, devi includere il client di interfaccia di runtime per Java nell'immagine.
- Utilizzo di un'immagine non di base AWS

  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

- 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	Definizione come obsoleto
21	Java 21	Amazon Linux 2023	Dockerfile per Java 2.1 su GitHub	
17	Java 17	Amazon Linux 2	Dockerfile per Java 17 attivo GitHub	
11	Java 11	Amazon Linux 2	Dockerfile per Java 11 attivo GitHub	
8.al2	Java 8	Amazon Linux 2	Dockerfile per Java 8 attivo GitHub	

Repository Amazon ECR: gallery.ecr.aws/lambda/java

Le immagini di base Java 21 e versioni successive si basano sull'immagine container minima di Amazon Linux 2023. Le immagini di base precedenti utilizzavano 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`.

Le immagini basate su AL2023 utilizzano `microdnf` (symlinked as`dnf`) come gestore di pacchetti anzichéyum, 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 immagini basate su AL2023, 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 la sezione Introduzione al runtime di Amazon Linux 2023 per AWS Lambda sul AWS Compute Blog.

> **Note**
>
> Per eseguire immagini basate su AL2023 localmente, incluso with AWS Serverless Application Model (AWS SAM), devi usare la versione Docker 20.10.10 o successiva.

Utilizzo di un'immagine di base per Java AWS

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Java (ad esempio, Amazon Corretto)
- Docker (versione minima 20.10.10 per immagini di base Java 21 e successive)
- Apache Maven o Gradle
- AWS Command Line Interface () versione 2 AWS CLI

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, consulta aws-sdk-java-v2/services on. GitHub
   - region — Il Regione AWS luogo in cui si desidera creare la funzione Lambda.
   - groupId: lo spazio dei nomi 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 for 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.

   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`.

```
docker build --platform linux/amd64 -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 variabile di `{LAMBDA_TASK_ROOT}` ambiente definita da Lambda.
   - Imposta l'argomento `CMD` specificando il gestore della funzione Lambda.

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`.

```
docker build --platform linux/amd64 -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 comando seguente: `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"
   ```

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:
   - Sostituisci `docker-image:test` con il nome e il tag della tua immagine Docker.
   - 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 image in un digest di immagini specifico. Ciò significa che se punti il tag immagine utilizzato per distribuire 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.

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 di runtime Lambda, 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 multipiattaforma 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)
- Docker
- Apache Maven
- AWS Command Line Interface (AWS CLI) versione 2

Creazione di un'immagine da un'immagine di base alternativa

1. Crea un progetto Maven. I parametri seguenti sono obbligatori:

   - groupId: lo spazio dei nomi 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.

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`.

```
docker build --platform linux/amd64 -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. È possibile creare l'emulatore nell'immagine o utilizzare la seguente procedura per installarlo sul 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. Tieni 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 commando 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:
   - Sostituisci `docker-image:test` con il nome e il tag della tua immagine Docker.
   - 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 image in un digest di immagini specifico. Ciò significa che se punti il tag immagine utilizzato per distribuire 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.

Utilizzo dei livelli per le funzioni Java Lambda

Un layer Lambda è un archivio di file.zip che contiene codice o dati supplementari. 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. Package del contenuto del layer. Ciò significa creare un archivio di file.zip che contenga le dipendenze da utilizzare nelle funzioni.
2. Crea il livello in Lambda.
3. Aggiungi il layer alle tue funzioni.

Questo argomento contiene passaggi e linee guida su come impacchettare e creare correttamente un layer Java Lambda con dipendenze di librerie esterne.

Argomenti

Prerequisiti

Per seguire i passaggi di questa sezione, è necessario disporre di quanto segue:

- Java 2.1
- Apache Maven 3.8.6 o versione successiva
- AWS Command Line InterfaceAWS CLI() versione 2

> **Note**
>
> Assicurati che la versione Java a cui fa riferimento Maven sia la stessa della versione Java della funzione che intendi distribuire. Ad esempio, per una funzione Java 21, il `mvn -v` comando dovrebbe elencare la versione Java 21 nell'output:

```
Apache Maven 3.8.6
...
Java version: 21.0.2, vendor: Oracle Corporation, runtime: /Library/Java/
JavaVirtualMachines/jdk-21.jdk/Contents/Home
...
```

In questo argomento, facciamo riferimento all'applicazione di `layer-java`esempio nel repository awsdocs GitHub . Questa applicazione contiene script che scaricano le dipendenze e generano il layer. L'applicazione contiene anche una funzione corrispondente che utilizza le dipendenze dal livello. Dopo aver creato un livello, potete implementare e richiamare la funzione corrispondente per verificare che tutto funzioni correttamente. Poiché utilizzate il runtime Java 21 per le funzioni, i livelli devono essere compatibili anche con Java 21.

L'applicazione `layer-java` di esempio contiene un singolo esempio all'interno di due sottodirectory. La `layer` directory contiene un `pom.xml` file che definisce le dipendenze dei livelli e gli script per generare il livello. La `function` directory contiene una funzione di esempio per verificare il funzionamento del layer. Questo tutorial spiega come creare e impacchettare questo layer.

Compatibilità del livello Java con Amazon Linux

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.

Il codice Java è progettato per essere indipendente dalla piattaforma, quindi puoi impacchettare i livelli sul tuo computer locale anche se non utilizza un ambiente Linux. Dopo aver caricato il layer Java su Lambda, sarà ancora compatibile con Amazon Linux.

Percorsi di livello per i runtime Java

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 `PATH` variabile raccolga il contenuto del layer, il file layer.zip dovrebbe avere le sue dipendenze nei seguenti percorsi di cartella:

- `java/lib`

Ad esempio, il file.zip del layer risultante creato in questo tutorial ha la seguente struttura di directory:

```
layer_content.zip
# java
    # lib
        # layer-java-layer-1.0-SNAPSHOT.jar
```

Il file `layer-java-layer-1.0-SNAPSHOT.jar` JAR (un uber-jar che contiene tutte le nostre dipendenze richieste) si trova correttamente nella directory. `java/lib` Ciò garantisce che Lambda possa localizzare la libreria durante le chiamate delle funzioni.

Imballaggio del contenuto del livello

In questo esempio, si impacchettano le seguenti due librerie Java in un unico file JAR:

- aws-lambda-java-core— Un set minimo di definizioni di interfaccia per lavorare con Java in AWS Lambda
- Jackson: una popolare suite di strumenti di elaborazione dati, in particolare per lavorare con JSON.

Completate i seguenti passaggi per installare e impacchettare il contenuto del layer.

Per installare e impacchettare il contenuto del layer

1. Clona il `aws-lambda-developer-guide` GitHub repository, che contiene il codice di esempio necessario nella `sample-apps/layer-java` directory.

   ```
   git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
   ```

2. Vai alla `layer` directory dell'app di `layer-java` esempio. Questa directory contiene gli script che utilizzate per creare e impacchettare correttamente il layer.

   ```
   cd aws-lambda-developer-guide/sample-apps/layer-java/layer
   ```

3. Esaminate il `pom.xml`file. Nella `<dependencies>` sezione, definisci le dipendenze che desideri includere nel layer, vale a dire le `jackson-databind` librerie `aws-lambda-java-core` and. È possibile aggiornare questo file per includere tutte le dipendenze che si desidera includere nel proprio layer.

Example pom.xml

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-lambda-java-core</artifactId>
        <version>1.2.3</version>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.17.0</version>
    </dependency>
</dependencies>
```

> **Note**
>
> La `<build>` sezione di questo `pom.xml` file contiene due plugin. maven-compiler-pluginCompila il codice sorgente. Quindi maven-shade-pluginimpacchetta i tuoi artefatti in un unico uber-jar.

4. Assicurati di disporre delle autorizzazioni per eseguire entrambi gli script.

```
chmod 744 1-install.sh && chmod 744 2-package.sh
```

5. Esegui lo `1-install.sh`script utilizzando il seguente comando:

```
./1-install.sh
```

Questo script viene eseguito `mvn clean install` nella directory corrente. Questo crea l'uber-jar con tutte le dipendenze richieste nella directory. `target/`

Example 1-install.sh

```
mvn clean install
```

6. Esegui lo `2-package.sh`script utilizzando il seguente comando:

```
./2-package.sh
```

Questo script crea la struttura di `java/lib` directory necessaria per impacchettare correttamente il contenuto del layer. Quindi copia l'uber-jar dalla `/target` directory nella directory appena creata. `java/lib` Infine, lo script comprime il contenuto della `java` directory in un file denominato. `layer_content.zip` Questo è il file.zip per il livello. È possibile decomprimere il file e verificare che contenga la struttura di file corretta, come mostrato nella the section called “Percorsi di livello per i runtime Java” sezione.

Example 2-package.sh

```
mkdir java
mkdir java/lib
cp -r target/layer-java-layer-1.0-SNAPSHOT.jar java/lib/
zip -r layer_content.zip java
```

Creazione del livello

In questa sezione, prendi il `layer_content.zip` file che hai generato nella sezione precedente e lo carichi come layer Lambda. È possibile caricare un layer utilizzando AWS Management Console o l'API Lambda tramite AWS Command Line Interface ()AWS CLI. Quando caricate il file Layer .zip, nel PublishLayerVersion AWS CLI comando seguente, specificate `java21` come runtime compatibile e `arm64` come architettura compatibile.

```
aws lambda publish-layer-version --layer-name java-jackson-layer \
    --zip-file fileb://layer_content.zip \
    --compatible-runtimes java21 \
    --compatible-architectures "arm64"
```

Dalla risposta, nota il`LayerVersionArn`, che `arn:aws:lambda:us-east-1:`*`123456789012`*`:layer:java-jackson-layer:1` assomiglia a. Avrai bisogno di questo Amazon Resource Name (ARN) nel passaggio successivo di questo tutorial, quando aggiungi il layer alla tua funzione.

Aggiungere il layer alla tua funzione

In questa sezione, distribuisci una funzione Lambda di esempio che utilizza la libreria Jackson nel suo codice funzione, quindi colleghi il layer. Per distribuire la funzione, è necessario un. the section called “Ruolo di esecuzione (autorizzazioni per le funzioni di accedere ad altre risorse)” Se non disponi di un ruolo di esecuzione esistente, segui i passaggi nella sezione comprimibile. Altrimenti, passa alla sezione successiva per distribuire la funzione.

(Facoltativo) Crea un ruolo di esecuzione

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 AWSLambdaBasicExecutionRolepolitica dispone delle autorizzazioni necessarie alla funzione per scrivere i log in Logs. CloudWatch

Per distribuire la funzione Lambda

1. Passa alla directory `function/`. Se ti trovi attualmente nella `layer/` directory, esegui il seguente comando:

   ```
   cd ../function
   ```

2. Esamina il codice della funzione. La funzione accetta un input `Map<String, String>` as e utilizza Jackson per scrivere l'input come stringa JSON prima di convertirlo in un oggetto Java F1Car predefinito. Infine, la funzione utilizza i campi dell'oggetto F1Car per costruire una stringa restituita dalla funzione.

   ```
   package example;

   import com.amazonaws.services.lambda.runtime.Context;
   ```

```
import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.IOException;
import java.util.Map;

public class Handler {

    public String handleRequest(Map<String, String> input, Context context) throws
 IOException {
        // Parse the input JSON
        ObjectMapper objectMapper = new ObjectMapper();
        F1Car f1Car =
 objectMapper.readValue(objectMapper.writeValueAsString(input), F1Car.class);

        StringBuilder finalString = new StringBuilder();
        finalString.append(f1Car.getDriver());
        finalString.append(" is a driver for team ");
        finalString.append(f1Car.getTeam());
        return finalString.toString();
    }
}
```

3. Costruisci il progetto usando il seguente comando Maven:

```
mvn package
```

Questo comando produce un file JAR nella `target/` directory denominata. `layer-java-function-1.0-SNAPSHOT.jar`

4. Implementa la funzione. Nel AWS CLI comando seguente, sostituite il `--role` parametro con il vostro ruolo di esecuzione ARN:

```
aws lambda create-function --function-name java_function_with_layer \
    --runtime java21 \
    --architectures "arm64" \
    --handler example.Handler::handleRequest \
    --timeout 30 \
    --role arn:aws:iam::123456789012:role/lambda-role \
    --zip-file fileb://target/layer-java-function-1.0-SNAPSHOT.jar
```

(Facoltativo) Richiamate la funzione senza collegare un livello

A questo punto, puoi facoltativamente provare a richiamare la tua funzione prima di collegare il livello. Se provate in questo modo, dovreste ottenere una, `ClassNotFoundException` perché la vostra funzione non può fare riferimento al pacchetto. `requests` Per richiamare la tua funzione, usa il seguente AWS CLI comando:

```
aws lambda invoke --function-name java_function_with_layer \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "driver": "Max Verstappen", "team": "Red Bull" }' response.json
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

Per visualizzare l'errore specifico, aprite il `response.json` file di output. Dovresti vedere un messaggio `ClassNotFoundException` con il seguente messaggio di errore:

```
"errorMessage":"com.fasterxml.jackson.databind.ObjectMapper","errorType":"java.lang.ClassNotFou
```

Quindi, collega il livello alla tua funzione. Nel AWS CLI comando seguente, sostituite il `--layers` parametro con la versione del layer ARN che avete notato in precedenza:

```
aws lambda update-function-configuration --function-name java_function_with_layer \
    --cli-binary-format raw-in-base64-out \
    --layers "arn:aws:lambda:us-east-1:123456789012:layer:java-jackson-layer:1"
```

Infine, provate a richiamare la vostra funzione usando il seguente comando: AWS CLI

```
aws lambda invoke --function-name java_function_with_layer \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "driver": "Max Verstappen", "team": "Red Bull" }' response.json
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{
```

```
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Ciò indica che la funzione è stata in grado di utilizzare la dipendenza Jackson per eseguire correttamente la funzione. È possibile verificare che il `response.json` file di output contenga la stringa restituita corretta:

```
"Max Verstappen is a driver for team Red Bull"
```

(Facoltativo) Pulisci le tue 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 ai tuoi Account AWS.

Per eliminare il layer Lambda

1. Apri la pagina Layers (Livelli) nella console Lambda.
2. Seleziona il livello che hai creato.
3. Scegliete Elimina, quindi scegliete nuovamente 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 **`delete`** nel campo di immissione testo e scegli Delete (Elimina).

Migliorare le prestazioni di avvio con Lambda SnapStart

Lambda SnapStart per Java può migliorare le prestazioni di avvio per le applicazioni sensibili alla latenza fino a 10 volte senza costi aggiuntivi, in genere senza modifiche al codice della funzione. 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 utilizza lo snapshot di una microVM Firecracker della memoria e dello stato del disco dell'ambiente di esecuzione inizializzato, crittografa lo snapshot e lo memorizza nella cache per l'accesso a bassa latenza. 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.

Argomenti

- Massimizza le prestazioni Lambda SnapStart

Funzionalità e limitazioni supportate

SnapStart supporta i runtime gestiti di Java 11 e versioni successive. Altri runtime gestiti (come `nodejs20.x` e `python3.12`), Runtime solo per il sistema operativo e le immagini di container non sono supportati.

SnapStart non supporta la concorrenza fornita, l'architettura arm64, Amazon Elastic File System (Amazon EFS) o lo storage temporaneo superiore a 512 MB.

Con cui lavorare SnapStart, puoi usare la console Lambda, the AWS Command Line Interface (AWS CLI), l'API Lambda,, AWS SDK for Java AWS CloudFormation, AWS Serverless Application Model (AWS SAM) e. AWS Cloud Development Kit (AWS CDK) Per ulteriori informazioni, consulta Attivazione e gestione di Lambda SnapStart.

> **Note**
>
> È possibile utilizzare SnapStart solo versioni di funzioni pubblicate e alias che rimandano a versioni. Non è possibile utilizzare SnapStart sulla versione non pubblicata di una funzione ($LATEST).

Regioni supportate

SnapStart è disponibile nelle seguenti versioni: Regioni AWS

- Stati Uniti orientali (Virginia settentrionale)
- Stati Uniti orientali (Ohio)
- Stati Uniti occidentali (California settentrionale)
- Stati Uniti occidentali (Oregon)
- Africa (Città del Capo)
- Asia Pacific (Hong Kong)
- Asia Pacifico (Mumbai)
- Asia Pacific (Hyderabad)
- Asia Pacifico (Tokyo)

- Asia Pacifico (Seul)
- Asia Pacifico (Osaka-Locale)
- Asia Pacifico (Singapore)
- Asia Pacifico (Sydney)
- Asia Pacifico (Giacarta)
- Asia Pacifico (Melbourne)
- Canada (Centrale)
- Europa (Stoccolma)
- Europa (Francoforte)
- Europa (Zurigo)
- Europa (Irlanda)
- Europe (London)
- Europa (Parigi)
- Europa (Milano)
- Europa (Spagna)
- Medio Oriente (Emirati Arabi Uniti)
- Medio Oriente (Bahrein)
- Sud America (San Paolo)

Considerazioni sulla compatibilità

Con SnapStart, Lambda utilizza una singola istantanea come stato iniziale per più ambienti di esecuzione. Se la 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 unici 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 riprendono 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

Non ci sono 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. 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.

I costi di durata si applicano al codice eseguito nel gestore della funzione, al codice di inizializzazione dichiarato all'esterno del gestore, al tempo necessario per il caricamento del runtime (JVM) e a qualsiasi codice eseguito in un hook di runtime. Per ulteriori informazioni su come Lambda calcola la durata, consulta Monitoraggio per Lambda SnapStart.

Per le funzioni configurate con SnapStart, Lambda ricicla periodicamente gli ambienti di esecuzione ed esegue nuovamente il codice di inizializzazione. Per la resilienza, Lambda crea snapshot in più zone di disponibilità. Gli addebiti si applicano ogni volta che Lambda esegue nuovamente il codice di inizializzazione in un'altra zona di disponibilità. Per ulteriori informazioni su come Lambda calcola gli addebiti, consulta Prezzi di AWS Lambda.

Confronto tra Lambda SnapStart e la concorrenza fornita

Sia la concorrenza Lambda SnapStart che quella fornita possono ridurre gli avviamenti a freddo e le latenze anomale quando una funzione aumenta. SnapStart ti aiuta a migliorare le prestazioni di avvio fino a 10 volte senza costi aggiuntivi. La simultaneità con provisioning mantiene le funzioni inizializzate e pronte a rispondere entro i 100 millisecondi. La configurazione della concorrenza fornita comporta addebiti a carico dell'utente. Account AWS Utilizza la simultaneità con provisioning se

l'applicazione ha requisiti di latenza rigorosi per l'avvio a freddo. Non è possibile utilizzare entrambe le funzioni SnapStart e la concorrenza predisposta nella stessa versione della funzione.

> **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.

Risorse aggiuntive

Oltre a leggere gli altri argomenti di questo capitolo, ti consigliamo anche di provare Starting up faster with AWS Lambda SnapStart workshop e di guardare la sessione Fast cold start for your Java functions di AWS re:Invent 2022.

Attivazione e gestione di Lambda SnapStart

Per utilizzarlo SnapStart, attivalo SnapStart su una funzione Lambda nuova o esistente. Quindi, pubblica e richiama una versione della funzione.

Argomenti

Attivazione SnapStart (console)

Attivare SnapStart per 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, per 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

Da attivare SnapStart per una funzione esistente

1. Aggiorna la configurazione della funzione eseguendo il update-function-configurationcomando con l'--snap-startopzione.

```
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. Confermate che SnapStart sia attivata per la versione della funzione eseguendo il get-function-configurationcomando 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 mostra che OptimizationStatusis 0n e State è`Active`, allora SnapStart viene attivata e un'istantanea è disponibile 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'opzione cli-binary-format è necessaria 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 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.

```
aws lambda create-function \
  --function-name my-function \
  --runtime "java21" \
  --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. Confermate che SnapStart sia attivata per la versione della funzione eseguendo il get-function-configurationcomando 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 mostra che OptimizationStatusis 0n e State è`Active`, allora SnapStart viene attivata e un'istantanea è disponibile 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'opzione cli-binary-format è necessaria 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 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 di queste operazioni:
   - Crea una nuova funzione con SnapStart activated utilizzando l'azione CreateFunctionAPI con il SnapStartparametro.
   - Attiva SnapStart per una funzione esistente utilizzando l'UpdateFunctionConfigurationazione con il SnapStartparametro.
2. Pubblica una versione della funzione con l'PublishVersionazione. 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. Verifica che SnapStart sia attivata per la versione della funzione utilizzando l'GetFunctionConfigurationazione. Specificate un numero di versione per confermare che SnapStart è attivata per quella versione. Se la risposta mostra che OptimizationStatusè 0n e State è`Active`, allora SnapStart viene attivata ed è disponibile un'istantanea 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 funzionali

I seguenti stati di funzione possono verificarsi quando si utilizza SnapStart. Possono verificarsi anche quando Lambda ricicla periodicamente l'ambiente di esecuzione ed esegue nuovamente il codice di inizializzazione per una funzione configurata con. SnapStart

- `Pending`: Lambda sta inizializzando il codice e acquisendo uno snapshot dell'ambiente di esecuzione inizializzato. Qualsiasi chiamata o altre operazioni API che operano sulla versione della funzione avranno esito negativo.
- `Active`: la creazione di istantanee è completa ed è possibile richiamare la funzione. Per utilizzarla SnapStart, è necessario richiamare la versione pubblicata della funzione, non quella non pubblicata ($LATEST).
- `Inactive`: la versione della funzione non viene richiamata da 14 giorni. Quando la versione della funzione diventa`Inactive`, Lambda elimina lo snapshot. 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.
- `Failed`: 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. Lambda aggiorna automaticamente gli snapshot con le patch di runtime e sicurezza più recenti.

Utilizzo con SnapStart AWS SDK for Java

Per effettuare chiamate all'SDK AWS dalla funzione, Lambda genera un insieme di credenziali temporaneo assumendo il ruolo di esecuzione della 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 Java utilizza automaticamente le credenziali del contenitore (`AWS_CONTAINER_CREDENTIALS_FULL_URI`and`AWS_CONTAINER_AUTHORIZATION_TOKEN`) anziché le variabili di ambiente della chiave di accesso. Ciò impedisce la scadenza delle credenziali prima che la funzione venga ripristinata.

Utilizzando SnapStart with AWS CloudFormationAWS SAM, e AWS CDK

- AWS CloudFormation: dichiara l'SnapStartentità nel modello.
- AWS Serverless Application Model(AWS SAM): dichiara la SnapStartproprietà nel modello.
- AWS Cloud Development Kit (AWS CDK): Usa il SnapStartPropertytipo.

Eliminazione di snapshot

Lambda elimina gli snapshot quando:

- Si elimina la funzione o la versione della funzione.
- 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 chiamate 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 unici 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. 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. Ti consigliamo invece di generare dati univoci all'interno del gestore delle funzioni o di utilizzare un hook di runtime.

Example : generazione di un ID univoco nel gestore delle funzioni

Il seguente esempio mostra come generare un UUID 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!";
    }
  }
```

Utilizza generatori di numeri pseudocasuali crittograficamente sicuri (CSPRNG)

Se l'applicazione dipende dalla casualità, si consiglia di utilizzare generatori di numeri casuali crittograficamente sicuri (CSPRNG). Il runtime gestito da Lambda per Java include due CSPRNG integrati (OpenSSL 1.0.2 e) che mantengono automaticamente la casualità con. `java.security.SecureRandom` SnapStart Software che ottiene sempre numeri casuali da o con cui mantiene la casualità. `/dev/random` `/dev/urandom` SnapStart

Example — java.security. SecureRandom

L'esempio seguente utilizza `java.security.SecureRandom`, che genera sequenze numeriche univoche anche quando la funzione viene ripristinata da uno snapshot.

```
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!";
    }
  }
```

SnapStart strumento di scansione

Lambda fornisce uno strumento di scansione che aiuta a verificare la presenza di codice che presuppone l'unicità. Lo strumento di SnapStart scansione è un SpotBugsplug-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, consultate il repository aws-lambda-snapstart-java-rules su. GitHub

Per saperne di più sulla gestione dell'unicità con SnapStart, consulta Starting up faster with AWS Lambda SnapStart sul blog di ComputeAWS.

Implementa il codice prima o dopo le istantanee 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 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 usare cRac con un'applicazione di riferimento, consulta 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 `Context` globale predefinito tramite il metodo statico `Core.getGlobalContext()`.

Per ulteriori informazioni su `Context` e `Resource`, consulta Pacchetto org.crac nella documentazione di CRaC.

Completa le seguenti operazioni per implementare gli hook di runtime con il pacchetto org.crac. Il runtime Lambda contiene un'implementazione del contesto CRaC personalizzata che chiama gli hook di runtime prima del checkpoint e dopo il ripristino.

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 dependecies 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 Definisci il gestore di funzioni 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 );
```

Monitoraggio per Lambda SnapStart

Puoi monitorare le tue SnapStart funzioni Lambda utilizzando Amazon CloudWatchAWS X-Ray, e il. API di telemetria Lambda

> **Note**
>
> Le variabili di `AWS_LAMBDA_LOG_STREAM_NAME` ambiente `AWS_LAMBDA_LOG_GROUP_NAME` e non sono disponibili nelle funzioni Lambda SnapStart .

CloudWatch per SnapStart

Esistono alcune differenze con il formato del flusso di CloudWatch log per SnapStart le funzioni:

- Log di inizializzazione: quando viene creato un nuovo ambiente di esecuzione, il `REPORT` non include il campo `Init Duration`. Questo perché Lambda inizializza SnapStart le funzioni quando crei una versione anziché durante la chiamata della funzione. Per SnapStart le funzioni, il `Init Duration` campo 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 `afterRestore`. 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 caricare il runtime (JVM) ed eseguire qualsiasi hook `afterRestore`. Non ti sarà addebitato alcun costo per il tempo necessario per ripristinare uno snapshot.

> **Note**
>
> I costi di durata si applicano al codice eseguito nel gestore della funzione, al codice di inizializzazione dichiarato all'esterno del gestore, al tempo necessario per il caricamento del runtime (JVM) e a qualsiasi codice eseguito in un hook di runtime. Per ulteriori informazioni, consulta SnapStart prezzi.

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 a raggi X per SnapStart

Puoi usare X-Ray per tracciare le richieste verso le funzioni Lambda. SnapStart Esistono alcune differenze con i sottosegmenti X-Ray per quanto riguarda le funzioni: SnapStart

- Non esiste un `Initialization` sottosegmento per le funzioni. SnapStart
- Il sottosegmento `Restore` mostra il tempo impiegato da Lambda per ripristinare uno snapshot, caricare il runtime (JVM) ed eseguire qualsiasi hook di runtime `afterRestore`. 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 SnapStart eventi a: API di telemetria

- `platform.restoreStart`: mostra l'ora in cui è iniziata la fase `Restore`.
- `platform.restoreRuntimeDone`: 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.

- `platform.restoreReport`: 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 API Gateway, puoi utilizzare la IntegrationLatencymetrica per misurare la end-to-end latenza (il tempo che intercorre tra il momento in cui API Gateway 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 la UrlRequestLatencymetrica per misurare la end-to-end latenza (il 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).

Modello di sicurezza per Lambda SnapStart

Lambda SnapStart supporta la crittografia a riposo. Lambda crittografa gli snapshot con una 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 nella funzione o nel update-function-configurationcomando create-function per fornire una chiave gestita AWS KMS dal cliente. 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, consulta Prezzi di AWS Key Management Service.

Quando si elimina una SnapStart funzione o una versione della funzione, tutte le `Invoke` richieste relative a tale funzione o versione della funzione hanno esito negativo. Lambda elimina automaticamente gli snapshot che non vengono richiamati per 14 giorni. 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

> **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.

Per massimizzare i vantaggi di SnapStart, si consiglia di precaricare le classi che contribuiscono alla latenza di avvio nel codice di inizializzazione anziché nel gestore delle funzioni. In questo modo la latenza associata al caricamento intensivo delle classi viene rimossa dal percorso di invocazione, ottimizzando le prestazioni di avvio con. SnapStart

Se non riesci a precaricare le classi durante l'inizializzazione, ti consigliamo di precaricare le classi con invocazioni fittizie. A tale scopo, aggiornate il codice del gestore delle funzioni, come illustrato nell'esempio seguente, tratto dalla funzione pet store del 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);
        }
    }
```

Le migliori pratiche 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. Nella maggior parte dei casi, le connessioni di rete stabilite da un AWS SDK riprendono automaticamente. Per altre connessioni, consigliamo le seguenti best practice.

Ristabilire le 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 `afterRestore`.

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 e tutti gli ambienti di esecuzione restituiscono lo stesso valore per. `hostname` `InetAddress.getLocalHost()` 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 `afterRestore` 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.

Evita di usare la cache DNS Java

Le funzioni Lambda memorizzano già nella cache le risposte DNS. Se utilizzi un'altra cache DNS con SnapStart, potrebbero verificarsi dei timeout di connessione quando la funzione riprende da un'istantanea.

La `java.util.logging.Logger` classe può abilitare indirettamente la cache DNS JVM. Per sovrascrivere le impostazioni predefinite, imposta networkaddress.cache.ttl su 0 prima dell'inizializzazione. `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`networkaddress.cache.negative.ttl`, consigliamo di impostare su 0. `UnknownHostException` È possibile impostare questa proprietà per una funzione Lambda con la variabile di `AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0` ambiente.

La disabilitazione della cache DNS JVM non disabilita la cache DNS gestita da Lambda.

Impostazioni di personalizzazione della funzione Lambda in Java

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

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 più livelli è una funzionalità della macchina virtuale Java (JVM). È possibile utilizzare impostazioni di compilazione specifiche a più livelli per utilizzare al meglio i compilatori JVM (JIT). just-in-time 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.

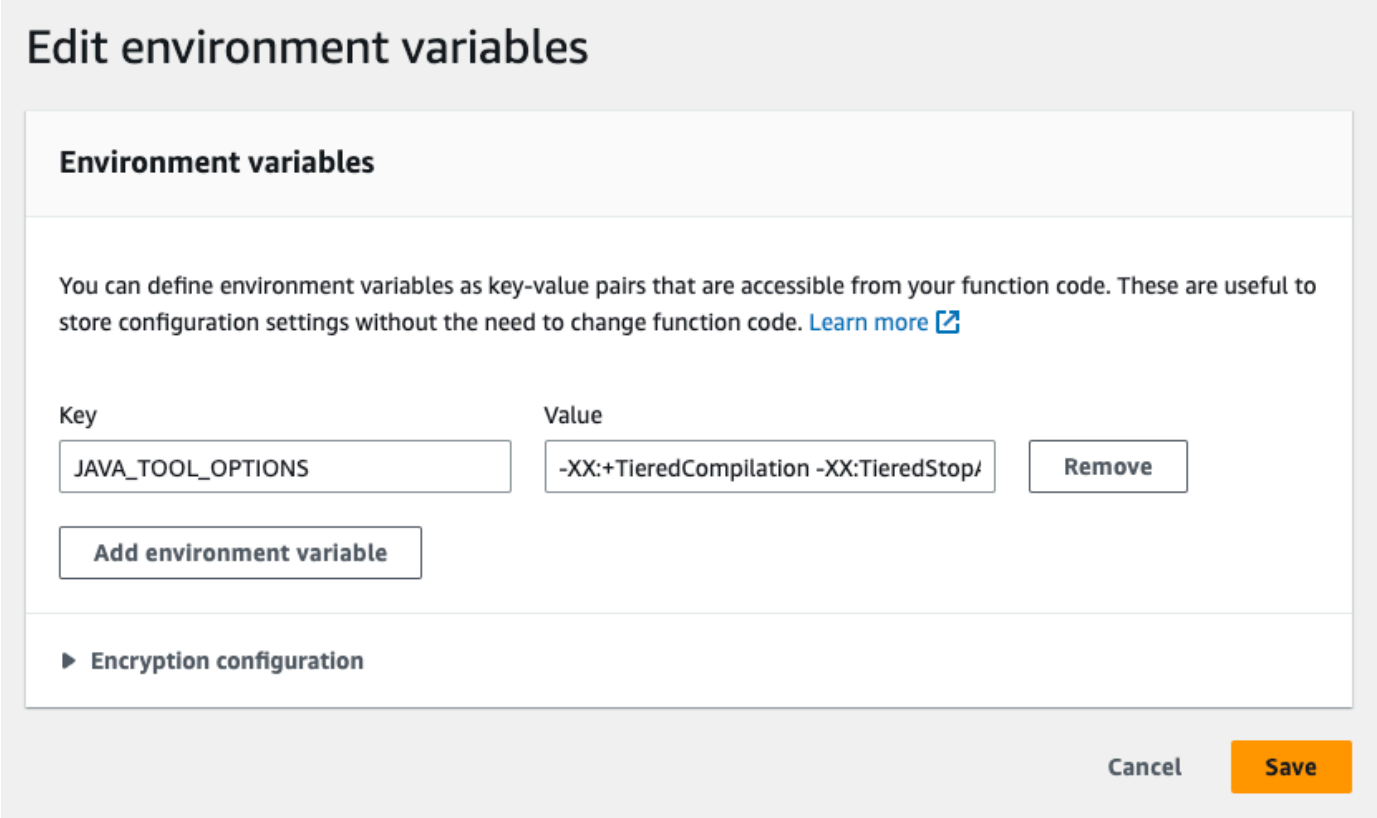
Esistono 5 diversi livelli di compilazione a più livelli. Al livello 0, la JVM interpreta il bytecode Java. Al livello 4, la JVM utilizza il compilatore C2 per analizzare i dati di profilazione raccolti durante l'avvio dell'applicazione. Nel tempo, monitora l'utilizzo del codice per identificare le migliori ottimizzazioni.

La personalizzazione del livello di compilazione a più livelli può aiutarti a ridurre la latenza di avvio a freddo delle funzioni Java. Ad esempio, imposta il livello di compilazione a più livelli su 1 per fare in modo che la JVM utilizzi il compilatore C1. Questo compilatore produce rapidamente codice nativo ottimizzato, ma non genera dati di profilazione e non utilizza mai il compilatore C2.

Nel runtime di Java 17, il flag JVM per la compilazione a più livelli è configurato per interrompersi al livello 1 per impostazione predefinita. Per il runtime di Java 11 e versioni precedenti, puoi impostare il livello di compilazione a più livelli su 1 effettuando le seguenti operazioni:

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`.



7. Selezionare Salva.

> **Note**
>
> Puoi anche usare Lambda SnapStart per mitigare i problemi di avvio a freddo. SnapStartutilizza istantanee memorizzate nella cache dell'ambiente di esecuzione per migliorare significativamente le prestazioni di avvio. Per ulteriori informazioni su SnapStart

funzionalità, limitazioni e aree supportate, consulta. Migliorare le 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
```

AWS Lambda oggetto di contesto in 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 restituirlo. `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/ .java TestContext

```
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/[$LATEST]704f8dxmpla04097b9134246b8438f1a");
  }
  public String getFunctionName(){
    return new String("my-function");
  }
  public String getFunctionVersion(){
    return new String("$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 AWS Lambda registrazione delle funzioni in Java.

Contesto nelle applicazioni di esempio

L' GitHub archivio di questa guida include applicazioni di esempio che dimostrano l'uso dell'oggetto context. Ogni applicazione di esempio include script per facilitare la distribuzione e la pulizia, un modello AWS Serverless Application Model (AWS SAM) e risorse di supporto.

Applicazioni Lambda di esempio in Java

- java17-examples: una funzione Java che dimostra come utilizzare un record Java per rappresentare un oggetto di dati dell'evento di input.

- 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' AWS SDK 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.
- Utilizza API Gateway per richiamare una funzione Lambda: una funzione Java che esegue la scansione di una tabella Amazon DynamoDB che contiene informazioni sui dipendenti. Quindi utilizza Amazon Simple Notification Service per inviare un messaggio di testo ai dipendenti per festeggiare i loro anniversari di lavoro. Questo esempio usa API Gateway per richiamare la funzione.

AWS Lambda registrazione delle funzioni in Java

AWS Lambda monitora automaticamente le funzioni Lambda e invia le voci di registro ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della 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 registri, consulta. Utilizzo dei CloudWatch log di Amazon con AWS Lambda

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

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: 200 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.
- 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 segmento X-Ray.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Utilizzo dei controlli di registrazione avanzati di Lambda con Java

Per avere 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 Java supportati:

- Formato di log: scegli tra i formati di testo normale e JSON strutturato per i log della 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 log a cui la CloudWatch 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 la funzione 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 si seleziona JSON come formato di log della funzione, Lambda invierà gli output log utilizzando la classe `LambdaLogger` come JSON strutturati. 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 di registro di questo codice di esempio verrebbe 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 log, Lambda gli assegnerà automaticamente il livello INFO.

Se il codice utilizza già un'altra libreria di registrazione per generare log JSON strutturati, non è necessario apportare alcuna modifica. 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 CloudWatch nella struttura JSON che definisci.

Utilizzo del filtraggio a livello di log con Java

Per AWS Lambda filtrare i log delle applicazioni in base al loro livello di registro, la funzione deve utilizzare 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. Successivamente, Lambda filtra gli output log utilizzando la coppia chiave-valore "livello" nell'oggetto JSON descritto in the section called “Utilizzo del formato di log JSON strutturato con Java”. Per informazioni su come configurare il formato di log della funzione, consulta la pagina the section called “Configurazione dei controlli di registrazione avanzati per la funzione Lambda”.
- Utilizza un'altra libreria o metodo di registrazione per creare nel codice dei log JSON strutturati che includono una coppia chiave-valore "livello" che definisce il livello dell'output 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 per la registrazione strutturata AWS SAM” e the section called “Registrazione avanzata con Log4j 2 e SLF4J”.

Quando configuri la tua funzione per utilizzare il filtraggio a livello di log, devi selezionare una delle seguenti opzioni per il livello di log 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 funzionam ento della funzione
WARN	Messaggi relativi a potenziali errori che possono portare a comportamenti imprevisti se non risolti

Livello di log	Utilizzo standard
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'appl icazione

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.

Lambda invia i log del livello selezionato e inferiore a. CloudWatch Ad esempio, se configuri un livello di log WARN, Lambda invierà i log corrispondenti ai livelli WARN, ERROR e FATAL.

Registrazione avanzata con Log4j 2 e SLF4J

> **Note**
>
> AWS Lambda non include Log4j2 nei suoi runtime gestiti o nelle immagini dei contenitori di base. Pertanto, non sono influenzati dai problemi descritti in CVE-2021-44228, CVE-2021-45046 e CVE-2021-45105.
> Per i casi in cui una funzione cliente include una versione Log4j2 interessata, abbiamo applicato una modifica ai temèi di esecuzione gestiti Lambda Java e alle immagini del container di base che aiutano a mitigare i problemi in CVE-2021-44228, CVE-2021-45046 e 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 mappatore jndi nell'output Log4J2 saranno sostituite con "`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-log 4j2 nelle proprie funzioni devono eseguire l'aggiornamento alla versione 1.5.0 (o successiva) e ridistribuire le proprie funzioni. Questa versione aggiorna le dipendenze dell'utility Log4j2 sottostanti alla versione 2.17.0 (o successiva). Il binario aws-

lambda-java-log 4j2 aggiornato è disponibile nel repository Maven e il suo 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 dei log, supportare la registrazione durante i test unitari e registrare le chiamate AWS SDK, utilizzate Apache Log4j2 con SLF4J. Log4j è una libreria di registrazione per i programmi Java che consente di configurare i livelli di log e utilizzare librerie appender. SLF4J è una libreria di tipo Facade che consente di modificare quale libreria si utilizza senza modificare il codice di 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'appender

```
<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>
```

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": "0bced576-3936-4e5a-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'appender Lambda. Per configurazioni di test di esempio, consulta le applicazioni di esempio nella sezione successiva.

Altri strumenti e librerie

Powertools for AWS Lambda (Java) è un toolkit per sviluppatori che implementa le migliori pratiche Serverless e aumenta 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 per la registrazione strutturata AWS SAM

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Java Hello World di esempio con i moduli Powertools for AWS Lambda (Java) ~ integrati utilizzando. AWS SAM Questa applicazione implementa un back-end 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
- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS SAM

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:

```
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:34.095000
 INIT_START Runtime Version: java:11.v15    Runtime Version ARN: arn:aws:lambda:eu-
central-1::runtime:0a25e3e7a1cc9ce404bc435eeb2ad358d8fa64338e618d0c224fe509403583ca
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:34.114000
 Picked up JAVA_TOOL_OPTIONS: -XX:+TieredCompilation -XX:TieredStopAtLevel=1
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:34.793000
 Transforming org/apache/logging/log4j/core/lookup/JndiLookup
 (lambdainternal.CustomerClassLoader@1a6c5a9e)
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:35.252000
 START RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765 Version: $LATEST
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 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": "2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81",
  "executionEnvironment": "AWS_Lambda_java11"
}
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.974000 Feb
 03, 2023 9:24:36 AM com.amazonaws.xray.AWSXRayRecorder <init>
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.993000 Feb
 03, 2023 9:24:36 AM com.amazonaws.xray.config.DaemonConfiguration <init>
2023/02/03/[$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.
2023/02/03/[$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_4eMpNsOMPfEVRczD7T1RdhbtiwV1Q==","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_4eMpNsOMPfEVRczD7T1RdhbtiwV1Q=="],"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}
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:37.351000
 09:24:37.351 [main] INFO  helloworld.App - Retrieving https://
checkip.amazonaws.com
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 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": "$LATEST",
  "Service": "service_undefined",
  "logStreamId": "2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81",
  "executionEnvironment": "AWS_Lambda_java11"
}
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:39.371000 END
 RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:39.371000
 REPORT RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765    Duration: 4118.98 ms
 Billed Duration: 4119 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 configurare la conservazione dei log, aggiungi quanto segue al tuo modello: AWS SAM

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

Uso della 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 del codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l'output del log nella sezione Dettagli.

Utilizzo della CloudWatch console

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. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegliere il gruppo di log della funzione (/aws/lambda/***`function-name`***).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Nuovi flussi di log vengono visualizzati quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire più 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 il flusso di log nella traccia.

AWS Command Line InterfaceAWS CLI Usando il ()

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- AWS CLI — Configurazione rapida con `aws configure`

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":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "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-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.

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 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.

```
#!/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 seguente comando 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.

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 facilitare la distribuzione e la pulizia, un AWS SAM modello e risorse di supporto.

Applicazioni Lambda di esempio in Java

- java17-examples: una funzione Java che dimostra come utilizzare un record Java per rappresentare un oggetto di dati dell'evento di input.
- 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' AWS SDK 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.
- Utilizza API Gateway per richiamare una funzione Lambda: una funzione Java che esegue la scansione di una tabella Amazon DynamoDB che contiene informazioni sui dipendenti. Quindi utilizza Amazon Simple Notification Service per inviare un messaggio di testo ai dipendenti per festeggiare i loro anniversari di lavoro. Questo esempio usa API Gateway per richiamare la funzione.

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 utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, che possono includere funzioni Lambda e altri servizi AWS .

Per inviare dati di tracciamento a X-Ray, è possibile utilizzare una delle due librerie SDK:

- AWS Distro for OpenTelemetry (ADOT): una distribuzione sicura, pronta per la produzione e supportata dell'SDK (Otel AWS). OpenTelemetry
- SDK AWS X-Ray per Java: un SDK per generare e inviare i dati di traccia su X-Ray.
- Powertools for AWS Lambda (Java): un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori.

Ciascun SDK offre dei modi per inviare i dati di telemetria al servizio X-Ray. Puoi quindi utilizzare X-Ray per visualizzare, filtrare e analizzare le metriche delle prestazioni dell'applicazione per identificare i problemi e le opportunità di ottimizzazione.

> **Important**
>
> 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 end-to-end tracciamento in X-Ray utilizzando entrambe le soluzioni. Per ulteriori informazioni sulla scelta più adatta, consulta Scegliere tra le AWS Distro per OpenTelemetry e SDK X-Ray.

Sections

- Attivazione del tracciamento con AWS CloudFormation
- Interpretazione di una traccia X-Ray
- Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray)
- Tracciamento X-Ray in applicazioni di esempio (SDK X-Ray)

Utilizzo di Powertools per AWS Lambda (Java) e per il tracciamento AWS SAM

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Java Hello World di esempio con i moduli Powertools for AWS Lambda (Java) integrati utilizzando il. AWS SAM Questa applicazione implementa un back-end 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
- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS SAM

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 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 14:31:48+01:00
  End time: 2023-02-03 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 (2023-02-03T14:31:48.500000) 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
```

Utilizzo di Powertools per AWS Lambda (Java) e AWS CDK per il tracciamento

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Java Hello World di esempio con i moduli Powertools for AWS Lambda (Java) integrati utilizzando. AWS CDK Questa applicazione implementa un back-end 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

- 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, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS CDK

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>
                    <aspectLibrary>
                        <groupId>software.amazon.lambda</groupId>
                        <artifactId>powertools-logging</artifactId>
                    </aspectLibrary>
                </aspectLibraries>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                    </goals>
                </execution>
            </executions>
        </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-cachefile-
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(singletonList(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 directory `hello-world` e implementa l'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: 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-YBg8yfYtOc9j -
 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-YBg8yfYtOc9j
 - 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-YBg8yfYtOc9j - 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-YBg8yfYtOc9j [HTTP: 200]
 - 0.016s - sam-app-HelloWorldFunction-YBg8yfYtOc9j
   - 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 della funzione all'avvio per raccogliere dati di tracciamento. Per istruzioni dettagliate su come utilizzare questo layer insieme all'agente Java ADOT, consulta AWS Distro for Lambda OpenTelemetry 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 layer, consulta AWS Distro for OpenTelemetry Lambda Support for Java nella documentazione ADOT.

Utilizzo dell'SDK X-Ray per strumentare le funzioni Java

Per registrare dati sulle chiamate effettuate dalla funzione ad altre risorse e servizi nell'applicazione, è possibile aggiungere l'SDK X-Ray per Java alla configurazione di compilazione. L'esempio seguente mostra una configurazione di build di 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 aver apportato le modifiche necessarie al codice, attivare 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. Scegli Modifica.
5. In X-Ray, attivare Active tracing (Tracciamento attivo).
6. Selezionare Salva.

Attivazione del tracciamento con l'API Lambda

Configura il tracciamento sulla tua funzione Lambda con AWS o SDK, utilizza AWS CLI le seguenti operazioni API:

- UpdateFunctionConfigurazione
- GetFunctionConfigurazione
- CreateFunction

Il AWS CLI comando di esempio seguente abilita il tracciamento attivo 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 AWS CloudFormation

Per attivare il tracciamento su una `AWS::Lambda::Function` risorsa in un AWS 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), utilizzate la `Tracing` proprietà.

Example template.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretazione di una traccia 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

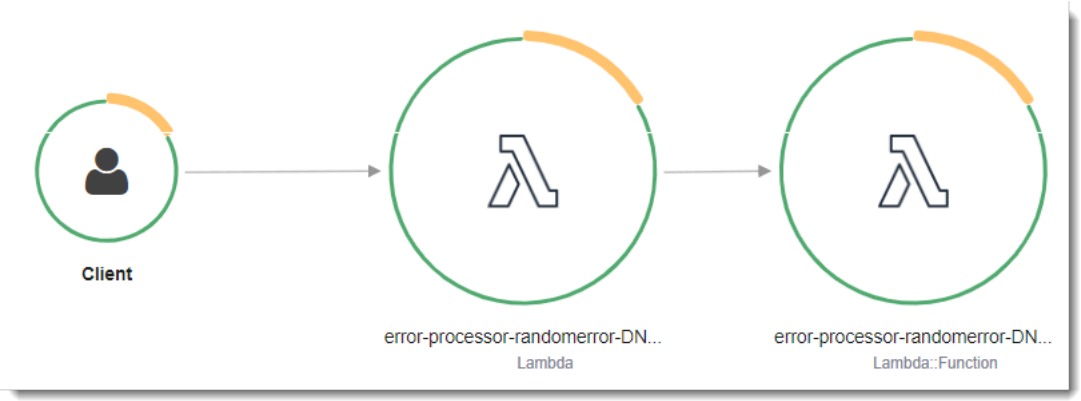
esecuzione della funzione. Altrimenti, aggiungete la AWSXRayDaemonWriteAccesspolitica al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico dei servizi X-Ray mostra informazioni sull'applicazione e tutti i suoi componenti. L'immagine seguente 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 compaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon Logs. CloudWatch

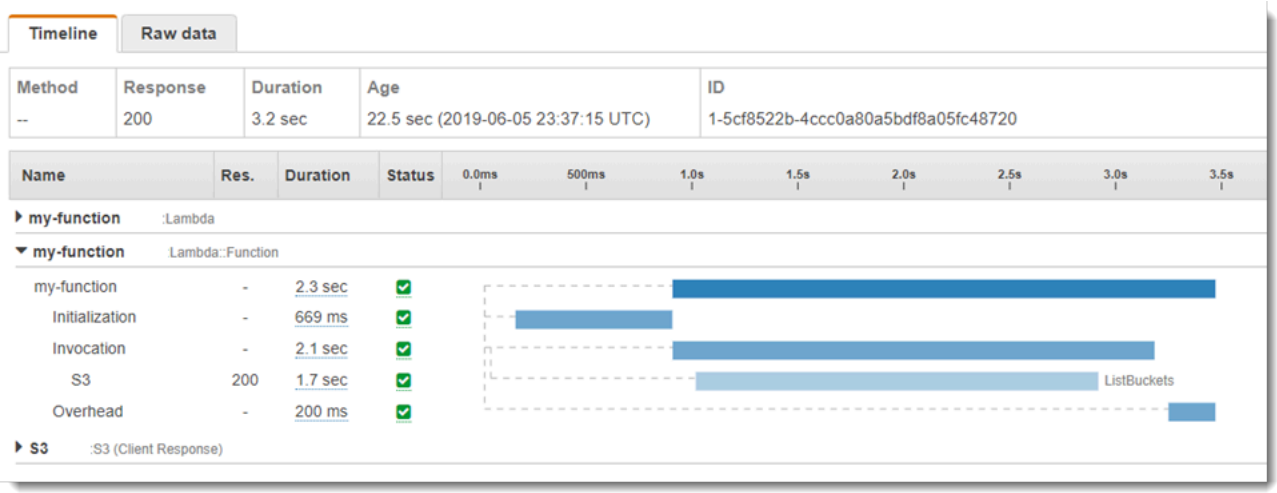
X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste.

> **Note**
>
> La frequenza di campionamento di X-Ray non può essere configurata per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, il che crea 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 denominati my-function, ma uno ha un'origine di `AWS::Lambda` e l'altro ha un'origine di. `AWS::Lambda::Function` Se il `AWS::Lambda` segmento mostra un errore, il servizio Lambda presentava un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione presentava un problema.



Questo esempio espande il `AWS::Lambda::Function` segmento per mostrarne i tre 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.

> **Note**
>
> Le funzioni Lambda SnapStart includono anche un sottosegmento `Restore`. Il sottosegmento `Restore` mostra il tempo impiegato da Lambda per ripristinare uno snapshot, caricare il runtime (JVM) ed eseguire qualsiasi hook di runtime `afterRestore`. 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 tracciamento X-Ray gratuitamente ogni mese fino a un determinato limite come parte del AWS piano gratuito. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consultare Prezzi di AWS X-Ray.

Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray)

Se utilizzi X-Ray SDK per strumentare i client AWS SDK del codice della funzione, il pacchetto di distribuzione può diventare piuttosto grande. Per evitare di caricare dipendenze di runtime ogni volta che si aggiorna il codice della funzione, includere l'SDK X-Ray in un livello Lambda.

L'esempio seguente mostra una risorsa `AWS::Serverless::LayerVersion` che memorizza l'SDK AWS SDK for Java e 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:
        - java21
```

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 aiutare a ridurre i tempi di caricamento.

La creazione di un layer per le dipendenze richiede modifiche alla configurazione di compilazione per generare l'archivio dei layer prima della distribuzione. Per un esempio funzionante, vedete l'applicazione di esempio java-basic su. GitHub

Tracciamento X-Ray in applicazioni di esempio (SDK X-Ray)

L' GitHub archivio di questa guida include applicazioni di esempio che dimostrano l'uso del tracciamento a raggi X. Ogni applicazione di esempio include script per facilitare l'implementazione e la pulizia, un AWS SAM modello e risorse di supporto.

Applicazioni Lambda di esempio in Java

- java17-examples: una funzione Java che dimostra come utilizzare un record Java per rappresentare un oggetto di dati dell'evento di input.
- 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' AWS SDK 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.
- Utilizza API Gateway per richiamare una funzione Lambda: una funzione Java che esegue la scansione di una tabella Amazon DynamoDB che contiene informazioni sui dipendenti. Quindi

utilizza Amazon Simple Notification Service per inviare un messaggio di testo ai dipendenti per festeggiare i loro anniversari di lavoro. Questo esempio usa API Gateway per richiamare la funzione.

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 archiviare le dipendenze di runtime.

Esempi di applicazioni 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 facilitare la distribuzione e la pulizia, un AWS CloudFormation modello e risorse di supporto.

Applicazioni Lambda di esempio in Java

- java17-examples: una funzione Java che dimostra come utilizzare un record Java per rappresentare un oggetto di dati dell'evento di input.
- 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' AWS SDK 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.
- Utilizza API Gateway per richiamare una funzione Lambda: una funzione Java che esegue la scansione di una tabella Amazon DynamoDB che contiene informazioni sui dipendenti. Quindi utilizza Amazon Simple Notification Service per inviare un messaggio di testo ai dipendenti per festeggiare i loro anniversari di lavoro. Questo esempio usa API Gateway per richiamare la funzione.

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 di più nelle guide Micronaut/Lambda.
- Demo dell'applicazione Quarkus senza server: un esempio che mostra come utilizzare Quarkus in un runtime Java gestito con e senza, o come immagine nativa GraalVM con un runtime

personalizzato. SnapStart Scopri di più nella guida Quarkus/Lambda e nella guida 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'SDK di Lambda. AWS 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 runtime solo per il sistema operativo (la famiglia `provided` di runtime) per distribuire le funzioni Go in Lambda.

Argomenti

Supporto per il runtime Go

Il runtime gestito Go 1.x per Lambda è obsoleto. Se disponi di funzioni che utilizzano il runtime Go 1.x, devi migrare le funzioni in or. `provided.al2023` `provided.al2` I `provided.al2` runtime `provided.al2023` and offrono diversi vantaggi`go1.x`, tra cui il supporto per l'architettura arm64 (processori AWS Graviton2), 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 la sezione Migrazione AWS Lambda delle funzioni dal runtime Go1.x al runtime personalizzato su Amazon Linux 2 sul blog di Compute AWS .

Solo per il sistema operativo

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Runtime solo per il sistema operativo	`provided.al2023`	Amazon Linux 2023			
Runtime solo per il sistema operativo	`provided.al2`	Amazon Linux 2			

Strumenti e librerie

Lambda fornisce i seguenti strumenti e librerie per il runtime Go:

- AWS SDK for Go: l'SDK AWS ufficiale per il linguaggio di programmazione Go.
- github.com/aws/aws-lambda-go/lambda: l'implementazione di un modello di programmazione Lambda per Go. Questo pacchetto viene utilizzato da per AWS Lambda richiamare il gestore.
- github.com/aws/aws-lambda-go/lambdacontext: gestori per l'accesso alle informazioni relative al contesto dall'oggetto contesto.
- github.com/aws/aws-lambda-go/events: questa libreria fornisce definizioni relative alla tipologia per le integrazioni delle origini eventi comuni.
- github.com/aws/aws-lambda-go/cmd/build-lambda-zip: questo strumento può essere utilizzato per creare un archivio file .zip su Windows.

Per ulteriori informazioni, consulta aws-lambda-go on. 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, la registrazione, le variabili di ambiente e l'SDK. AWS Questa app utilizza il runtime `go1.x`.

Definisci il gestore 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.

Una funzione Lambda scritta in Go viene creata come eseguibile di Go. Nel codice della funzione Lambda è necessario includere il pacchetto github.com/aws/aws-lambda-go/lambda che implementa il modello di programmazione Lambda per Go. Inoltre, è necessario implementare il codice della funzione del gestore e una funzione `main()`.

Example Funzione Go Lambda

```
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/lambda"
)

type MyEvent struct {
 Name string `json:"name"`
}

func HandleRequest(ctx context.Context, event *MyEvent) (*string, error) {
 if event == nil {
  return nil, fmt.Errorf("received nil event")
 }
 message := fmt.Sprintf("Hello %s!", event.Name)
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Di seguito è riportato un input di esempio per questa funzione:

```
{
  "name": "Jane"
```

```
}
```

Tieni presente quanto segue:

- package main: in Go, il pacchetto contenente `func main()` deve essere sempre denominato `main`.
- import: consente di includere le librerie richieste dalla funzione Lambda. In questa istanza, include:
  - context: Oggetto contesto AWS Lambda in Go.
  - fmt: l'oggetto di formattazione di Go utilizzato per formattare il valore restituito della funzione.
  - github.com/aws/aws-lambda-go/lambda: come già accennato, implementa un modello di programmazione Lambda per Go.
- func HandleRequest (ctx context.context, event *MyEvent) (*string, error): Questa è la firma del tuo gestore Lambda. È il punto di ingresso per la funzione Lambda e contiene la logica che viene eseguita quando la funzione viene richiamata. Inoltre, i parametri inclusi indicano quanto segue:
  - ctx context.Context: fornisce le informazioni di runtime relative all'invocazione della funzione Lambda. `ctx` è la variabile dichiarata per sfruttare le informazioni disponibili mediante Oggetto contesto AWS Lambda in Go.
  - evento* MyEvent: Questo è un parametro denominato che punta a. `event` `MyEvent` Rappresenta l'input per la funzione Lambda.
  - *string, error: il gestore restituisce due valori. Il primo è un puntatore a una stringa con il risultato della funzione Lambda. Il secondo è un tipo di errore, vale a dire `nil` se non si verifica alcun errore e contiene informazioni standard sull'errore se qualcosa va storto.
  - return &message, nil: restituisce due valori. Il primo è un puntatore a un messaggio in formato stringa, che è un saluto costruito utilizzando il campo `Name` dell'evento di input. Il secondo valore, `nil`, indica che la funzione non ha riscontrato errori.
- func main(): il punto di ingresso che esegue il codice della funzione Lambda. Questo dato è obbligatorio.

  La funzione Lambda verrà eseguita se si aggiunge `lambda.Start(HandleRequest)` tra le parentesi del codice `func main(){}`. Per gli standard del linguaggio Go, la parentesi iniziale, `{`, deve essere posizionata direttamente alla fine della firma della funzione `main`.

Denominazione

runtime provided.al2 e provided.al2023

Per le funzioni Go che utilizzano il runtime `provided.al2` o `provided.al2023` in un pacchetto di implementazione .zip, il file eseguibile che contiene il codice della funzione deve essere denominato `bootstrap`. Se desideri implementare la funzione con un file .zip, il file `bootstrap` deve trovarsi nella root di quello .zip. Per le funzioni Go che utilizzano il runtime `provided.al2` o `provided.al2023` in un'immagine di container, puoi utilizzare qualsiasi nome per il file eseguibile.

Puoi usare qualsiasi nome per il gestore. Per fare riferimento al valore del gestore nel codice, puoi usare la variabile di ambiente `_HANDLER`.

Runtime go1.x

Per le funzioni Go che utilizzano il runtime `go1.x`, il file eseguibile e il gestore possono condividere qualsiasi nome. Ad esempio, se imposti il valore del gestore su `Handler`, Lambda chiamerà la funzione `main()` nel file eseguibile `Handler`.

Per modificare il nome del gestore funzioni nella console Lambda, nella scheda Impostazioni runtime, scegliere Modifica.

Gestore della funzione Lambda che utilizza tipi di dato strutturati

Nell'esempio riportato sopra, il tipo di input era una semplice stringa. Tuttavia, è anche possibile passare eventi strutturati al gestore della funzione:

```
package main

import (
 "fmt"
 "github.com/aws/aws-lambda-go/lambda"
)

type MyEvent struct {
 Name string `json:"What is your name?"`
 Age  int    `json:"How old are you?"`
}

type MyResponse struct {
```

```
 Message string `json:"Answer"`
}

func HandleLambdaEvent(event *MyEvent) (*MyResponse, error) {
 if event == nil {
  return nil, fmt.Errorf("received nil event")
 }
 return &MyResponse{Message: fmt.Sprintf("%s is %d years old!", event.Name,
 event.Age)}, nil
}

func main() {
 lambda.Start(HandleLambdaEvent)
}
```

Di seguito è riportato un input di esempio per questa funzione:

```
{
    "What is your name?": "Jim",
    "How old are you?": 33
}
```

La risposta avrebbe questa struttura:

```
{
    "Answer": "Jim is 33 years old!"
}
```

Affinché siano esportati, i nomi dei campi nella struttura dell'evento devono avere l'iniziale maiuscola. Per ulteriori informazioni sulla gestione degli eventi dalle AWS fonti di eventi, consulta aws-lambda-go/events.

Firme del gestore valide

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 richiedere 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 del gestore valide. `TIn` e `TOut` rappresentano le tipologie compatibili con la libreria standard encoding/json. Per ulteriori informazioni su come deserializzare queste tipologie, consultare func Unmarshal.

- `func ()`
- `func () error`
- `func (TIn) error`
- `func () (TOut, error)`
- `func (context.Context) error`
- `func (context.Context, TIn) error`
- `func (context.Context) (TOut, error)`
- `func (context.Context, TIn) (TOut, error)`

Utilizzo dello stato globale

È possibile dichiarare e modificare le variabili globali indipendenti dal codice del gestore della funzione Lambda. Inoltre, il gestore può dichiarare una funzione `init` che viene eseguita quando il gestore viene caricato. Si comporta allo stesso modo dei programmi Go standard. AWS Lambda Una singola istanza della funzione Lambda non gestirà mai più eventi simultaneamente.

Example Funzione Go con variabili globali

> **Note**
>
> Questo codice utilizza la AWS SDK for Go V2. Per ulteriori informazioni, vedere Guida introduttiva alla AWS SDK for Go V2.

```
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "log"
)

var invokeCount int
var myObjects []types.Object

func init() {
 // Load the SDK configuration
 cfg, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  log.Fatalf("Unable to load SDK config: %v", err)
 }

 // Initialize an S3 client
 svc := s3.NewFromConfig(cfg)

 // Define the bucket name as a variable so we can take its address
 bucketName := "DOC-EXAMPLE-BUCKET"
 input := &s3.ListObjectsV2Input{
  Bucket: &bucketName,
 }

 // List objects in the bucket
 result, err := svc.ListObjectsV2(context.TODO(), input)
 if err != nil {
  log.Fatalf("Failed to list objects: %v", err)
 }
 myObjects = result.Contents
}

func LambdaHandler(ctx context.Context) (int, error) {
 invokeCount++
 for i, obj := range myObjects {
  log.Printf("object[%d] size: %d key: %s", i, obj.Size, *obj.Key)
 }
```

```
 return invokeCount, nil
}

func main() {
 lambda.Start(LambdaHandler)
}
```

Oggetto contesto AWS Lambda in Go

Quando Lambda esegue la funzione, passa un oggetto Context al gestore. Questo oggetto fornisce i metodi e le proprietà con informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

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 l' CognitoIdentityPoolID.

L'esempio che segue illustra come utilizzare l'oggetto contesto per monitorare il tempo impiegato per il completamento della funzione Lambda. In tal modo è possibile analizzare le aspettative prestazionali e, se necessario modificare la funzione di conseguenza.

```
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)
}
```

Distribuisci funzioni Lambda per Go con gli archivi di file .zip

Il codice della 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, quindi utilizzare il file.zip per distribuire il codice della funzione su AWS Management Console, AWS Command Line Interface (AWS CLI) e (). AWS Lambda AWS Serverless Application Model AWS SAM

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.

Sections

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. 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 for Go, scarica il set standard di moduli SDK, insieme a tutti i client API AWS di servizio richiesti dall'applicazione. Per informazioni su come installare l'SDK for Go, consulta Guida introduttiva AWS SDK for Go alla versione 2.

Utilizzo della famiglia di runtime fornita

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato in modo nativo in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un runtime solo per il sistema operativo (la famiglia `provided` di runtime) per distribuire le funzioni Go in Lambda.

Creazione di un pacchetto di implementazione .zip (macOS/Linux)

1. Nella directory del progetto contenente il file `main.go` dell'applicazione, compila il tuo eseguibile. Tieni presente quanto segue:

   - L'eseguibile deve essere denominato `bootstrap`. Per ulteriori informazioni, consulta Denominazione.
   - Imposta l'architettura del set di istruzioni di destinazione. I runtime solo per il sistema operativo 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 necessario solo se 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 . Tieni presente quanto segue:

   - Il file binario deve essere denominato `bootstrap` ma il nome del gestore può essere qualsiasi cosa. Per ulteriori informazioni, consulta Denominazione.
   - 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 build-lambda-ziplo strumento per Windows da GitHub, compilare il file eseguibile e creare un pacchetto di distribuzione con estensione 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. 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 for Go, scarica il set standard di moduli SDK, insieme a tutti i client API AWS di servizio richiesti dall'applicazione. Per informazioni su come installare l'SDK for Go, consulta Guida introduttiva AWS SDK for Go alla versione 2.

Utilizzo della famiglia di runtime fornita

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato in modo nativo in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un runtime solo per il sistema operativo (la famiglia `provided` di runtime) per distribuire le funzioni Go in Lambda.

Creazione di un pacchetto di implementazione .zip (Windows)

1. Scarica lo strumento da. build-lambda-zip 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 operazioni riportate, a seconda della tua architettura del set di istruzioni di destinazione. I runtime solo per il sistema operativo 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 necessario solo se 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 GOOS=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 GOOS=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. I runtime solo per il sistema operativo 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 necessario solo se 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 . Tieni presente quanto segue:

- Il file binario deve essere denominato `bootstrap` ma il nome del gestore può essere qualsiasi cosa. Per ulteriori informazioni, consulta Denominazione.

- 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 Lambda AWS Command Line Interface e l'API Lambda. Puoi anche creare e aggiornare le funzioni Lambda usando AWS Serverless Application Model (AWS SAM) e AWS 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 Linux, Lambda richiede 644 permessi per i file non eseguibili (rw-r--r--) e 755 permessi () per le directory e i file eseguibili. rwxr-xr-x

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 eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 755 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta Set, View, Change, or Remove Permissions on an Object nella documentazione di Microsoft Windows.

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 AWS Management Console, consulta la Guida introduttiva ad Amazon S3. Per caricare file utilizzando la AWS CLI, consulta Move objects 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. Selezionare 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. Usa la funzione create-function e update-function-codei comandi per distribuire 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 su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

> **Note**
>
> Se carichi il tuo file.zip da un bucket Amazon S3 utilizzando AWS CLI il, il bucket deve trovarsi nella stessa posizione della Regione AWS tua funzione.

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=DOC-EXAMPLE-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 DOC-EXAMPLE-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

Il 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 utilizzi 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 eventuali dipendenze specificate. Per ulteriori informazioni sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda, consulta la Guida introduttiva AWS Serverless Application Model alla AWS SAM Developer Guide.

È inoltre possibile utilizzare 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 tua macchina di compilazione. Per istruzioni su come caricare un file su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.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 del codice della funzione, sul percorso della cartella locale o sull'oggetto FunctionCode
- `Runtime`: imposta il runtime prescelto

Inoltre AWS SAM, se il tuo file.zip è più grande di 50 MB, non è necessario caricarlo prima in un bucket Amazon S3. AWS SAM puoi caricare pacchetti.zip fino alla dimensione massima consentita di 250 MB (decompressi) da una posizione sulla macchina di compilazione locale.

Per ulteriori informazioni sulla distribuzione delle funzioni utilizzando il file.zip in AWS SAM, consulta la Guida per gli sviluppatori. AWS::Serverless::FunctionAWS 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 AWS CloudFormation

È possibile utilizzare AWS 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 su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

Nel AWS 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 che AWS CloudFormation genera non può superare i 4 MB. Per ulteriori informazioni sulla distribuzione delle funzioni utilizzando il file.zip in AWS CloudFormation, AWS::Lambda::Functionconsultate la Guida per l'utente.AWS CloudFormation

Creazione di un livello Go per dipendenze

> **Note**
>
> L'utilizzo di livelli con funzioni in un linguaggio compilato come Go potrebbe non offrire gli stessi vantaggi di un linguaggio interpretato come Python. Siccome Go è un linguaggio compilato, 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. È preferibile, invece, includere qualunque codice condiviso in fase di compilazione per sfruttare le ottimizzazioni integrate del compilatore.

Le istruzioni in questa sezione spiegano come includere dipendenze in un livello.

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.

Distribuzione delle funzioni Go Lambda con immagini di container

Esistono due modi per creare un'immagine contenitore per una funzione Go Lambda:

- Utilizzo di un'immagine di base solo per il sistema operativo AWS

  Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato in modo nativo in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un'immagine di base solo per il sistema operativo per creare immagini Go per Lambda. Per rendere l'immagine compatibile con Lambda, devi includere il pacchetto `aws-lambda-go/lambda` nell'immagine.
- Utilizzo di un'immagine non di base AWS

  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 in modo nativo in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un'immagine di base solo per il sistema operativo per distribuire le funzioni Go su Lambda.

Solo per il sistema operativo

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Runtime solo per il sistema operativo	`provided.al2023`	Amazon Linux 2023			
Runtime solo per il sistema operativo	`provided.al2`	Amazon Linux 2			

Galleria pubblica di Amazon Elastic Container Registry: gallery.ecr.aws/lambda/provided

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 Utilizzo di un'immagine di base solo per il sistema operativo AWS o Utilizzo di un'immagine non di base AWS.

Utilizzo di un'immagine di base solo per il sistema operativo AWS

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato in modo nativo in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un'immagine di base solo per il sistema operativo per creare immagini di contenitori per le funzioni Go.

Tag	Runtime	Sistema operativo	Dockerfile	Definizione come obsoleto
al2023	Runtime solo per il sistema operativo	Amazon Linux 2023	Dockerfile per Runtime solo per sistema operativo su GitHub	

Tag	Runtime	Sistema operativo	Dockerfile	Definizione come obsoleto
al2	Runtime solo per il sistema operativo	Amazon Linux 2	Dockerfile per Runtime solo per sistema operativo attivo GitHub	

Per ulteriori informazioni su queste immagini di base, consulta la documentazione fornita nella galleria pubblica di Amazon ECR.

Devi includere il pacchetto aws-lambda-go/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:

- Go
- Docker
- AWS Command Line Interface () versione 2 AWS CLI

Creazione di un'immagine dall'immagine di base provided.al2023

Creazione e implementazione di 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)
 (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. Il seguente Dockerfile di esempio utilizza una build multi-fase. Ciò consente di utilizzare un'immagine di base diversa in ogni passaggio. Puoi utilizzare un'immagine, ad esempio un'immagine di base Go, per compilare il codice e creare il file 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. Il processo di compilazione è separato dall'immagine di implementazione 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 se si utilizza il runtime Go 1.x obsoleto. L'esclusione dell'RPC riduce le dimensioni del pacchetto di implementazione.

Example - Dockerfile di compilazione multi-fase

> **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`.

```
docker build --platform linux/amd64 -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. Tieni 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.
   - 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:
   - Sostituisci `docker-image:test` con il nome e il tag della tua immagine Docker.
   - 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 image in un digest di immagini specifico. Ciò significa che se punti il tag immagine utilizzato per distribuire 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.

Utilizzo di un'immagine non di base AWS

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 pacchetto aws-lambda-go/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:

- Go
- Docker
- AWS Command Line Interface (AWS CLI) versione 2

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)
 (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.

   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`.

```
docker build --platform linux/amd64 -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. È possibile creare l'emulatore nell'immagine o utilizzare la seguente procedura per installarlo sul 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. Tieni 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.
   - 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:
   - Sostituisci `docker-image:test` con il nome e il tag della tua immagine Docker.
   - 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 image in un digest di immagini specifico. Ciò significa che se punti il tag immagine utilizzato per distribuire 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.

AWS Lambda funzione di registrazione in Go

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della 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 Utilizzo dei CloudWatch log di Amazon con AWS Lambda.

Questa pagina descrive come produrre un output di registro dal codice della funzione Lambda o accedere ai log utilizzando AWS Command Line Interface la console Lambda o la console. CloudWatch

Sections

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: $LATEST
2020/03/27 03:40:05 EVENT: {
  "Records": [
    {
      "messageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
      "receiptHandle": "MessageReceiptHandle",
      "body": "Hello from SQS!",
      "md5OfBody": "7b27xmplb47ff90a553787216d55d91d",
      "md5OfMessageAttributes": "",
      "attributes": {
        "ApproximateFirstReceiveTimestamp": "1523232000001",
        "ApproximateReceiveCount": "1",
        "SenderId": "123456789012",
        "SentTimestamp": "1523232000000"
      },
      ...
2020/03/27 03:40:05 AWS_LAMBDA_LOG_STREAM_NAME=2020/03/27/
[$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: 39 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 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.

- 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 segmento X-Ray.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Uso della 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 del codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l'output del log nella sezione Dettagli.

Utilizzo della 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. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegliere il gruppo di log della funzione (/aws/lambda/***function-name***).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Nuovi flussi di log vengono visualizzati quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire più 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 il flusso di log nella traccia.

AWS Command Line InterfaceAWS CLI Usando il ()

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2

- AWS CLI — Configurazione rapida con `aws configure`

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":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "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-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.

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 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.

```
#!/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 seguente comando 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.

Strumentazione del codice Go in AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, che possono includere funzioni Lambda e altri servizi AWS .

Per inviare dati di tracciamento a X-Ray, è possibile utilizzare una delle due librerie SDK:

- AWS Distro for OpenTelemetry (ADOT): una distribuzione sicura, pronta per la produzione e supportata dell'SDK (Otel AWS). OpenTelemetry
- AWS X-Ray SDK for Go: un SDK per la generazione e l'invio di dati di traccia a X-Ray.

Ciascun SDK offre dei modi per inviare i dati di telemetria al servizio X-Ray. Puoi quindi utilizzare X-Ray per visualizzare, filtrare e analizzare le metriche delle prestazioni dell'applicazione per identificare i problemi e le opportunità di ottimizzazione.

> **Important**
>
> 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 end-to-end tracciamento in X-Ray utilizzando entrambe le soluzioni. Per ulteriori informazioni sulla scelta più adatta, consulta Scegliere tra le AWS Distro per OpenTelemetry e SDK X-Ray.

Sections

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, consulta AWS Distro for OpenTelemetry Lambda Support for Go nella documentazione ADOT.

Utilizzo dell'SDK X-Ray per strumentare le funzioni Go

Per registrare i dettagli sulle chiamate che la funzione Lambda effettua ad altre risorse dell'applicazione, puoi anche utilizzare l' AWS X-Ray SDK for Go. Per scaricare l'SDK, scarica l'SDK dal suo repository con: GitHub `go get`

```
go get github.com/aws/aws-xray-sdk-go
```

Per utilizzare i client AWS SDK di Instrument, passate 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 aver apportato le modifiche necessarie al codice, attivare 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. Scegli Modifica.
5. In X-Ray, attivare Active tracing (Tracciamento attivo).
6. Selezionare Salva.

Attivazione del tracciamento con l'API Lambda

Configura il tracciamento sulla tua funzione Lambda con AWS o SDK, utilizza AWS CLI le seguenti operazioni API:

- UpdateFunctionConfigurazione
- GetFunctionConfigurazione
- CreateFunction

Il AWS CLI comando di esempio seguente abilita il tracciamento attivo 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 AWS CloudFormation

Per attivare il tracciamento su una `AWS::Lambda::Function` risorsa in un AWS 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), utilizzate la `Tracing` proprietà.

Example template.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretazione di una traccia X-Ray

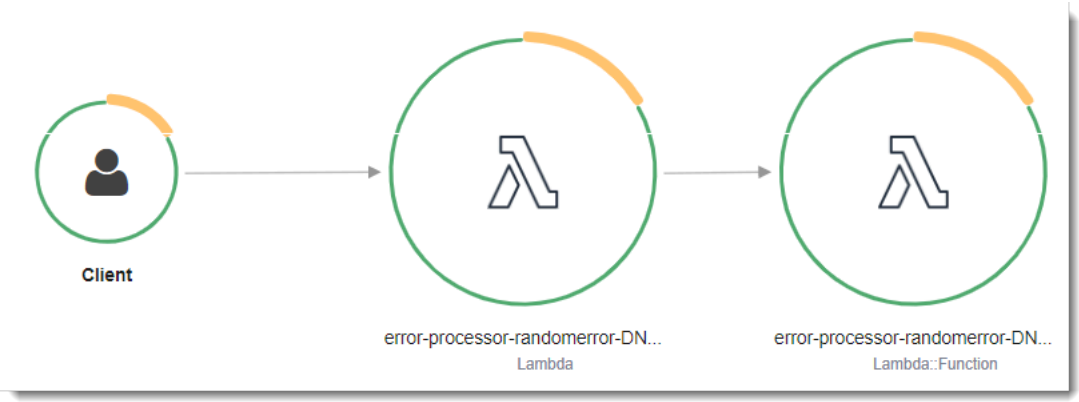
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al ruolo di esecuzione della funzione. Altrimenti, aggiungete la AWSXRayDaemonWriteAccesspolitica al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico dei servizi X-Ray mostra informazioni sull'applicazione e tutti i suoi componenti. L'immagine seguente 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 compaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon Logs. CloudWatch

X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste.

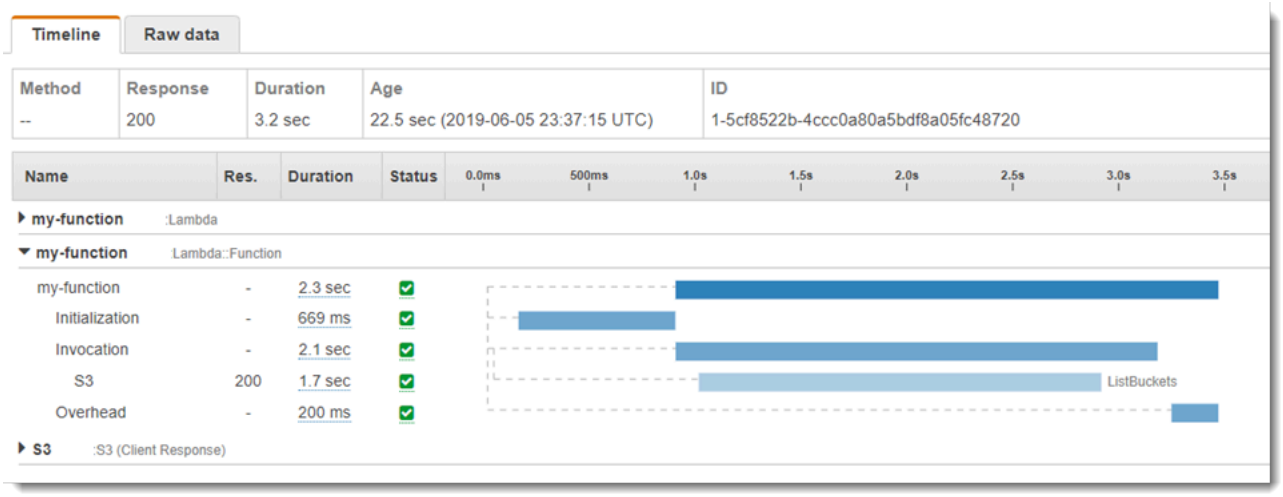
> **Note**
>
> La frequenza di campionamento di X-Ray non può essere configurata per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, il che crea 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 denominati my-function, ma uno ha un'origine di `AWS::Lambda` e l'altro ha un'origine di. `AWS::Lambda::Function` Se il `AWS::Lambda` segmento mostra un errore, il servizio Lambda presentava un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione presentava un problema.



Questo esempio espande il `AWS::Lambda::Function` segmento per mostrarne i tre 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.

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 tracciamento X-Ray gratuitamente ogni mese fino a un determinato limite come parte del AWS piano gratuito. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consultare Prezzi di AWS X-Ray.

Utilizzo delle variabili di ambiente

Per accedere alle variabili di ambiente in Go, utilizzare la funzione Getenv.

Di seguito viene spiegato come procedere. La funzione importa il pacchetto fmt per formattare i risultati stampati e il pacchetto os, un'interfaccia di sistema indipendente dalla piattaforma che consente di accedere alle variabili di ambiente.

```
package main

import (
 "fmt"
 "os"
 "github.com/aws/aws-lambda-go/lambda"
)

func main() {
 fmt.Printf("%s is %s. years old\n", os.Getenv("NAME"), os.Getenv("AGE"))

}
```

Per un elenco di variabili di ambiente impostate dal runtime Lambda, consulta Variabili di ambiente di runtime definite.

Compilazione di funzioni Lambda con C#

È possibile eseguire l'applicazione.NET in Lambda utilizzando i runtime gestiti.NET 6 o .NET 8, un runtime personalizzato o un'immagine del 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:

.NET

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
.NET 8	`dotnet8`	Amazon Linux 2023			
.NET 6	`dotnet6`	Amazon Linux 2	12 novembre 2024	28 febbraio 2025	31 marzo 2025

Configurazione dell'ambiente di sviluppo .NET

Per sviluppare e creare le funzioni Lambda, puoi utilizzare uno qualsiasi degli ambienti di sviluppo integrati.NET (IDE) 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 (.NET 8):

```
dotnet new install Amazon.Lambda.Templates
```

Per installare i modelli di progetto (.NET 6):

```
dotnet new --install Amazon.Lambda.Templates
```

> **Note**
>
> Se utilizzi il runtime Lambda gestito.NET 6, ti consigliamo di eseguire l'aggiornamento a utilizzare.NET 8. Per ulteriori informazioni, consulta Gestire gli aggiornamenti AWS Lambda di runtime e Introduzione al runtime di .NET 8 nel blog AWS Lambda di AWS Compute.

Installazione e aggiornamento degli strumenti CLI

Esegui i seguenti comandi per installare, aggiornare e disinstallare la `Amazon.Lambda.Tools` CLI.

Per installare gli strumenti a riga di comando:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Per aggiornare gli strumenti della riga di comando:

```
dotnet tool update -g Amazon.Lambda.Tools
```

Per disinstallare gli strumenti da riga di comando:

```
dotnet tool uninstall -g Amazon.Lambda.Tools
```

Definisci il gestore di funzioni 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.

Quando la funzione viene richiamata e Lambda esegue il metodo del gestore della funzione, passa due argomenti alla funzione. Il primo argomento è l'oggetto `event`. Quando un altro Servizio AWS richiama la funzione, l'`event`oggetto contiene dati sull'evento che ha causato l'invocazione della funzione. Ad esempio, un oggetto `event` di Gateway API contiene informazioni sul percorso, sul metodo HTTP e sulle intestazioni HTTP. L'esatta struttura degli eventi varia a seconda dell' Servizio AWS invocazione della funzione. Per ulteriori informazioni sui formati degli eventi per i singoli servizi, consulta Integrazione con altri servizi.

Lambda passa alla funzione anche un oggetto `context`. Questo oggetto di contesto contiene informazioni sull'invocazione, sulla funzione e sull'ambiente di esecuzione. Per ulteriori informazioni, consulta the section called “Context”.

Il formato nativo per tutti gli eventi Lambda è costituito da flussi di byte che rappresentano l'evento in formato JSON. Se i parametri di input e output della funzione non sono di tipo `System.IO.Stream`, è necessario serializzarli. Specifica il serializzatore che desideri utilizzare impostando l'attributo assembly `LambdaSerializer`. Per ulteriori informazioni, consulta the section called “Serializzazione nelle funzioni Lambda”.

Argomenti

- Modelli di esecuzione .NET per Lambda
- Gestori di librerie di classi
- Gestori di assembly eseguibili
- Serializzazione nelle funzioni Lambda
- Semplificazione del codice delle funzioni con il framework Lambda Annotations
- Restrizioni del gestore della funzione Lambda

Modelli di esecuzione .NET per Lambda

Esistono due diversi modelli di esecuzione per l'esecuzione delle funzioni Lambda in .NET: l'approccio della libreria di classi e l'approccio dell'assembly eseguibile.

Nell'approccio alla libreria di classi, fornisci a Lambda una stringa che indica `AssemblyName`, `ClassName` e `Method` della funzione da richiamare. Per ulteriori informazioni sul formato di questa stringa, consulta the section called “Gestori di librerie di classi”. Durante la fase di inizializzazione della funzione, la classe della funzione viene inizializzata e viene eseguito qualsiasi codice nel costruttore.

Nell'approccio di assembly eseguibile, si utilizza la funzionalità delle istruzioni di livello superiore di 9 C#. Questo approccio genera un assembly eseguibile che Lambda esegue ogni volta che riceve un comando invoke per la tua funzione. Fornisci a Lambda solo il nome dell'assembly eseguibile da eseguire.

Le sezioni seguenti forniscono esempi di codice di funzione per questi due approcci.

Gestori di librerie di classi

Il seguente codice di funzione Lambda mostra un esempio di metodo handler (`FunctionHandler`) per una funzione Lambda che utilizza l'approccio della libreria di classi. In questa funzione di esempio, Lambda riceve un evento da Gateway API che richiama la funzione. La funzione legge un record da un database e restituisce il record come parte della risposta di Gateway API.

```
[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)
    {
        var id = request.PathParameters["id"];

        var databaseRecord = await this._repo.GetById(id);

        return new APIGatewayProxyResponse
```

```
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = JsonSerializer.Serialize(databaseRecord)
        };
    }
}
```

Quando crei una funzione Lambda, devi fornire a Lambda informazioni sul gestore della funzione sotto forma di stringa del gestore. Questo indica a Lambda quale metodo del codice eseguire quando la funzione viene richiamata. In C#, il formato della stringa del gestore quando si utilizza l'approccio della libreria di classi è il seguente:

`ASSEMBLY::TYPE::METHOD`, dove:

- `ASSEMBLY` è il nome del file di assembly .NET per l'applicazione. Se si crea un'applicazione utilizzando `Amazon.Lambda.Tools` CLI e non si specifica il nome dell'assembly con la proprietà `AssemblyName` in .csproj, allora `ASSEMBLY` è semplicemente il nome del file .csproj.
- `TYPE` è il nome completo del tipo di gestore, costituito da `Namespace` e `ClassName`.
- `METHOD` è il nome del metodo del gestore della funzione nel codice.

Per il codice di esempio mostrato, se l'assembly è denominato `GetProductHandler`, la stringa del gestore sarebbe `GetProductHandler::GetProductHandler.Function::FunctionHandler`.

Gestori di assembly eseguibili

Nell'esempio seguente, la funzione Lambda è definita come un assembly eseguibile. Il metodo handler in questo codice è denominato `Handler`. Quando si utilizzano assembly eseguibili, è necessario avviare il runtime Lambda. Per far ciò, viene utilizzato il metodo `LambdaBootstrapBuilder.Create`. Questo metodo accetta come input il metodo utilizzato dalla funzione come gestore e il serializzatore Lambda da utilizzare.

Per ulteriori informazioni sull'utilizzo delle istruzioni di primo livello, consulta Introducing the .NET 6 runtime for AWS Lambda sul blog di AWS calcolo.

```
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 = input.PathParameters["id"];

    var databaseRecord = await this.repo.GetById(id);

    return new APIGatewayProxyResponse
    {
        StatusCode = (int)HttpStatusCode.OK,
        Body = JsonSerializer.Serialize(databaseRecord)
    };
};
```

Quando si utilizzano assembly eseguibili, la stringa del gestore che indica a Lambda come eseguire il codice è il nome dell'assembly. In questo esempio, sarebbe `GetProductHandler`.

Serializzazione nelle funzioni Lambda

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 possibile implementare la serializzazione utilizzando la serializzazione standard basata sulla riflessione fornita da `System.Text.Json` e `Newtonsoft.Json` oppure utilizzando la serializzazione generata dal codice sorgente.

Utilizzo della serializzazione generata dal codice sorgente

La serializzazione generata dal codice sorgente è una funzionalità delle versioni 6 e successive di .NET che consente di generare il codice di serializzazione in fase di compilazione. Elimina la necessità di riflessione e può migliorare le prestazioni della tua funzione. Per utilizzare la serializzazione generata dal codice sorgente nella tua funzione, procedi come segue:

- 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 o deserializzazione manuale nel codice dell'applicazione per utilizzare la classe appena creata.

Nel codice seguente viene illustrata una funzione di esempio che utilizza la serializzazione generata dal codice sorgente.

```
[assembly:
 LambdaSerializer(typeof(SourceGeneratorLambdaJsonSerializer<CustomSerializer>))]

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function()
    {
        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,
 CustomSerializer.Default.Product)
        };
    }
}

[JsonSerializable(typeof(APIGatewayProxyRequest))]
[JsonSerializable(typeof(APIGatewayProxyResponse))]
[JsonSerializable(typeof(Product))]
public partial class CustomSerializer : JsonSerializerContext
{

}
```

> **Note**
>
> Se desideri utilizzare la compilazione anticipata (AOT) nativa con Lambda, devi utilizzare la serializzazione generata dal codice sorgente.

Utilizzo della serializzazione basata sulla riflessione

AWS fornisce librerie predefinite che consentono di aggiungere rapidamente la serializzazione all'applicazione. È possibile configurarlo utilizzando i pacchetti `Amazon.Lambda.Serialization.SystemTextJson` o `Amazon.Lambda.Serialization.Json` NuGet . Dietro le quinte, `Amazon.Lambda.Serialization.SystemTextJson` utilizza `System.Text.Json` per eseguire attività di serializzazione e `Amazon.Lambda.Serialization.Json` utilizza il pacchetto `Newtonsoft.Json`.

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`.

Semplificazione del codice delle funzioni con il framework Lambda Annotations

Lambda Annotations è un framework per .NET 6 e.NET 8 che semplifica la scrittura di funzioni Lambda utilizzando C#. 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.

Il codice di esempio riportato di seguito mostra come l'utilizzo del framework delle annotazioni può semplificare la scrittura di funzioni Lambda. Il primo esempio mostra il codice scritto utilizzando il normale modello di programmazione Lambda e il secondo mostra l'equivalente utilizzando il framework Annotations.

```
public APIGatewayHttpApiV2ProxyResponse LambdaMathAdd(APIGatewayHttpApiV2ProxyRequest
 request, ILambdaContext context)
{
    if (!request.PathParameters.TryGetValue("x", out var xs))
    {
        return new APIGatewayHttpApiV2ProxyResponse
        {
            StatusCode = (int)HttpStatusCode.BadRequest
        };
    }
    if (!request.PathParameters.TryGetValue("y", out var ys))
    {
        return new APIGatewayHttpApiV2ProxyResponse
        {
            StatusCode = (int)HttpStatusCode.BadRequest
        };
    }
    var x = int.Parse(xs);
    var y = int.Parse(ys);
    return new APIGatewayHttpApiV2ProxyResponse
    {
        StatusCode = (int)HttpStatusCode.OK,
        Body = (x + y).ToString(),
        Headers = new Dictionary#string, string> { { "Content-Type", "text/plain" } }
    };
}
```

```
[LambdaFunction]
[HttpApi(LambdaHttpMethod.Get, "/add/{x}/{y}")]
public int Add(int x, int y)
{
    return x + y;
}
```

Per un altro esempio di come l'utilizzo di Lambda Annotations può semplificare il codice, consulta questo esempio di applicazione cross-service nel repository. `awsdocs/aws-doc-sdk-examples` GitHub La cartella `PamApiAnnotations` utilizza Lambda Annotations nel file `function.cs`

principale. Per fare un confronto, la cartella `PamApi` contiene file equivalenti scritti utilizzando il normale modello di programmazione Lambda.

Il framework Annotations utilizza generatori di codice sorgente per generare codice che si traduce dal modello di programmazione Lambda al codice visto nel secondo esempio.

Per ulteriori informazioni su come utilizzare Lambda Annotations per .NET, consulta le seguenti risorse:

- `aws/aws-lambda-dotnet` GitHub Il repository.
- Presentazione di .NET Annotations Lambda Framework (anteprima) sul blog Developer AWS Tools.
- Il `Amazon.Lambda.Annotations` NuGet pacchetto.

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 attributo `[LambdaStartup]` a un file `Startup.cs`, il framework Lambda Annotations genererà 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);
    }
}
```

Restrizioni del gestore della funzione Lambda

Tieni in considerazione che la firma del gestore presenta alcune restrizioni.

- Nella firma del gestore non è possibile utilizzare `unsafe` e tipi di puntatori. Tuttavia, il contesto `unsafe` può essere utilizzato nel metodo del gestore e nelle relative dipendenze. Per ulteriori informazioni, consulta unsafe (Riferimenti per C#) sul sito web Microsoft Docs.
- Non può passare un numero variabile di parametri utilizzando la parola chiave `params` o utilizzare `ArgIterator` come parametro di input o di restituzione per supportare un numero variabile di parametri.
- Il gestore non può essere un metodo generico, ad esempio IList<T> Sort<T>(IList<T> input).
- I gestori asincroni con firma `async void` non sono supportati.

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, è consigliabile utilizzare framework come AWS Serverless Application Model (AWS SAM) o the su cui distribuire AWS Cloud Development Kit (AWS CDK) le applicazioni.NET. AWS

Le applicazioni serverless di solito comprendono una combinazione di funzioni Lambda e altre Servizi AWS funzioni gestite che interagiscono per eseguire una particolare attività aziendale. AWS SAM e AWS CDK semplifica la creazione e l'implementazione di funzioni Lambda con Servizi AWS altre su larga scala. La specifica del AWS SAM modello fornisce una sintassi semplice e pulita per descrivere le funzioni Lambda, le API, le autorizzazioni, le configurazioni e AWS altre risorse 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. Entrambi AWS SAM utilizzano AWS CDK la CLI globale .NET Lambda per creare pacchetti di 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 sfruttare le ottimizzazioni integrate del compilatore .NET.

Nelle sezioni seguenti sono disponibili le istruzioni per la creazione e la distribuzione di funzioni.NET Lambda utilizzando AWS SAM AWS CDK la, e.NET Lambda Global CLI.

Argomenti

- Utilizzo della CLI globale Lambda di .NET
- Distribuisci le funzioni Lambda C# usando AWS SAM
- Distribuisci le funzioni Lambda C# usando AWS CDK
- Implementa applicazioni ASP.NET

Utilizzo della CLI globale Lambda di .NET

La CLI .NET e l'estensione degli strumenti globali Lambda .NET (`Amazon.Lambda.Tools`) offrono un modo multi-piattaforma per creare applicazioni Lambda basate su .NET, impacchettarle e implementarle in 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

.NET 8 SDK

Se non l'hai già fatto, installa .NET 8 SDK 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 Strumenti CLI globali Amazon.Lambda.Tools.NET

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.Toolsestensione.NET CLI, consulta l'archivio 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 `Amazon.Lambda.Templates` NuGet pacchetto.

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 for .NET credenziali. Per ulteriori informazioni sui profili di credenziali in.NET, consulta 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: 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. 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.DefaultLambdaJsonSerializ

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 indicato in precedenza, si tratta di 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 la distribuzione, la procedura guidata chiede di selezionare un. the section called “Ruolo di esecuzione (autorizzazioni per le funzioni di 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**
>
> L'uso di livelli con funzioni in un linguaggio compilato come C# potrebbe non offrire gli stessi vantaggi di un linguaggio interpretato come Python. Siccome C# è un linguaggio compilato, 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. È preferibile, invece, includere qualunque codice condiviso in fase di compilazione per sfruttare le ottimizzazioni integrate del compilatore.

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, vedi l' blank-csharp-with-layeresempio.

Distribuisci le funzioni Lambda C# usando AWS SAM

Il 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 utilizzi 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 eventuali dipendenze specificate. AWS SAM quindi distribuisce la funzione utilizzando uno stack.AWS CloudFormation Per ulteriori informazioni sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda, consulta la Guida introduttiva AWS Serverless Application Model alla AWS SAM Developer Guide.

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 crei e distribuisci l'applicazione utilizzando AWS SAM, dietro le quinte, la AWS SAM CLI utilizza il comando per impacchettare i singoli `dotnet lambda package` bundle di codici delle funzioni Lambda.

Prerequisiti

.NET 8 SDK

Installa .NET 8 SDK e Runtime.

AWS SAM CLI versione 1.39 o successiva

Per informazioni su come installare la versione più recente della AWS SAM CLI, consulta Installazione della AWS SAM CLI.

Implementa un'applicazione di esempio AWS SAM

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 in 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 un'autorizzazione definita, va bene? , assicurati di entrarey.

- Nome dello stack: il nome dello stack da implementare su AWS CloudFormation. Questo nome deve essere unico per te Account AWS e Regione AWS.
- Regione AWS: La Regione AWS destinazione in cui vuoi distribuire l'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 alle applicazioni.
- Consenti la creazione di ruoli IAM SAM CLI: molti AWS SAM modelli, incluso quello Hello world in questo esempio, creano ruoli AWS Identity and Access Management (IAM) per consentire alle funzioni Lambda di accedere ad altre funzioni Lambda. Servizi AWS Seleziona

Sì per fornire l'autorizzazione a distribuire uno AWS CloudFormation stack che crea o modifica i ruoli IAM.

- Disabilita il rollback: per impostazione predefinita, se si AWS SAM verifica un errore durante la creazione o la distribuzione dello stack, lo stack torna alla versione precedente. Seleziona No per accettare questa impostazione predefinita.
- HelloWorldFunction potrebbe non avere un'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
```

Passaggi successivi

Per ulteriori informazioni sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda usando.NET, consulta le seguenti risorse:

- La Guida per gli sviluppatori di AWS Serverless Application Model (AWS SAM)
- Creazione di applicazioni.NET serverless con AWS Lambda e la CLI SAM

Distribuisci le funzioni Lambda C# usando AWS CDK

AWS Cloud Development Kit (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 AWS 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

.NET 8 SDK

Installa .NET 8 SDK e Runtime.

AWS CDK versione 2

Per informazioni su come installare la versione più recente di, AWS CDK consulta la Guida per sviluppatori introduttiva alla AWS Cloud Development Kit (AWS CDK) versione 2. AWS CDK

Implementa un'applicazione di esempio AWS CDK

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 implementerai utilizzando il 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
```

Passaggi successivi

Per ulteriori informazioni sull'utilizzo AWS CDK per creare e distribuire funzioni Lambda usando.NET, consulta le seguenti risorse:

- Utilizzo del AWS CDK in C#
- Crea, impacchetta e pubblica funzioni.NET C# Lambda con CDK AWS

Implementa applicazioni ASP.NET

Oltre a ospitare funzioni basate sugli eventi, puoi utilizzare .NET con Lambda anche per ospitare applicazioni ASP.NET leggere. È possibile creare e distribuire applicazioni ASP.NET utilizzando il pacchetto. `Amazon.Lambda.AspNetCoreServer` NuGet In questa sezione, imparerai come distribuire un'API Web ASP.NET in Lambda utilizzando gli strumenti della CLI .NET Lambda.

Argomenti

- Prerequisiti
- Implementazione di un'API Web ASP.NET in Lambda
- Implementazione di API minime ASP.NET su Lambda

Prerequisiti

.NET 8 SDK

Installa .NET 8 SDK e ASP.NET Core 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.Toolsestensione.NET CLI, consulta l'archivio AWS Extensions for .NET CLI su. GitHub

Amazon.Lambda.Templates

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
```

Implementazione di un'API Web ASP.NET in Lambda

Per implementare un'API Web utilizzando ASP.NET, puoi utilizzare i modelli Lambda .NET per creare un nuovo progetto di API Web. Utilizza il comando seguente per inizializzare un nuovo progetto di API Web ASP.NET. 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 farà parte della stringa del gestore che Lambda usa per richiamare la tua funzione. Il metodo `FunctionHandlerAsync` trasforma il payload dell'evento in entrata nel formato ASP.NET corretto e la risposta ASP.NET in un payload di 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 AWS CloudFormation.

```
cd AspNetOnLambda/src/AspNetOnLambda
dotnet lambda deploy-serverless
```

> **Tip**
>
> Quando distribuisci un'API utilizzando il **`dotnet lambda deploy-serverless`** comando, AWS CloudFormation assegna alla funzione Lambda un nome basato sul nome dello stack specificato durante la distribuzione. Per assegnare un nome personalizzato alla funzione Lambda, modifica il `serverless.template` file per aggiungere una `FunctionName` proprietà alla `AWS::Serverless::Function` risorsa. Per ulteriori informazioni, consulta Name type nella Guida per AWS CloudFormation l'utente.

Implementazione di API minime ASP.NET su Lambda

Per implementare un'API minima ASP.NET, puoi utilizzare i modelli Lambda .NET per creare un nuovo progetto di API minima. 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 l'API minima da eseguire su Lambda, potrebbe essere necessario modificare questo codice in modo che le richieste e le risposte tra Lambda e ASP.NET Core vengano 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 AWS CloudFormation.

```
cd MinimalApiOnLambda/src/MinimalApiOnLambda
dotnet lambda deploy-serverless
```

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 di AWS 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.
- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  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

- 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	Definizione come obsoleto
8	.NET 8	Amazon Linux 2023	Dockerfile per.NET 8 su GitHub	
6	.NET 6	Amazon Linux 2	Dockerfile per .NET 6 attivo GitHub	12 novembre 2024

Repository Amazon ECR: gallery.ecr.aws/lambda/dotnet

Utilizzo di un'immagine di AWS base per.NET

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- .NET SDK: 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

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 NuGet Amazon.Lambda.Templates.

```
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
```

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: è il file in cui si specificano le opzioni della riga di comando quando si implementa la funzione Lambda.
   - 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 la pagina Serializzazione nelle funzioni Lambda. 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.
   - Readme.md: questo file contiene ulteriori informazioni sulla funzione Lambda di esempio.

4. Esamina il Dockerfile nella directory `src/`*MyFunction*. 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. La tua versione .NET deve corrispondere alla versione dell'immagine di base.
   - Imposta l'argomento `CMD` specificando il gestore della funzione Lambda. Questo dovrebbe corrispondere a `image-command` in `aws-lambda-tools-defaults.json`.

   Example Dockerfile

```
# You can also pull these images from DockerHub amazon/aws-lambda-dotnet:8
FROM public.ecr.aws/lambda/dotnet:8

# Copy function code to Lambda-defined environment variable
COPY publish/* ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler (could also be done as a parameter override outside
 of the Dockerfile)
CMD [ "MyFunction::MyFunction.Function::FunctionHandler" ]
```

5. 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
```

6. Passa alla directory *MyFunction*/src/*MyFunction*, se non lo hai ancora fatto.

```
cd src/MyFunction
```

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 sullo strumento globale Amazon.Lambda.Tools, consulta l'archivio Extensions AWS for .NET CLI su. 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: id  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
```

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 di runtime Lambda, 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 runtime Lambda per .NET. L'esempio Dockerfile utilizza l'immagine di base di 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 8. Assicurati che la tua versione di .NET corrisponda alla versione dell'immagine di base specificata nel tuo Dockerfile.
- 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. RuntimeSupportpacchetto.

   ```
   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: è il file in cui si specificano le opzioni della riga di comando quando si implementa 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 Usare 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. La tua versione .NET deve corrispondere alla versione dell'immagine di base.
   - 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.

   Example Dockerfile

```
# You can also pull these images from DockerHub amazon/aws-lambda-dotnet:8
FROM mcr.microsoft.com/dotnet/runtime:8.0

# Set the image's internal work directory
WORKDIR /var/task

# Copy function code to Lambda-defined environment variable
COPY "bin/Release/net8.0/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: id  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.NET Lambda in un formato di runtime nativo

.NET 8 supporta la compilazione nativa ahead-of-time (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, vedi Introduzione al runtime di .NET 8 nel blog AWS Lambda di AWS Compute.

Sections

Runtime Lambda

Per distribuire una build di funzione Lambda con compilazione AOT nativa, usa il runtime gestito.NET 8 Lambda. Questo runtime supporta l'uso di entrambe le architetture x86_64 e arm64.

Quando si distribuisce una funzione.NET Lambda senza utilizzare AOT, l'applicazione viene prima compilata nel codice Intermediate Language (IL). In fase di esecuzione, il compilatore just-in-time (JIT) nel runtime Lambda prende il codice IL e lo compila in codice macchina secondo necessità. Con una funzione Lambda compilata in anticipo con AOT nativo, compili il codice in codice macchina quando distribuisci 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) utilizzato dal runtime .NET 8. La CLI.NET Lambda offre funzionalità per compilare l'applicazione in un contenitore Docker utilizzando un'immagine AL2023.

Per evitare potenziali problemi di compatibilità tra architetture diverse, consigliamo vivamente di compilare il codice in un ambiente con la stessa architettura di processore configurata per la

funzione. Per ulteriori informazioni sui limiti della compilazione tra architetture diverse, vedere Compilazione incrociata nella 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 DI.NET 8. I comandi CLI.NET nelle seguenti sezioni utilizzano Docker per sviluppare e creare funzioni Lambda in un ambiente AL2023.

.NET 8 SDK

La compilazione AOT nativa è una funzionalità di.NET 8. È necessario installare il.NET 8 SDK sulla macchina di compilazione, non solo sul 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.Toolsestensione.NET CLI, consulta l'archivio AWS Extensions for .NET CLI su. GitHub

Amazon.Lambda.Templates

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
```

Nozioni di base

Sia il.NET Global CLI 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 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 tempo di esecuzione 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 ti vengano addebitati addebiti inutili. 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 taglio migliorato rispetto alle versioni precedenti di.NET. È stato aggiunto il supporto alle librerie di runtime Lambda, AWS .NET SDK, .NET Lambda Annotations e .NET 8 stesso.

Questi miglioramenti offrono la possibilità di eliminare gli avvisi di taglio in fase di compilazione, ma .NET non sarà mai completamente sicuro. Ciò significa che parti delle librerie su cui si basa la funzione possono essere eliminate durante la fase di compilazione. Puoi gestirlo definendo `TrimmerRootAssemblies` come parte del tuo `.csproj` file, 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 taglio, l'aggiunta della classe che genera l'avviso `TrimmerRootAssembly` potrebbe non risolvere il problema. Un avviso di trim indica che la classe sta tentando di accedere a un'altra classe che non può essere determinata fino al runtime. Per evitare errori di runtime, aggiungete questa seconda classe a`TrimmerRootAssembly`.

Per ulteriori informazioni sulla gestione degli avvisi di taglio, vedere Introduzione agli avvisi di taglio nella documentazione di Microsoft.NET.

Risoluzione dei problemi

Errore: la compilazione nativa tra sistemi operativi 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 comuni, consulta il repository AWS NativeAOT for .NET su. GitHub

Oggetto contesto AWS Lambda in 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)
            };
        }
        finally
        {
            cts.Cancel();

            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.InternalServerError,
                Body = JsonSerializer.Serialize(databaseRecord)
            };
        }
    }
}
```

Registrazione della funzione Lambda in C#

AWS Lambda monitora automaticamente le funzioni Lambda e invia le voci di registro ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della 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 registri, consulta. Utilizzo dei CloudWatch log di Amazon con AWS Lambda

Sections

Creazione di una funzione che restituisce i registri

Per i registri di output del codice della funzione, puoi usare i metodi della classe della console o qualsiasi libreria di registrazione che scriva in `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 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.
- 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 segmento X-Ray.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Strumenti e librerie

Powertools for AWS Lambda (.NET) è un toolkit per sviluppatori che implementa le migliori pratiche Serverless e aumenta 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 per la registrazione strutturata AWS SAM

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 il. AWS SAM Questa applicazione implementa un back-end 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:

- .NET 6 o .NET 8
- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Distribuisci un'applicazione di esempio AWS SAM

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 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:

```
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8
 2023-02-20T14:15:27.988000 INIT_START Runtime Version:
 dotnet:6.v13        Runtime Version ARN: arn:aws:lambda:ap-
southeast-2::runtime:699f346a05dae24c58c45790bc4089f252bf17dae3997e79b17d939a288aa1ec
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:28.229000
 START RequestId: bed25b38-d012-42e7-ba28-f272535fb80e Version: $LATEST
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:29.259000
 2023-02-20T14:15:29.201Z        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}
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:30.479000
 2023-02-20T14:15:30.479Z        bed25b38-d012-42e7-ba28-f272535fb80e    info
 {"ColdStart":true,"XrayTraceId":"1-63f3807f-5dbcb9910c96f50742707542","CorrelationId":"d3d
a549-4d67b2fdc015","FunctionName":"sam-app-HelloWorldFunction-
haKIoVeose2p","FunctionVersion":"$LATEST","FunctionMemorySize":256,"FunctionArn":"arn:aws:l
southeast-2:123456789012:function:sam-app-HelloWorldFunction-
haKIoVeose2p","FunctionRequestId":"bed25b38-d012-42e7-ba28-
f272535fb80e","Timestamp":"2023-02-20T14:15:30.4602970Z","Level":"Information","Service":"P
 world API - HTTP 200"}
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:30.599000
 2023-02-20T14:15:30.599Z        bed25b38-d012-42e7-ba28-f272535fb80e    info
   {"_aws":{"Timestamp":1676902528922,"CloudWatchMetrics":[{"Namespace":"sam-
app-logging","Metrics":[{"Name":"ApiRequestCount","Unit":"Count"}],"Dimensions":
[["Service"]]}]},"Service":"PowertoolsHelloWorld","ApiRequestCount":1}
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:30.680000 END
 RequestId: bed25b38-d012-42e7-ba28-f272535fb80e
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:30.680000
 REPORT RequestId: bed25b38-d012-42e7-ba28-f272535fb80e  Duration: 2450.99 ms
```

```
  Billed Duration: 2451 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 configurare la conservazione dei log, aggiungi quanto segue al tuo modello: AWS SAM

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

Uso della 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 del codice incorporato, troverai i log nei risultati dell’esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l’output del log nella sezione Dettagli.

Utilizzo della CloudWatch console

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. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegliere il gruppo di log della funzione (/aws/lambda/***function-name***).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Nuovi flussi di log vengono visualizzati quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire più 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 il flusso di log nella traccia.

AWS Command Line InterfaceAWS CLI Usando il ()

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- AWS CLI — Configurazione rapida con `aws configure`

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":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "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-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.

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 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.

```
#!/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 seguente comando 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.

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 utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, che possono includere funzioni Lambda e altri servizi AWS .

Per inviare dati di tracciamento a X-Ray, è possibile utilizzare una delle tre librerie SDK:

- AWS Distro for OpenTelemetry (ADOT): una distribuzione sicura, pronta per la produzione e supportata dell'SDK (Otel AWS). OpenTelemetry
- SDK AWS X-Ray per .NET: un SDK per generare e inviare i dati di traccia su X-Ray.
- Powertools for AWS Lambda (.NET): un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori.

Ciascun SDK offre dei modi per inviare i dati di telemetria al servizio X-Ray. Puoi quindi utilizzare X-Ray per visualizzare, filtrare e analizzare le metriche delle prestazioni dell'applicazione per identificare i problemi e le opportunità di ottimizzazione.

> **Important**
>
> 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 end-to-end tracciamento in X-Ray utilizzando entrambe le soluzioni. Per ulteriori informazioni sulla scelta più adatta, consulta Scegliere tra le AWS Distro per OpenTelemetry e SDK X-Ray.

Sections

Utilizzo di Powertools per AWS Lambda (.NET) e per il tracciamento AWS SAM

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Hello World C# di esempio con i moduli Powertools for AWS Lambda (.NET) integrati utilizzando. AWS SAM Questa applicazione implementa un back-end 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:

- .NET 6 o .NET 8
- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS SAM

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 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 <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-pNjujb7mEoew - 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-pNjujb7mEoew
 - 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 nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste.

> **Note**
>
> La frequenza di campionamento di X-Ray non può essere configurata per le funzioni.

Utilizzo dell'SDK X-Ray per strumentare le 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 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. Ciò consente di sfruttare il riutilizzo dell'ambiente di esecuzione per migliorare le prestazioni della funzione. Nel seguente esempio di codice, il `RegisterXRayForAllServices` metodo viene chiamato nel costruttore di funzioni per aggiungere la 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. Scegli Modifica.
5. In X-Ray, attivare Active tracing (Tracciamento attivo).
6. Selezionare Salva.

Attivazione del tracciamento con l'API Lambda

Configura il tracciamento sulla tua funzione Lambda con AWS o SDK, utilizza AWS CLI le seguenti operazioni API:

- UpdateFunctionConfigurazione
- GetFunctionConfigurazione
- CreateFunction

Il AWS CLI comando di esempio seguente abilita il tracciamento attivo 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 AWS CloudFormation

Per attivare il tracciamento su una `AWS::Lambda::Function` risorsa in un AWS 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), utilizzate la `Tracing` proprietà.

Example template.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretazione di una traccia X-Ray

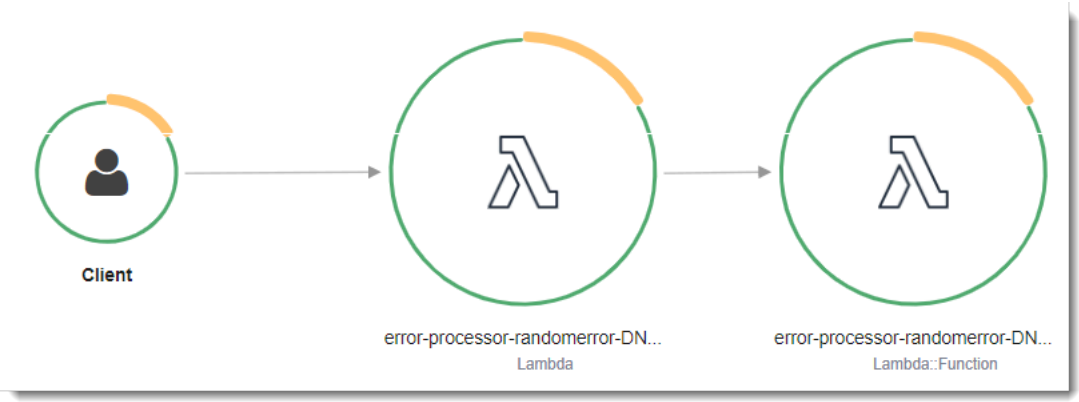
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al ruolo di esecuzione della funzione. Altrimenti, aggiungete la AWSXRayDaemonWriteAccesspolitica al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico dei servizi X-Ray mostra informazioni sull'applicazione e tutti i suoi componenti. L'immagine seguente 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 compaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon Logs. CloudWatch

X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste.

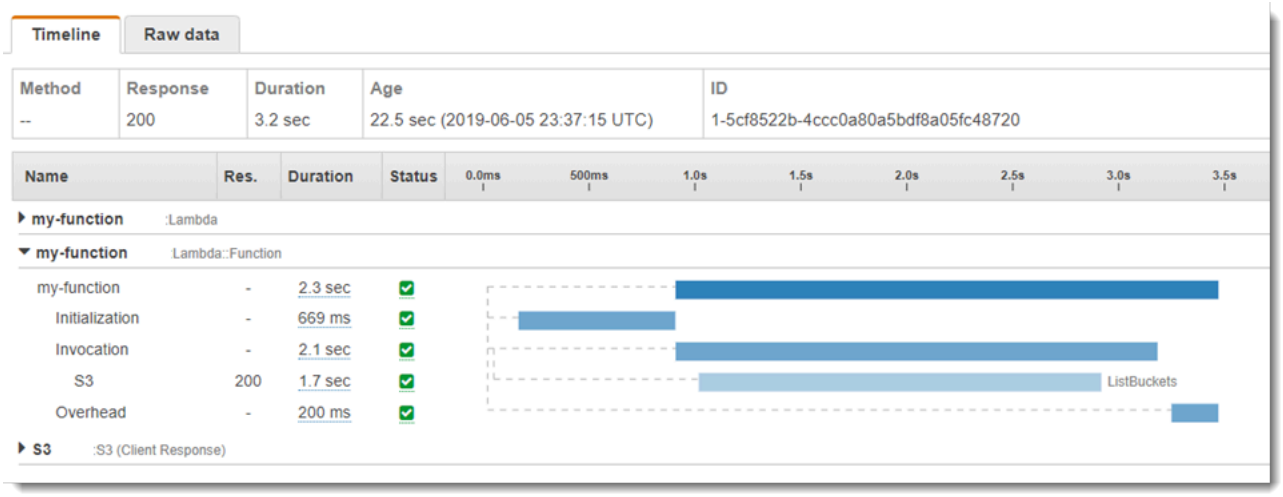
> **Note**
>
> La frequenza di campionamento di X-Ray non può essere configurata per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, il che crea 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 denominati my-function, ma uno ha un'origine di `AWS::Lambda` e l'altro ha un'origine di. `AWS::Lambda::Function` Se il `AWS::Lambda` segmento mostra un errore, il servizio Lambda presentava un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione presentava un problema.



Questo esempio espande il `AWS::Lambda::Function` segmento per mostrarne i tre 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.

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 tracciamento X-Ray gratuitamente ogni mese fino a un determinato limite come parte del AWS piano gratuito. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consultare Prezzi di AWS X-Ray.

Test della funzione AWS Lambda in C#

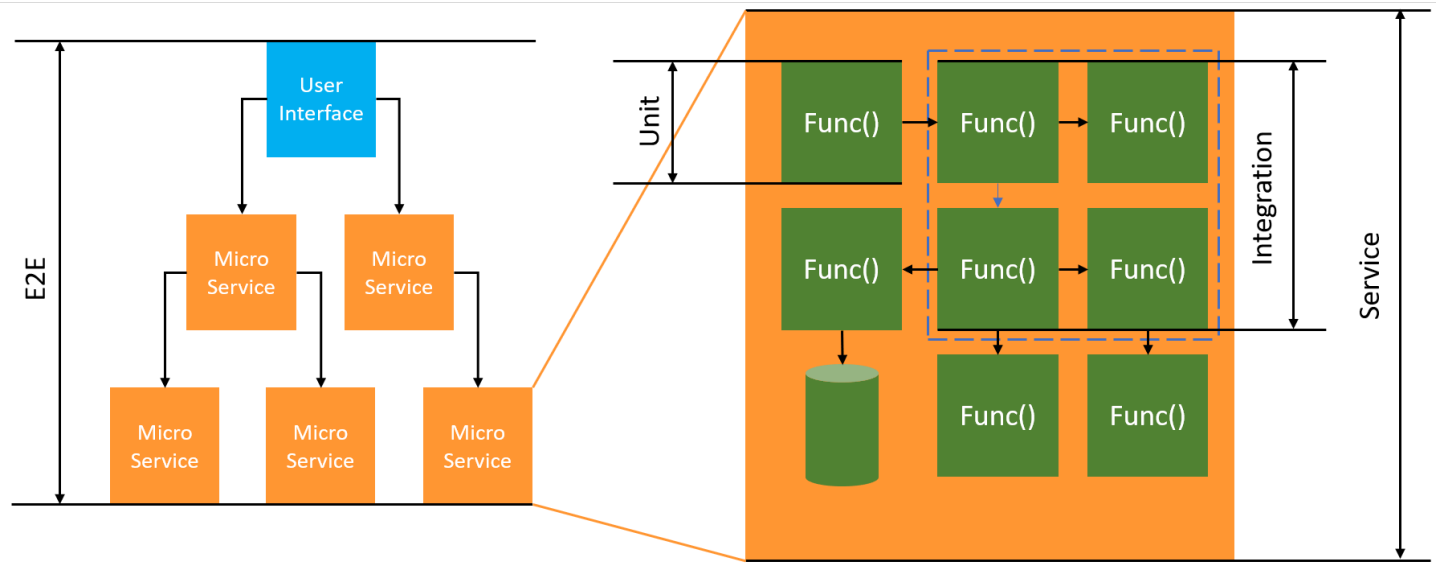
> **Note**
>
> Consulta il capitolo sul Test delle funzioni per un'introduzione completa alle tecniche e alle best practice per testare soluzioni serverless.

Sebbene il test delle funzioni serverless si basi su tipologie e tecniche di test consolidate, è importante tenere in considerazione la possibilità di effettuare test sulle applicazioni serverless nella loro interezza. I test basati sul cloud forniranno la misura più accurata della qualità sia delle funzioni sia 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 senza server, continuerai a scrivere unità, integrazioni e end-to-endtest.

- 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.
- E nd-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

Generalmente, utilizzerai una combinazione di approcci per testare il codice dell'applicazione serverless, inclusi test nel cloud, test con mock e occasionalmente test con emulatori.

Test nel cloud

I test nel cloud sono utili per tutte le fasi del test, inclusi test unitari, test di integrazione e end-to-end test. 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, consulta l'archivio dei campioni di testing .NET su 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 codice di funzione Lambda. 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

Prima di iniziare, devi prima configurare un ambiente di sviluppo. PowerShell Per istruzioni su come eseguire questa operazione, consultare Configurazione di un ambiente di PowerShell sviluppo.

Per ulteriori informazioni su come utilizzare il AWSLambdaPSCore modulo per scaricare PowerShell progetti di esempio dai modelli, creare pacchetti di PowerShell distribuzione e distribuire PowerShell funzioni nel AWS cloud, consultaImplementa le funzioni PowerShell Lambda con archivi di file.zip.

Lambda fornisce i runtime seguenti per i linguaggi .NET:

.NET

Nome	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
.NET 8	`dotnet8`	Amazon Linux 2023			
.NET 6	`dotnet6`	Amazon Linux 2	12 novembre 2024	28 febbraio 2025	31 marzo 2025

Argomenti

- Configurazione di un ambiente di PowerShell sviluppo
- Implementa le funzioni PowerShell Lambda con archivi di file.zip
- Definisci il gestore di funzioni Lambda in PowerShell
- AWS Lambdaoggetto di contesto in PowerShell
- AWS Lambda funzione di accesso PowerShell

Configurazione di un ambiente di PowerShell sviluppo

Lambda fornisce un set di strumenti e librerie per il PowerShell runtime. Per le istruzioni di installazione, consulta Lambda tools for PowerShell on. GitHub

Il AWSLambdaPSCore modulo include i seguenti cmdlet per aiutare a creare e pubblicare funzioni Lambda PowerShell :

- Get AWSPowerShellLambdaTemplate -: restituisce un elenco di modelli introduttivi.
- Nuovo- AWSPowerShellLambda — Crea uno PowerShell script iniziale basato su un modello.
- Publish- AWSPowerShellLambda — Pubblica un determinato PowerShell script in Lambda.
- Nuovo- AWSPowerShellLambdaPackage — Crea un pacchetto di distribuzione Lambda che è possibile utilizzare in un sistema CI/CD per la distribuzione.

Implementa le funzioni PowerShell Lambda con archivi di file.zip

Un pacchetto di distribuzione per il PowerShell runtime contiene PowerShell lo script, PowerShell i moduli necessari per PowerShell lo script e gli assembly necessari per ospitare Core. PowerShell

Creazione di una funzione Lambda

Per iniziare a scrivere e richiamare uno PowerShell script con Lambda, è possibile utilizzare `New-AWSPowerShellLambda` il cmdlet per creare uno script di avvio basato su un modello. È possibile utilizzare il cmdlet `Publish-AWSPowerShellLambda` 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 PowerShell script, caricarlo e testarlo, procedi come segue:

1. Per visualizzare l'elenco dei modelli disponibili, esegui il comando seguente:

   ```
   PS C:\> Get-AWSPowerShellLambdaTemplate

   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-AWSPowerShellLambda -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 del tuo PowerShell script vengono inviati ad Amazon CloudWatch Logs, decommenta la `Write-Host` riga dello script di esempio.

   Per dimostrare come puoi restituire i dati dalle tue funzioni Lambda, aggiungi una nuova riga alla fine dello script con `$PSVersionTable`. Questo aggiunge `$PSVersionTable` alla pipeline. PowerShell Una volta completato lo PowerShell script, l'ultimo oggetto nella PowerShell pipeline è costituito dai dati di ritorno per la funzione Lambda. `$PSVersionTable`è una variabile PowerShell globale che fornisce anche 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
   ```

Definisci il gestore di funzioni Lambda in PowerShell

Quando viene richiamata una funzione Lambda, il gestore Lambda richiama lo script. PowerShell

Quando lo PowerShell script viene richiamato, le seguenti variabili sono predefinite:

- *`$ LambdaInput`* — Un oggetto PSObject che contiene l'input del 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 LambdaContext oggetto Amazon.Lambda.Core.I che puoi utilizzare per accedere alle informazioni sulla chiamata corrente, come il nome della funzione corrente, il limite di memoria, il tempo di esecuzione rimanente e la registrazione.

Ad esempio, considera il seguente codice di esempio. PowerShell

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host 'Function Name:' $LambdaContext.FunctionName
```

Questo script restituisce la FunctionName proprietà ottenuta dalla LambdaContext variabile $.

> **Note**
>
> È necessario utilizzare l'`#Requires`istruzione all'interno PowerShell degli script per indicare i moduli da cui dipendono gli script. Questa istruzione svolge due attività importanti. 1) Comunica agli altri sviluppatori i moduli utilizzati dallo script e 2) identifica i moduli dipendenti che AWS PowerShell gli strumenti devono includere nello script, come parte della distribuzione. Per ulteriori informazioni sull'`#Requires`istruzione in PowerShell, vedere About requires. Per ulteriori informazioni sui pacchetti PowerShell di distribuzione, vedereImplementa le funzioni PowerShell Lambda con archivi di file.zip.
> Quando la funzione PowerShell Lambda utilizza i AWS PowerShell cmdlet, assicurati di impostare un'`#Requires`istruzione che faccia riferimento al `AWSPowerShell.NetCore` modulo, che supporta PowerShell Core, e non al modulo, che supporta solo Windows`AWSPowerShell`. 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, consultare AWS Strumenti per PowerShell.

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 che viene aggiunto alla PowerShell pipeline sono 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 PowerShell dichiarazione, che si aggiunge alla pipeline: `$PSVersionTable` PowerShell

```
$PSVersionTable
```

Al termine dello PowerShell script, l'ultimo oggetto nella PowerShell pipeline è costituito dai dati di ritorno per la funzione Lambda. `$PSVersionTable`è una variabile PowerShell globale che fornisce anche informazioni sull'ambiente di esecuzione.

AWS Lambdaoggetto di contesto in 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.

Il seguente frammento di PowerShell codice mostra una semplice funzione di gestione che stampa alcune informazioni di 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
```

AWS Lambda funzione di accesso PowerShell

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della 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 Utilizzo dei CloudWatch log di Amazon con AWS Lambda.

Questa pagina descrive come produrre un output di registro dal codice della funzione Lambda o accedere ai log utilizzando AWS Command Line Interface la console Lambda o la console. CloudWatch

Sections

Creazione di una funzione che restituisce i registri

Per generare log dal codice della funzione, è possibile utilizzare i cmdlet in 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: 4000 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 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.
- 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 segmento X-Ray.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Uso della 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 del codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l'output del log nella sezione Dettagli.

Utilizzo della 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. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegliere il gruppo di log della funzione (/aws/lambda/***function-name***).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Nuovi flussi di log vengono visualizzati quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire più 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 il flusso di log nella traccia.

AWS Command Line InterfaceAWS CLI Usando il ()

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- AWS CLI — Configurazione rapida con `aws configure`

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":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "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-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.

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 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.

```
#!/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 seguente comando 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.

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 implementalo su Lambda utilizzando il runtime `provided.al2023` o `provided.al2`. Durante l'utilizzo di `provided.al2023` o `provided.al2`, Lambda mantiene automaticamente aggiornato il sistema operativo con le patch più recenti.

> **Note**
>
> Il client di runtime Rust è un pacchetto sperimentale. È soggetto a modifiche ed è destinato esclusivamente a scopi di valutazione.

Strumenti e librerie per Rust

- AWS SDK for Rust: L' AWS SDK per Rust fornisce API Rust per interagire con i servizi di infrastruttura Amazon Web Services.
- Client di runtime Rust per Lambda: il client di runtime Rust è un pacchetto sperimentale. È soggetto a modifiche e non è consigliato per l'uso in ambiente di produzione.
- Cargo Lambda: questa libreria fornisce un'applicazione a riga di comando per lavorare con le funzioni Lambda create con Rust.
- 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 dei tipi per le integrazioni di sorgenti di eventi comuni.

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 di Rust senza server: un progetto Rust che mostra l'uso delle librerie Rust di Lambda, la registrazione, le variabili di ambiente e l'SDK. AWS
- Estensione Lambda di base: un'estensione Rust che mostra come elaborare gli eventi di estensione di base.
- Lambda Logs Amazon Data Firehose Extension: un'estensione Rust che mostra come inviare log Lambda a Firehose.

Argomenti

Definisci il gestore di funzioni Lambda in Rust

> **Note**
>
> Il client di runtime Rust è un pacchetto sperimentale. È soggetto a modifiche ed è destinato esclusivamente a scopi di valutazione.

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.

Scrivi il codice della tua funzione Lambda come eseguibile Rust. Implementa il codice della funzione del gestore e una funzione principale e includi quanto segue:

- La cassa lambda_runtime di crates.io, che implementa il modello di programmazione Lambda per Rust.
- Includi Tokio nelle tue dipendenze. Il client di runtime Rust per Lambda utilizza Tokio per gestire le chiamate asincrone.

Example - Gestore di Rust che elabora eventi JSON

L'esempio seguente utilizza la cassa serde_json per elaborare eventi JSON di base:

```
use lambda_runtime::{service_fn, LambdaEvent, Error};
use serde_json::{json, Value};

async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
 let payload = event.payload;
 let first_name = payload["firstName"].as_str().unwrap_or("world");
 Ok(json!({ "message": format!("Hello, {first_name}!") }))
}

#[tokio::main]
async fn main() -> Result<(), Error> {
 lambda_runtime::run(service_fn(handler)).await
}
```

Tieni presente quanto segue:

- `use`: importa le librerie richieste dalla funzione Lambda.
- `async fn main`: il punto di ingresso che esegue il codice della funzione Lambda. Il client di runtime Rust utilizza Tokio come runtime asincrono, quindi è necessario annotare la funzione principale con `#[tokio::main]`.
- `async fn handler(event: LambdaEvent<Value>) -> Result<Value,Error>`: questa è la firma del gestore Lambda. Include del codice che viene eseguito quando la funzione viene richiamata.
  - `LambdaEvent<Value>`: si tratta di un tipo generico che descrive l'evento ricevuto dal runtime Lambda e dal contesto della funzione Lambda.
  - `Result<Value, Error>`: la funzione restituisce un tipo `Result`. Se la funzione ha esito positivo, il risultato è un valore JSON. Se la funzione ha esito negativo, il risultato è un errore.

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.

Example - Condivisione del client Amazon S3 tra istanze di funzione

Tieni presente quanto segue:

- `use aws_sdk_s3::Client`: questo esempio richiede l'aggiunta di `aws-sdk-s3 = "0.26.0"` all'elenco delle dipendenze nel file `Cargo.toml`.
- `aws_config::from_env`: questo esempio richiede l'aggiunta di `aws-config = "0.55.1"` all'elenco delle dipendenze nel file `Cargo.toml`.

```
use aws_sdk_s3::Client;
use lambda_runtime::{service_fn, Error, LambdaEvent};
use serde::{Deserialize, Serialize};

#[derive(Deserialize)]
struct Request {
    bucket: String,
}

#[derive(Serialize)]
```

```
struct Response {
    keys: Vec<String>,
}

async fn handler(client: &Client, event: LambdaEvent<Request>) -> Result<Response,
 Error> {
    let bucket = event.payload.bucket;
    let objects = client.list_objects_v2().bucket(bucket).send().await?;
    let keys = objects
        .contents()
        .map(|s| s.iter().flat_map(|o| o.key().map(String::from)).collect())
        .unwrap_or_default();
    Ok(Response { keys })
}

#[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
}
```

Oggetto di contesto Lambda in Rust

> **Note**
>
> Il client di runtime Rust è un pacchetto sperimentale. È soggetto a modifiche ed è destinato esclusivamente a scopi di valutazione.

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 richiesta AWS generata 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 della traccia AWS X-Ray della chiamata corrente.
- `client_content`: l'oggetto di contesto del client inviato dall'SDK AWS per dispositivi mobili. Questo campo è vuoto a meno che la funzione non venga richiamata utilizzando un SDK AWS per dispositivi mobili.
- `identity`: l'identità Amazon Cognito che ha richiamato la funzione. Questo campo è vuoto a meno che la richiesta di chiamata alle API Lambda non sia stata effettuata utilizzando le credenziali AWS 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 `LambaEvent` 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

> **Note**
>
> Il client di runtime Rust è un pacchetto sperimentale. È soggetto a modifiche ed è destinato esclusivamente a scopi di valutazione.

Le API di Gateway Amazon API, gli Application Load Balancer e URL della funzione Lambda 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

Tieni 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 consente di lavorare con 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`, consultate l'esempio di codice http-axum nel AWS repository Labs. 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

> **Note**
>
> Il client di runtime Rust è un pacchetto sperimentale. È soggetto a modifiche ed è destinato esclusivamente a scopi di valutazione.

Questa pagina descrive come compilare la funzione Rust e quindi implementare il file binario compilato su AWS Lambda utilizzando Cargo Lambda. Mostra anche come implementare il file binario compilato con la AWS Command Line Interface e la CLI AWS Serverless Application Model.

Sections

Prerequisiti

- Rust
- AWS Command Line Interface (AWS CLI) versione 2

Compilazione di funzioni Rust su macOS, Windows o Linux

La procedura seguente mostra come creare il progetto per la prima funzione Lambda con Rust e compilarlo con Cargo Lambda.

1. Installa Cargo Lambda, un sottocomando Cargo, che compila le funzioni Rust per Lambda su macOS, Windows e Linux.

   Per installare Cargo Lambda su qualsiasi sistema in cui è installato Python 3, utilizza pip:

   ```
   pip3 install cargo-lambda
   ```

Per installare Cargo Lambda su macOS o Linux, utilizza Homebrew:

```
brew tap cargo-lambda/cargo-lambda
brew install cargo-lambda
```

Per installare Cargo Lambda su Windows, utilizza Scoop:

```
scoop bucket add cargo-lambda
scoop install cargo-lambda/cargo-lambda
```

Per altre opzioni, consulta la pagina 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 desideri utilizzare AWS Graviton2 su Lambda, aggiungi il flag `--arm64` per compilare il codice per le CPU ARM.

```
cargo lambda build --release --arm64
```

4. Prima di implementare la funzione Rust, configura le credenziali AWS sul tuo computer.

```
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
```

Implementazione del file binario delle funzioni Rust con la AWS CLI

Puoi implementare il file binario anche con la AWS CLI.

1. Per compilare il pacchetto di implementazione .zip, utilizza il sottocomando build.

   ```
   cargo lambda build --release --output-format zip
   ```

2. Implementa il pacchetto .zip 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::111122223333:role/lambda-role \
        --handler rust.handler \
        --zip-file fileb://target/lambda/my-function/bootstrap.zip
   ```

Implementazione del file binario delle funzioni Rust con la CLI AWS SAM

Puoi implementare il file binario anche con la CLI AWS SAM.

1. Crea un modello AWS SAM con la definizione della risorsa e della proprietà. Per ulteriori informazioni, consulta la sezione AWS::Serverless::Function nella Guida per gli sviluppatori di AWS Serverless Application Model.

   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 CLI AWS SAM, consulta la pagina Compilazione di funzioni Lambda Rust con Cargo Lambda nella Guida per gli sviluppatori di 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
```

Richiamo della funzione Rust con la AWS CLI

È inoltre possibile utilizzare 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'opzione cli-binary-format è necessaria 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 la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Registrazione della funzione Lambda in Rust

> **Note**
>
> Il client di runtime Rust è un pacchetto sperimentale. È soggetto a modifiche ed è destinato esclusivamente a scopi di valutazione.

AWS Lambdamonitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della 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 Utilizzo dei CloudWatch log di Amazon con AWS Lambda. 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

Tieni 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"}]}
```

Richiamare Lambda con eventi di altri servizi AWS

Alcuni AWS servizi 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 crei 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.

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, la chiamata 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 elabora l'evento, Lambda non restituisce una risposta al servizio generatore di eventi.

Creazione di un trigger

Il modo più semplice per creare un trigger è utilizzare la console Lambda. Quando crei un trigger utilizzando la console, Lambda aggiunge automaticamente le autorizzazioni richieste alla politica 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 che desideri richiamare la tua funzione.
5. Compila le opzioni nel riquadro di configurazione Trigger e scegli Aggiungi. A seconda della Servizio AWS funzione scelta per richiamare la funzione, le opzioni di configurazione del trigger saranno diverse.

Servizi che possono richiamare funzioni Lambda

La tabella seguente elenca i servizi che possono richiamare le funzioni Lambda.

Servizio	Metodo di chiamata
Amazon Alexa	Chiamata sincrona basata su eventi
Amazon Managed Streaming per Apache Kafka	Mappatura delle sorgenti degli eventi
Apache Kafka gestito dal cliente	Mappatura delle sorgenti degli eventi
Gateway Amazon API	Chiamata sincrona basata su eventi
AWS CloudFormation	Chiamata asincrona basata su eventi
Amazon CloudFront (Lambda @Edge)	Chiamata sincrona basata su eventi
CloudWatch Registri Amazon	Chiamata asincrona basata su eventi
AWS CodeCommit	Chiamata asincrona basata su eventi
AWS CodePipeline	Chiamata asincrona basata su eventi
Amazon Cognito	Chiamata sincrona basata su eventi
AWS Config	Chiamata asincrona basata su eventi
Amazon Connect	Chiamata sincrona basata su eventi

Servizio	Metodo di chiamata
Amazon DynamoDB	Mappatura delle sorgenti degli eventi
Amazon Elastic File System	Integrazione speciale
Elastic Load Balancer (Application Load Balancer)	Chiamata sincrona basata su eventi
AWS IoT	Chiamata asincrona basata su eventi
Amazon Kinesis	Mappatura delle sorgenti degli eventi
Amazon Data Firehose	Chiamata sincrona basata su eventi
Amazon Lex	Chiamata sincrona basata su eventi
Amazon MQ	Mappatura delle sorgenti degli eventi
Amazon Simple Email Service	Chiamata asincrona basata su eventi
Amazon Simple Notification Service	Chiamata asincrona basata su eventi
Amazon Simple Queue Service	Mappatura delle sorgenti degli eventi
Amazon Simple Storage Service (Amazon S3)	Chiamata asincrona basata su eventi
Batch di Amazon Simple Storage Service	Chiamata sincrona basata su eventi
Secrets Manager	Chiamata sincrona basata su eventi
Amazon VPC Lattice	Chiamata sincrona basata su eventi
AWS X-Ray	Integrazione speciale

Casi d'uso e tipi di applicazioni Lambda comuni

Costruendo applicazioni in AWS Lambda, i componenti principali sono le funzioni e le attivazioni Lambda. Una funzione Lambda è il codice e il tempo di esecuzione che elabora gli eventi, mentre un'attivazione è il servizio o l'applicazione AWS che richiama la funzione. A titolo illustrativo, si considerino gli scenari riportati di seguito:

- Elaborazione di file – Si supponga di avere un'applicazione per la condivisione di foto. Le persone utilizzano l'applicazione per caricare foto e l'applicazione memorizza queste foto degli utenti in un bucket Amazon S3. Quindi, l'applicazione crea una versione di anteprima delle foto di ciascun utente e le visualizza nella pagina del profilo dell'utente. In questo scenario si può scegliere di creare una funzione Lambda che crea automaticamente un'anteprima. Amazon S3 è una delle origini eventi AWS supportate che possono pubblicare eventi di creazione di un oggetto e invocare la funzione Lambda. Il codice della funzione Lambda può leggere l'oggetto foto dal bucket S3, creare una versione di anteprima e salvarla in un altro bucket S3.
- Dati e analisi – Si supponga di creare un'applicazione di analisi e di memorizzare i dati non elaborati in una tabella DynamoDB. Quando si scrivono, si aggiornano o si eliminano voci in una tabella, Dynamo DB Streams può pubblicare gli eventi di aggiornamento delle voci in un flusso associato alla tabella. In questo caso, i dati dell'evento forniscono la chiave della voce, il nome dell'evento, ad esempio inserimento, aggiornamento ed eliminazione, nonché altri dettagli rilevanti. È possibile scrivere una funzione Lambda per generare parametri personalizzati aggregando dati non elaborati.
- Siti Web – Si supponga di creare un sito Web e di voler ospitare la logica back-end in Lambda. È possibile invocare la funzione Lambda su HTTP utilizzando Amazon API Gateway come endpoint HTTP. Il client Web può invocare l'API, quindi API Gateway può instradare la richiesta a Lambda.
- Applicazioni per dispositivi mobili – Si supponga di avere un'applicazione per dispositivi mobili che produce eventi. È possibile creare una funzione Lambda per elaborare eventi pubblicati dall'applicazione personalizzata. Ad esempio, è possibile configurare una funzione Lambda per elaborare i clic nell'applicazione per dispositivi mobili personalizzata.

AWS Lambda supporta molti servizi AWS come origini eventi. Per ulteriori informazioni, consulta Richiamare Lambda con eventi di altri servizi AWS. Quando si configurano tali origini eventi per attivare una funzione Lambda, la funzione Lambda viene invocata automaticamente quando si verificano gli eventi. Si definisce la mappatura origine evento, ovvero la modalità con cui si identificano gli eventi da monitorare e quale funzione Lambda invocare.

Di seguito sono riportati alcuni esempi introduttivi delle fonti di eventi e del funzionamento dell' end-to-end esperienza.

Esempio 1: Amazon S3 effettua il push degli eventi e invoca una funzione Lambda

Amazon S3 può pubblicare eventi di tipo diverso, ad esempio eventi di PUT, POST, COPY e DELETE di oggetti in un bucket. Mediante la funzione di notifica del bucket, è possibile configurare una mappatura origine evento che indirizza Amazon S3 per invocare una funzione Lambda quando si verifica un tipo di evento specifico.

Di seguito è riportata una sequenza tipica:

1. L'utente crea un oggetto in un bucket.
2. Amazon S3 rileva l'evento di creazione di un oggetto.
3. Amazon S3 richiama la funzione Lambda utilizzando le autorizzazioni fornite dal ruolo di esecuzione.
4. AWS Lambda esegue la funzione Lambda specificando l'evento come parametro.

È possibile configurare Amazon S3 per richiamare la funzione come operazione di notifica del bucket. Per concedere ad Amazon S3 l'autorizzazione per richiamare la funzione, aggiorna la policy basata su risorse della funzione.

Esempio 2: AWS Lambda effettua il pull degli eventi da un flusso Kinesis e invoca una funzione Lambda

Per le origini eventi basate su polling, AWS Lambda esegue il polling dell'origine e invoca la funzione Lambda quando vengono rilevati record su tale origine.

- CreateEventSourceMapping
- UpdateEventSourceMapping

I seguenti passaggi illustrano come un'applicazione personalizzata scrive record in un flusso Kinesis.

1. L'applicazione personalizzata scrive record su un flusso Kinesis.

2. AWS Lambda esegue il polling del flusso in modo continuo e invoca la funzione Lambda quando il servizio rileva nuovi record nel flusso. AWS Lambda rileva il flusso di cui eseguire il polling e la funzione Lambda da invocare in base alla mappatura origine evento creata in Lambda.
3. La funzione Lambda viene invocata con l'evento in ingresso.

Quando si utilizzano origini eventi basate sul flusso, si creano le mappature delle origini eventi in AWS Lambda. Lambda legge gli elementi del flusso e richiama la funzione in modo sincrono. Non è necessario concedere a Lambda l'autorizzazione per richiamare la funzione, ma l'autorizzazione è richiesta per la lettura del flusso.

Utilizzo di AWS Lambda con Alexa

È possibile utilizzare le funzioni Lambda per creare servizi che forniscano nuove funzionalità ad Alexa, l'assistente vocale in Amazon Echo. Alexa Skills Kit fornisce le API, gli strumenti e la documentazione per creare le nuove funzionalità, supportate dai servizi personalizzati in esecuzione come funzioni Lambda. Gli utenti di Amazon Echo possono accedere alle nuove funzionalità rivolgendo domande o inviando richieste ad Alexa.

L'Alexa Skills Kit è disponibile su GitHub.

- Alexa Skills Kit SDK per Java
- Alexa Skills Kit SDK per Node.js
- SDK per Python per Alexa Skills Kit

Example Evento Alexa Smart Home

```
{
  "header": {
    "payloadVersion": "1",
    "namespace": "Control",
    "name": "SwitchOnOffRequest"
  },
  "payload": {
    "switchControlAction": "TURN_ON",
    "appliance": {
      "additionalApplianceDetails": {
        "key2": "value2",
        "key1": "value1"
      },
      "applianceId": "sampleId"
    },
    "accessToken": "sampleAccessToken"
  }
}
```

Per ulteriori informazioni, consulta Ospitare un'abilità personalizzata come funzione AWS Lambda nella guida Build Skills with Alexa Skills Kit.

Richiamo di una funzione Lambda utilizzando un endpoint Amazon API Gateway

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.

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

API Gateway supporta tre tipi di API che richiamano le funzioni Lambda:

- API HTTP: un'API RESTful leggera e a bassa latenza.
- API REST: un'API RESTful personalizzabile e ricca di funzionalità.
- WebSocket API: un'API Web che mantiene connessioni persistenti 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 opzioni di AWS X-Ray tracciamento e registrazione aggiuntive.

Per un confronto dettagliato, consultare 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). L'esempio seguente mostra un evento da un'API HTTP.

Example event-v2.json - evento proxy di API Gateway (HTTP API)

```
{
    "version": "2.0",
    "routeKey": "ANY /nodejs-apig-function-1G3XMPLZXVXYI",
    "rawPath": "/default/nodejs-apig-function-1G3XMPLZXVXYI",
    "rawQueryString": "",
    "cookies": [
        "s_fid=7AABXMPL1AFD9BBF-0643XMPL09956DE2",
        "regStatus=pre-register"
    ],
    "headers": {
        "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
        "accept-encoding": "gzip, deflate, br",
        ...
    },
    "requestContext": {
        "accountId": "123456789012",
        "apiId": "r3pmxmplak",
        "domainName": "r3pmxmplak.execute-api.us-east-2.amazonaws.com",
        "domainPrefix": "r3pmxmplak",
        "http": {
            "method": "GET",
            "path": "/default/nodejs-apig-function-1G3XMPLZXVXYI",
            "protocol": "HTTP/1.1",
            "sourceIp": "205.255.255.176",
            "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"
        },
        "requestId": "JKJaXmPLvHcESHA=",
        "routeKey": "ANY /nodejs-apig-function-1G3XMPLZXVXYI",
        "stage": "default",
        "time": "10/Mar/2020:05:16:23 +0000",
        "timeEpoch": 1583817383220
    },
    "isBase64Encoded": true
}
```

Per ulteriori informazioni, consultare Integrazioni di AWS Lambda nella Guida per gli sviluppatori di 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, consultare API types (Tipi di API).
    b. API esistente: selezionare l'API dal menu a discesa o inserire 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 Configurare le integrazioni del proxy Lambda in 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. L'esempio seguente mostra un evento proxy da una REST API di API Gateway.

Example Evento proxy event.json di API Gateway (REST API)

```
{
      "resource": "/",
      "path": "/",
      "httpMethod": "GET",
      "requestContext": {
          "resourcePath": "/",
          "httpMethod": "GET",
          "path": "/Prod/",
          ...
      },
      "headers": {
          "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
          "accept-encoding": "gzip, deflate, br",
          "Host": "70ixmpl4fl.execute-api.us-east-2.amazonaws.com",
          "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36",
          "X-Amzn-Trace-Id": "Root=1-5e66d96f-7491f09xmpl79d18acf3d050",
          ...
      },
      "multiValueHeaders": {
          "accept": [
              "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"
          ],
          "accept-encoding": [
              "gzip, deflate, br"
          ],
          ...
      },
      "queryStringParameters": null,
      "multiValueQueryStringParameters": null,
      "pathParameters": null,
      "stageVariables": null,
```

```
      "body": null,
      "isBase64Encoded": false
  }
```

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 risposta da una funzione Node.js. L'oggetto risposta rappresenta una risposta HTTP riuscita che contiene un documento JSON.

Example index.mjs: oggetto risposta di integrazione 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 tempo di esecuzione 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
  }
```

Autorizzazioni

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

```
{
  "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`.

```
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 di 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 in modo che corrisponda 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.

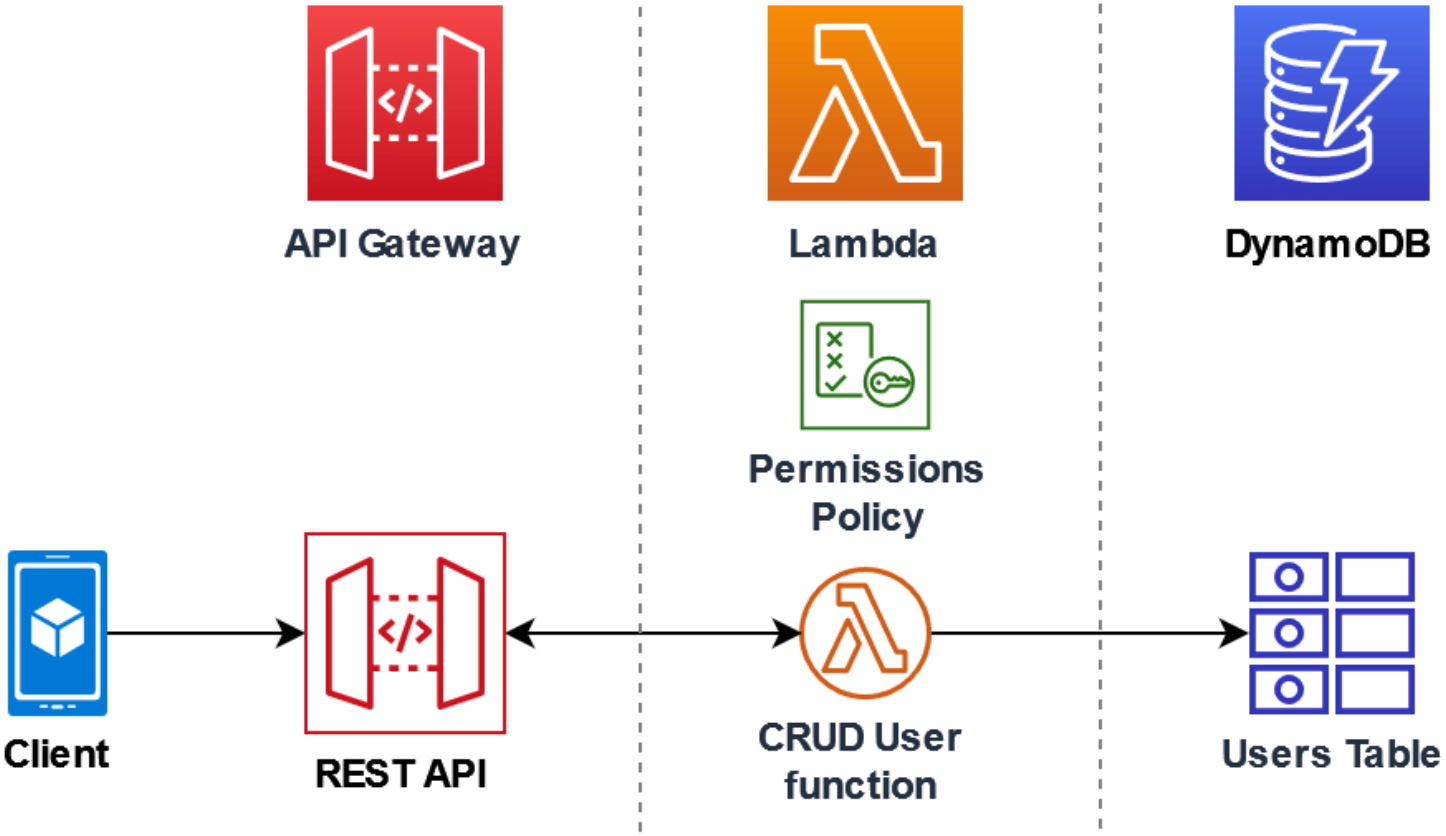
Per informazioni dettagliate sulla visualizzazione delle policy e sulla rimozione delle istruzioni, vedere Pulizia delle policy basate sulle risorse.

Applicazione di esempio

L'app di esempio API Gateway with Node.js include una funzione con un AWS SAM modello che crea un'API REST con AWS X-Ray tracciamento abilitato. Include anche script per la distribuzione, il richiamo della funzione, il test dell'API e la pulizia.

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 eseguirà operazioni di creazione, aggiornamento ed eliminazione (CRUD) su una tabella DynamoDB. Questa funzione viene fornita qui a scopo dimostrativo, ma imparerai a configurare una REST API di Gateway API 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.

Per completare questo tutorial, saranno completate 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, imparerai come utilizzare Gateway API 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.

Sections

Prerequisiti

Registrati per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come procedura consigliata in materia di sicurezza, assegnate l'accesso amministrativo a un utente e utilizzate solo l'utente root per eseguire attività che richiedono l'accesso da parte dell'utente root.

AWS ti invia un'e-mail di conferma dopo il completamento della procedura di registrazione. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .
2. In IAM Identity Center, concedi l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con le impostazioni predefinite IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accedi come utente con accesso amministrativo

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso ad altri utenti

1. In IAM Identity Center, crea un set di autorizzazioni che segua la migliore pratica di applicazione delle autorizzazioni con privilegi minimi.

   Per istruzioni, consulta Creare un set di autorizzazioni nella Guida per l'utente.AWS IAM Identity Center
2. Assegna gli utenti a un gruppo, quindi assegna l'accesso Single Sign-On al gruppo.

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente.AWS IAM Identity Center

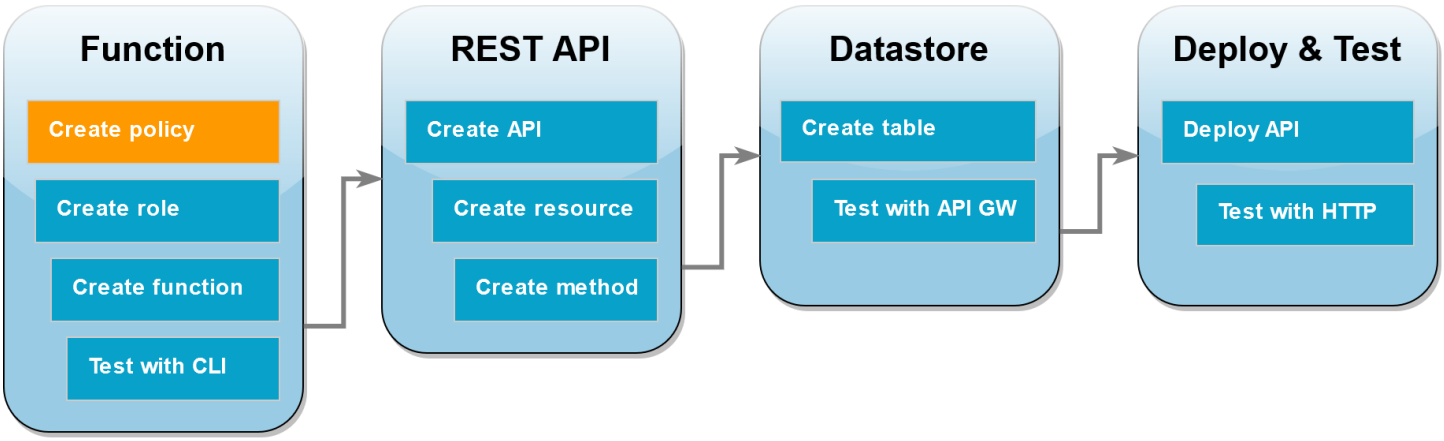
Installa il AWS Command Line Interface

Se non l'hai ancora installato AWS Command Line Interface, segui i passaggi indicati in Installazione o aggiornamento della versione più recente di 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 versione integrata su Windows di Ubuntu e Bash, installa il sottosistema Windows per Linux.

Creazione di una policy di autorizzazione



Prima di poter creare un ruolo di esecuzione per la tua funzione Lambda, devi prima creare una politica di autorizzazioni per concedere alla 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

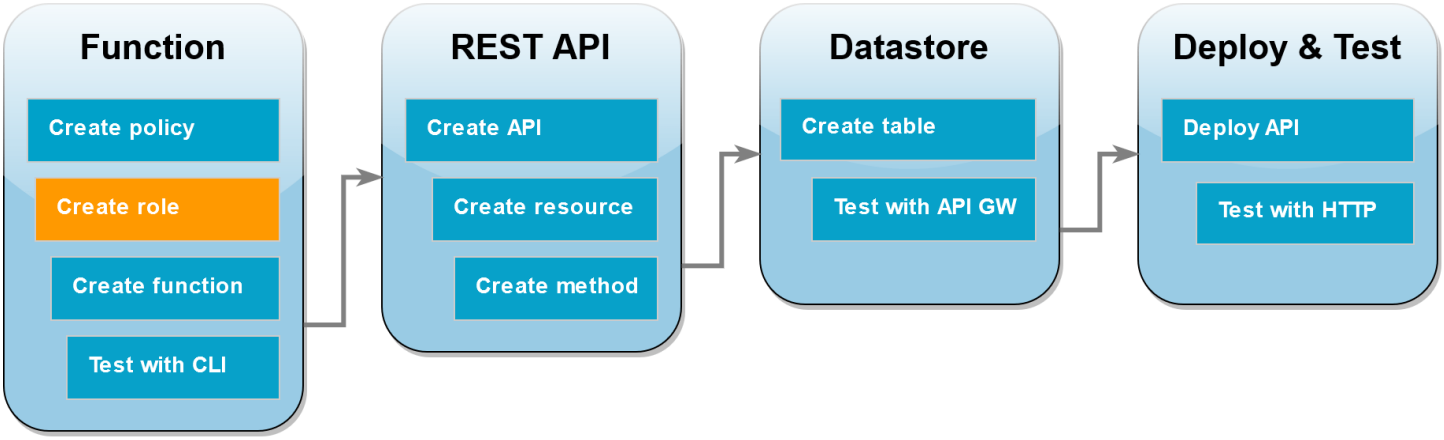
1. Apri la pagina Policies (Policy) nella console IAM.
2. Scegliere Create Policy (Crea policy).
3. Scegliere la scheda JSON e quindi incollare la seguente policy personalizzata nell'editor JSON.

```
{
  "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 ad AWS accedere a servizi e risorse. Per consentire alla funzione di eseguire operazioni su una tabella DynamoDB, collega la policy di autorizzazione che hai creato nella fase precedente.

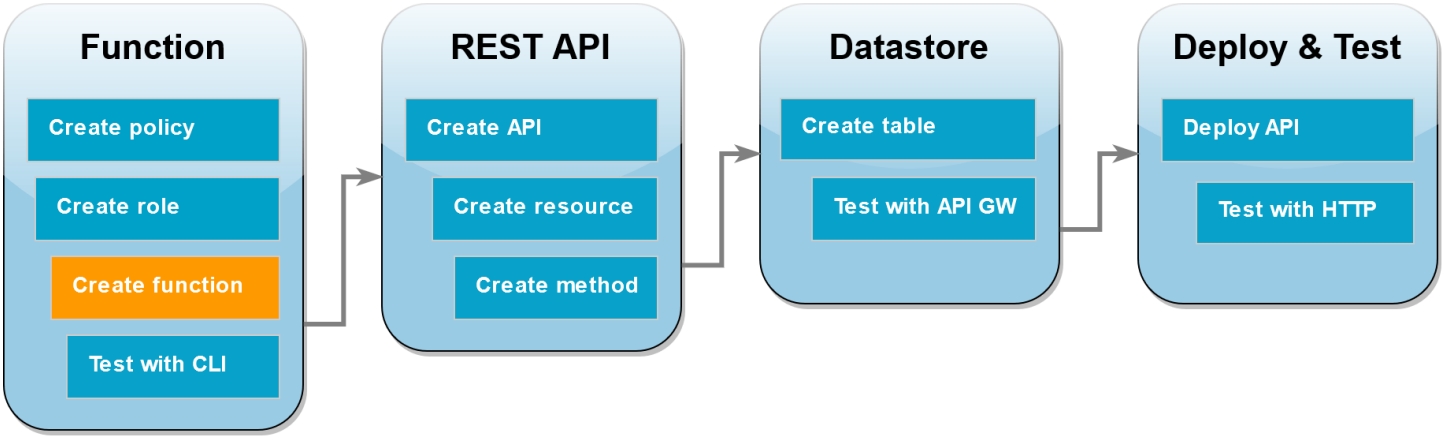
Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione personalizzata

1. Aprire la pagina Roles (Ruoli) della console IAM.
2. Scegliere Create role (Crea ruolo).
3. Per il tipo di entità attendibile, scegli Servizio AWS , quindi per il caso d'uso seleziona Lambda.

4. Seleziona Successivo.
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).

Più avanti nel tutorial sarà necessario il nome della risorsa Amazon (ARN) del ruolo appena creato. Nella pagina Roles (Ruoli) della console IAM, scegli il nome del ruolo (`lambda-apigateway-role`) e copia l'ARN del ruolo visualizzato nella pagina Summary (Riepilogo).

Creazione della funzione



Il seguente esempio di codice riceve un input di evento da Gateway API che specifica un'operazione da eseguire sulla tabella DynamoDB che verrà creata e alcuni dati del payload. Se i parametri ricevuti dalla funzione sono validi, l'operazione richiesta viene eseguita sulla tabella.

Node.js

Example index.mjs

```
console.log('Loading function');

import { DynamoDBDocumentClient, PutCommand, GetCommand,
         UpdateCommand, DeleteCommand} from "@aws-sdk/lib-dynamodb";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const ddbClient = new DynamoDBClient({ region: "us-west-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;

        switch (operation) {
          case 'create':
               await ddbDocClient.send(new PutCommand(event.payload));
               break;
          case 'read':
               var table_item = await ddbDocClient.send(new
 GetCommand(event.payload));
               console.log(table_item);
               break;
          case 'update':
               await ddbDocClient.send(new UpdateCommand(event.payload));
               break;
          case 'delete':
               await ddbDocClient.send(new DeleteCommand(event.payload));
               break;
          default:
            return ('Unknown 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 delle variabili di AWS Lambda ambiente.

Creazione della funzione

1. Salvate l'esempio di codice come file denominato `index.mjs` e, se necessario, modificate l' AWS area specificata nel codice. La regione specificata nel codice deve essere la stessa regione in cui verrà creata la tabella DynamoDB più avanti nel tutorial.
2. Crea un pacchetto di implementazione utilizzando il seguente comando `zip`.

```
zip function.zip index.mjs
```

3. Crea una funzione Lambda utilizzando il `create-function` AWS CLI comando. Per il parametro `role`, immetti il nome della risorsa Amazon (ARN) del ruolo di esecuzione copiato in precedenza.

```
aws lambda create-function \
--function-name LambdaFunctionOverHttps \
--zip-file fileb://function.zip \
--handler index.handler \
--runtime nodejs20.x \
--role arn:aws:iam::123456789012:role/service-role/lambda-apigateway-role
```

Python 3

Example LambdaFunctionOverHttps.py

```
import boto3
import json

# define the DynamoDB table that Lambda will connect to
tableName = "lambda-apigateway"

# create the DynamoDB resource
```

```
dynamo = boto3.resource('dynamodb').Table(tableName)

print('Loading function')

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
    '''

    # define the functions used to perform the CRUD operations
    def ddb_create(x):
        dynamo.put_item(**x)

    def ddb_read(x):
        dynamo.get_item(**x)

    def ddb_update(x):
        dynamo.update_item(**x)

    def ddb_delete(x):
        dynamo.delete_item(**x)

    def echo(x):
        return x

    operation = event['operation']

    operations = {
        'create': ddb_create,
        'read': ddb_read,
        'update': ddb_update,
        'delete': ddb_delete,
        'echo': echo,
    }

    if operation in operations:
        return operations[operation](event.get('payload'))
    else:
        raise ValueError('Unrecognized operation "{}"'.format(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, vedere Utilizzo delle variabili di AWS Lambda ambiente.

Creazione della funzione

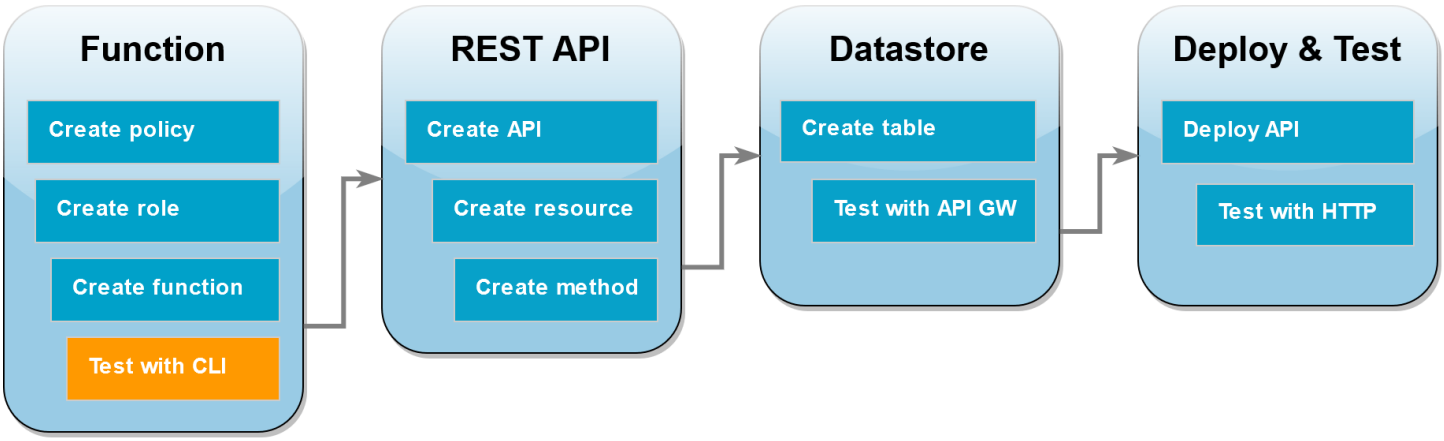
1. Salva l'esempio di codice come un file denominato `LambdaFunctionOverHttps.py`.
2. Crea un pacchetto di implementazione utilizzando il seguente comando `zip`.

```
zip function.zip LambdaFunctionOverHttps.py
```

3. Crea una funzione Lambda utilizzando il `create-function` AWS CLI comando. Per il parametro `role`, immetti il nome della risorsa Amazon (ARN) del ruolo di esecuzione copiato in precedenza.

```
aws lambda create-function \
--function-name LambdaFunctionOverHttps \
--zip-file fileb://function.zip \
--handler LambdaFunctionOverHttps.lambda_handler \
--runtime python3.12 \
--role arn:aws:iam::123456789012:role/service-role/lambda-apigateway-role
```

Richiamate la funzione utilizzando il AWS CLI



Prima di integrare la tua funzione con Gateway API, verifica di aver implementato correttamente la funzione. Crea un evento di test contenente i parametri che l'API API Gateway invierà a Lambda e usa il AWS CLI `invoke` comando per eseguire la funzione.

Per richiamare la funzione Lambda con AWS CLI

1. Salva il seguente JSON come un file denominato `input.txt`.

```
{
    "operation": "echo",
    "payload": {
        "somekey1": "somevalue1",
        "somekey2": "somevalue2"
    }
}
```

2. Eseguire il seguente comando `invoke` AWS CLI .

```
aws lambda invoke \
--function-name LambdaFunctionOverHttps \
--payload file://input.txt outputfile.txt \
--cli-binary-format raw-in-base64-out
```

   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.

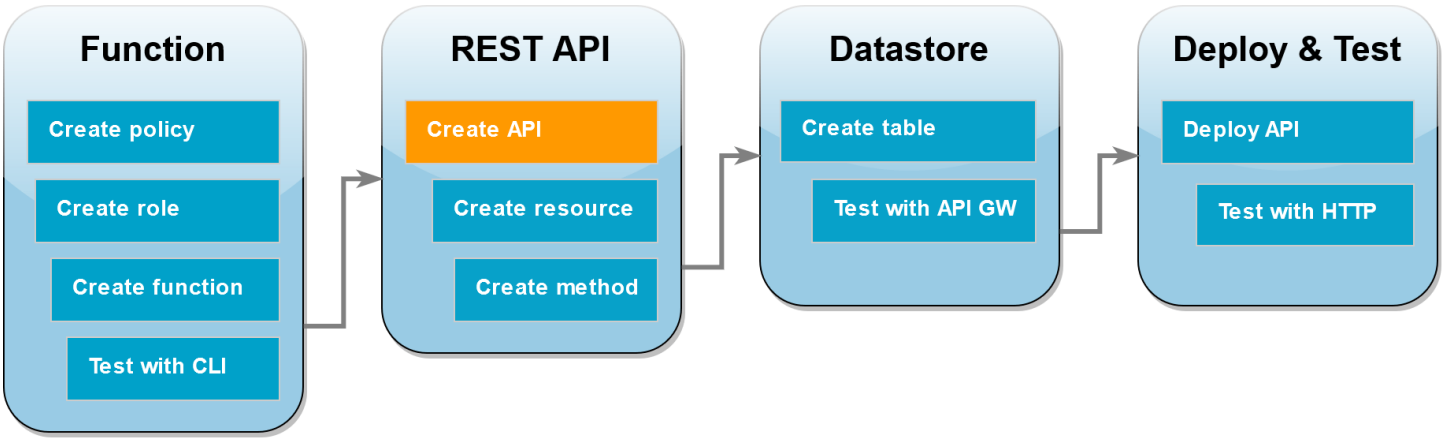
   Dovrebbe essere visualizzata la seguente risposta:

```
{
"StatusCode": 200,
"ExecutedVersion": "LATEST"
}
```

3. Verifica che la tua funzione abbia eseguito l'operazione `echo` specificata nell'evento di test JSON. Analizza il file `outputfile.txt` e verifica che contenga quanto segue:

```
{"somekey1": "somevalue1", "somekey2": "somevalue2"}
```

Creazione di una REST API utilizzando API Gateway

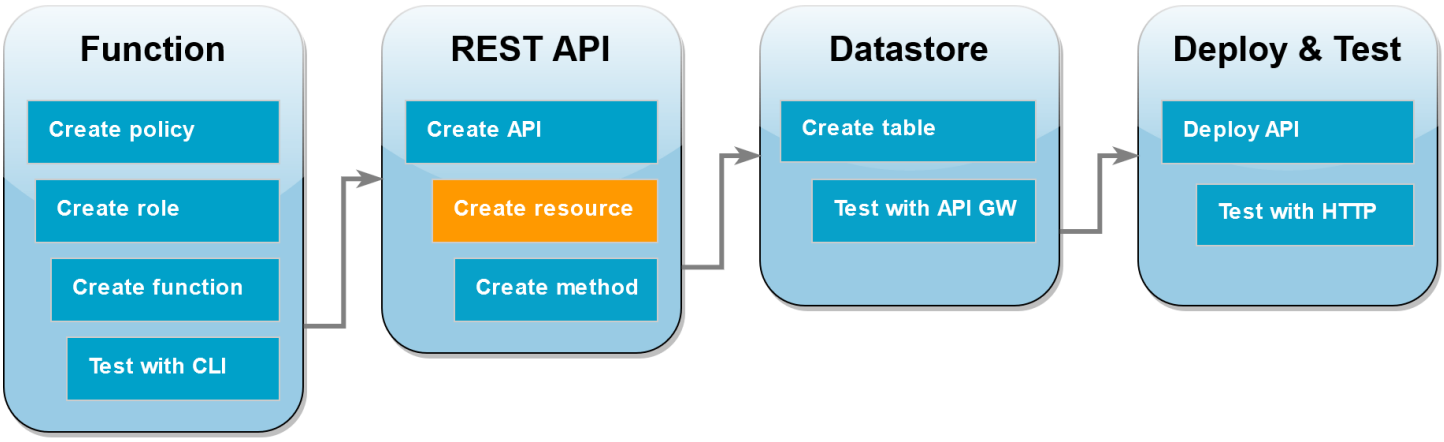


In questa fase, viene creata la REST API di Gateway API che sarà utilizzata per richiamare la funzione Lambda.

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

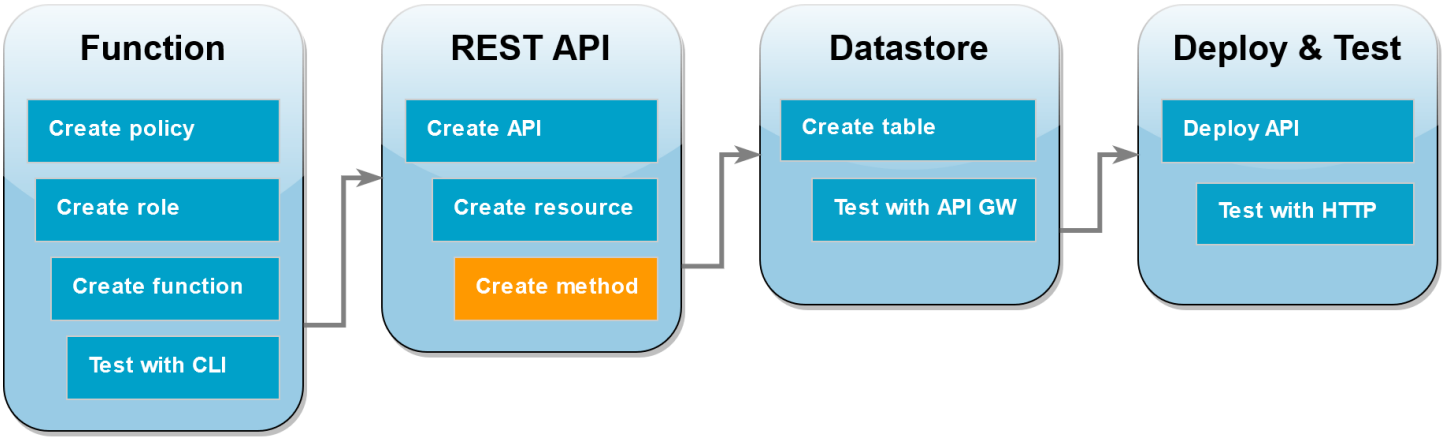


Per aggiungere un metodo HTTP all'API, è necessario prima creare una risorsa su cui quel metodo possa operare. Qui viene creata la risorsa per gestire la tabella DynamoDB.

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.

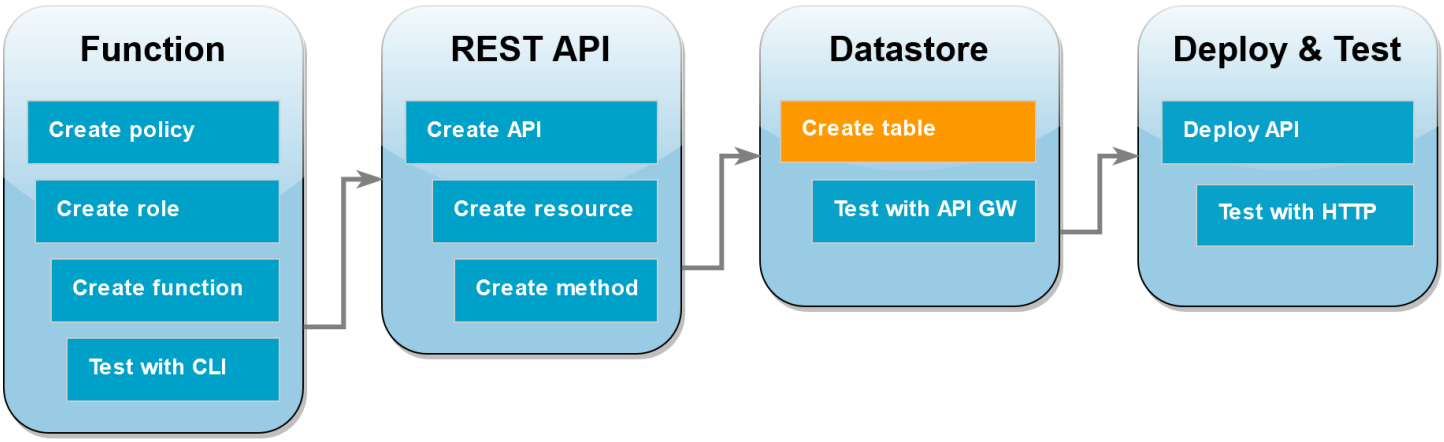
> **Note**
>
> Ai fini di questo tutorial, viene utilizzato un metodo HTTP (`POST`) per richiamare una singola funzione Lambda che esegue tutte le operazioni sulla tabella DynamoDB. In un'applicazione reale, la best practice consiste nell'utilizzare una funzione Lambda e un metodo HTTP diversi per ogni 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. 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

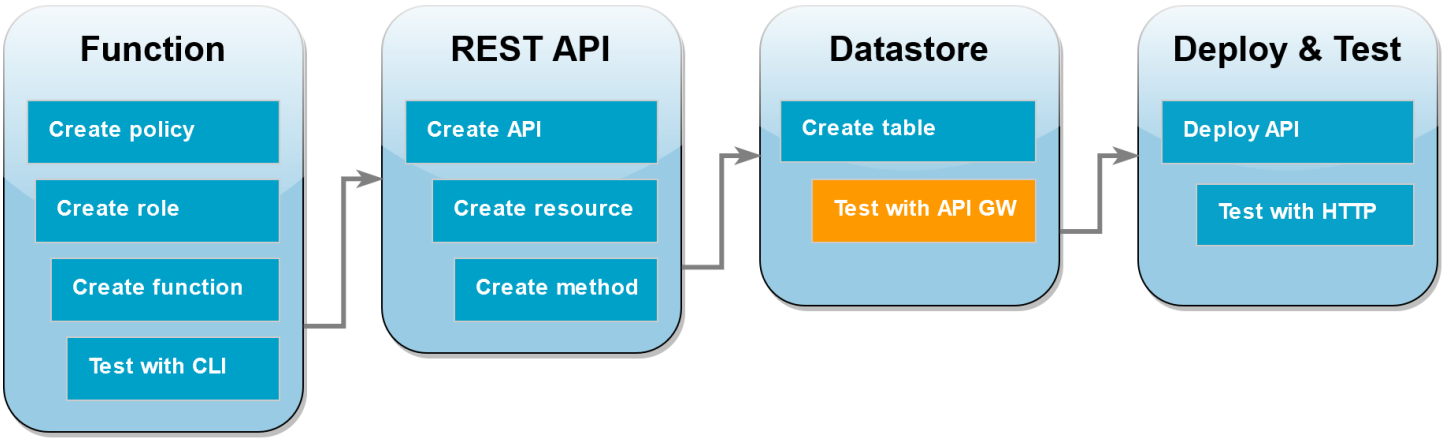


Crea una tabella DynamoDB vuota su cui la funzione Lambda eseguirà le operazioni CRUD.

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).

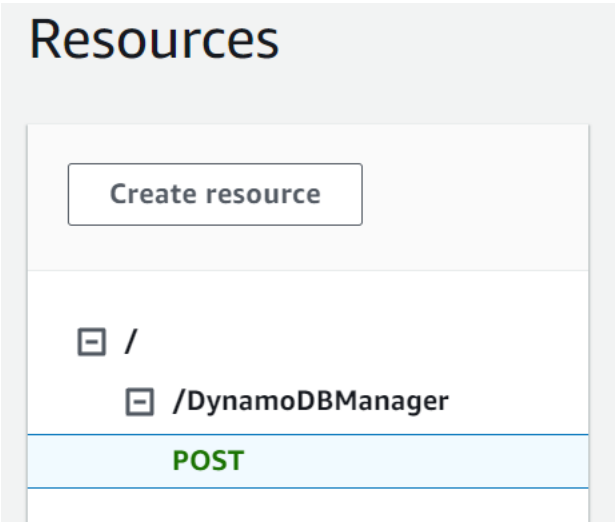
Test dell'integrazione di Gateway API, Lambda e DynamoDB



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 `DynamoDBManager` risorsa.



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`.

Test 2: aggiornamento dell'elemento nella tabella DynamoDB

1. Nella Console API Gateway, tornare alla scheda Test del metodo POST.
2. 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"
        },
        "AttributeUpdates": {
            "number": {
                "Value": 10
            }
        }
    }
```

```
}
```

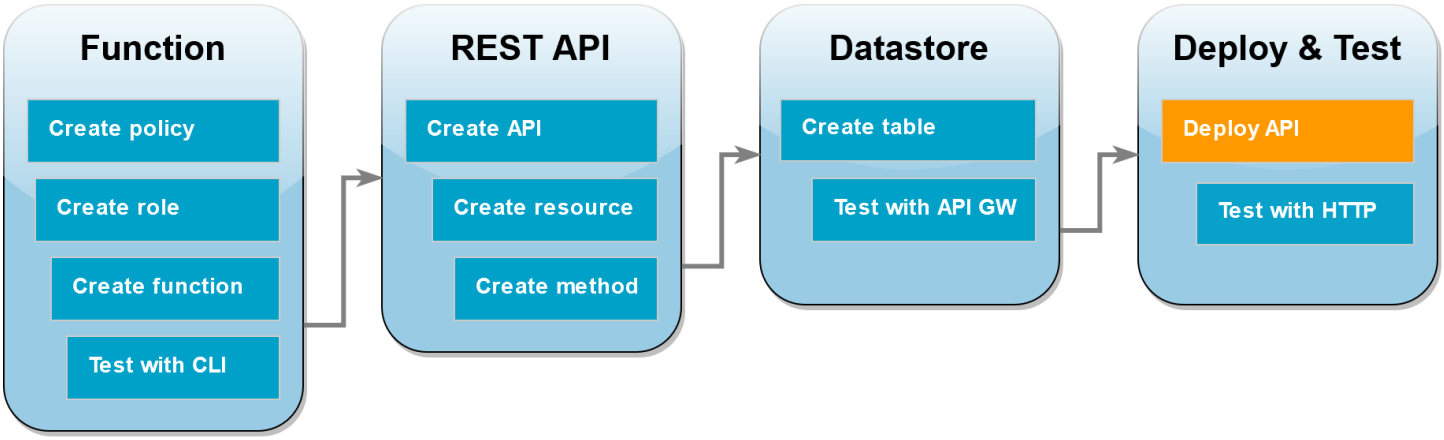
3. Scegli Test (Esegui test).

   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`.

Distribuzione dell'API

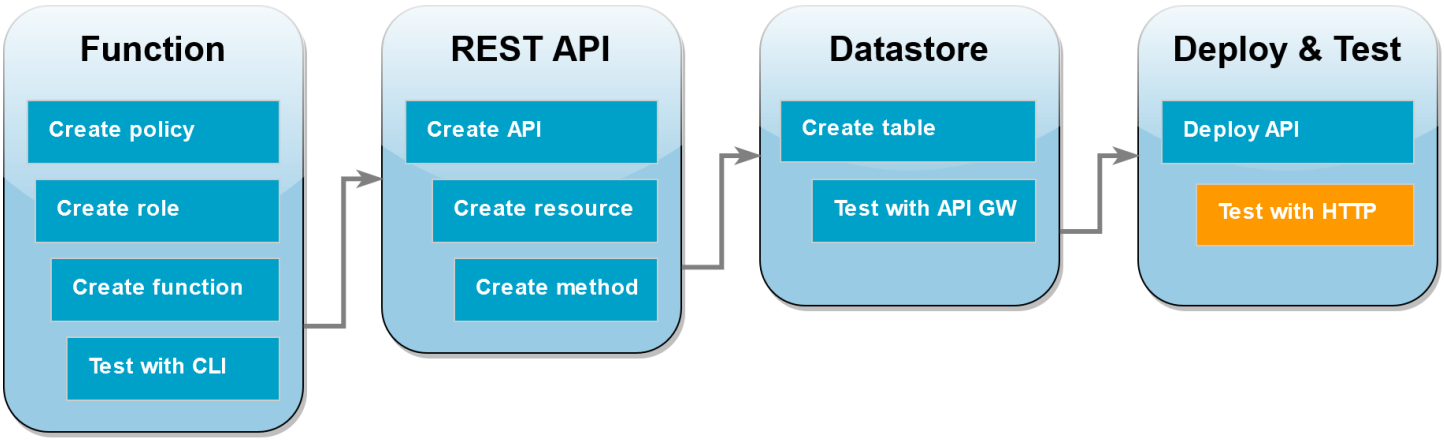


Per consentire al client di chiamare l'API, è necessario creare una implementazione e una fase associata. Una fase rappresenta un'istantanea dell'API, inclusi i suoi metodi e le sue integrazioni.

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 Deploy (Implementa).
5. Nel riquadro Editor fase (Editor fasi) del test, copia l'URL di richiamo. Questo verrà utilizzato nella fase successiva 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, verrà creato un nuovo elemento nella tabella DynamoDB e poi sarà eliminato.

Richiamo della funzione Lambda tramite curl

1. Esegui il comando `curl` riportato utilizzando l'URL di richiamo che hai copiato nel passaggio precedente. Quando si utilizza curl con l'opzione `-d` (dati), viene utilizzato automaticamente il metodo HTTP POST.

   ```
   curl https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/DynamoDBManager \
   -d '{"operation": "create", "payload": {"Item": {"id": "5678EFGH", "number": 15}}}'
   ```

2. Per verificare che l'operazione di creazione è andata a buon fine, procedi come segue:

   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`.

3. Esegui il comando `curl` riportato per eliminare l'elemento appena creato. Utilizzo dei propri URL di richiamo

   ```
   curl https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/DynamoDBManager \
   -d '{"operation": "delete", "payload": {"Key": {"id": "5678EFGH"}}}'
   ```

4. Verifica che l'operazione di eliminazione è andata a buon fine. Nel riquadro Items returned (Elementi restituiti) della pagina Explore items (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 addebiti inutili ai tuoi 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 **`delete`** 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

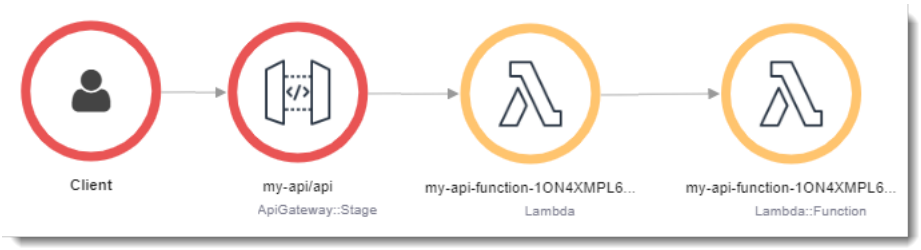
1. Aprire la pagina Tables (Tabelle) della console DynamoDB.
2. Selezionare la tabella creata.
3. Scegliere Delete (Elimina).
4. Immettere **`delete`** nella casella di testo.
5. Seleziona Delete Table (Elimina tabella).

Gestione degli errori Lambda con un'API Gateway API

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.

Nell'esempio seguente viene illustrata una mappa di tracciamento X-Ray per una richiesta che ha generato un errore di funzione e un codice 502 da API Gateway. Il client riceve il messaggio di errore generico.



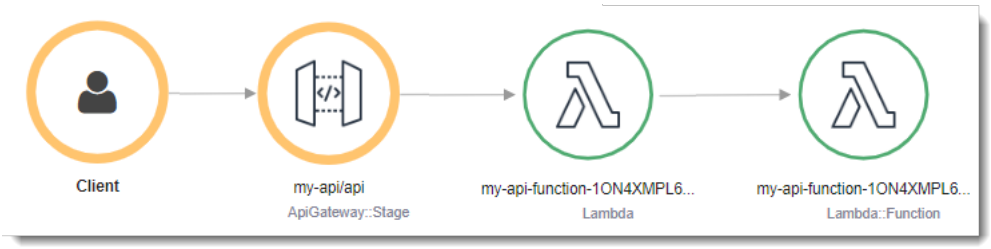
Per personalizzare la risposta di errore, è necessario rilevare gli errori nel codice e formattare una risposta nel formato richiesto.

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.



Utilizzo AWS Lambda con Strumento AWS per la creazione di applicazioni

Strumento AWS per la creazione di applicazioni è un generatore visivo per la progettazione di applicazioni moderne su. AWS Progetta l'architettura della tua applicazione trascinandola, raggruppandola e connettendola in un'area di disegno visiva. Servizi AWS Strumento per la creazione di applicazioni crea modelli di infrastructure as code (IaC) a partire dal tuo progetto che puoi implementare utilizzando AWS SAM oppure AWS CloudFormation.

Esportazione di una funzione Lambda in Strumento per la creazione di applicazioni

Per iniziare a utilizzare Strumento per la creazione di applicazioni, 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 Strumento per la creazione di applicazioni 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 Strumento per la creazione di applicazioni.
3. Nel riquadro Panoramica della funzione, scegli Esporta in Strumento per la creazione di applicazioni.

   Per esportare la configurazione e il codice della funzione in Strumento per la creazione di applicazioni, 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 Strumento per la creazione di applicazioni.
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 Strumento per la creazione di applicazioni, 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 di proprietario del bucket per garantire che solo il tuo 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 Eliminare un bucket. Il nome predefinito del bucket utilizza il prefisso`lambdasam`, una stringa alfanumerica di 10 cifre, e la funzione in cui hai creato la funzione in: Regione AWS

```
lambdasam-06f22da95b-us-east-1
```

Per evitare costi aggiuntivi Account AWS, ti consigliamo di eliminare il bucket Amazon S3 non appena hai finito di esportare la funzione in Application Composer.

Si applicano i prezzi standard di Amazon S3.

Autorizzazioni richieste

Per utilizzare l'integrazione Lambda con la funzionalità Application 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
- iam: GetPolicy versione
- sono: GetRole
- iam: GetRole Politica
- obiettivo: ListAttached RolePolicies
- iam: ListRole Politiche
- obiettivo: ListRoles

Puoi concedere l'autorizzazione a utilizzare tutte queste azioni aggiungendo la policy `AWSLambda_ReadOnlyAccess` AWS gestita 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: crittografia PutBucket
- S3: PutBucket LifecycleConfiguration

Se non riesci a esportare la configurazione della tua funzione in Strumento per la creazione di applicazioni, 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 Strumento per la creazione di applicazioni basata su una funzione Lambda esistente, consulta la pagina the section called “Infrastructure as code (IaC)”.

Per utilizzare Application Composer e AWS SAM progettare e distribuire un'applicazione serverless completa utilizzando Lambda, puoi anche seguire il Strumento AWS per la creazione di applicazioni tutorial nel Serverless Patterns Workshop.AWS

Usare Lambda con i log CloudWatch

Puoi utilizzare una funzione Lambda per monitorare e analizzare i log da un flusso di log di Amazon CloudWatch Logs. Crea le sottoscrizioni per uno o più flussi di log per richiamare una funzione quando i log vengono creati o corrispondono a un modello facoltativo. Utilizza la funzione per inviare una notifica o memorizza definitivamente il log in un database o uno storage.

CloudWatch Logs richiama la funzione in modo asincrono con un evento che contiene dati di log. Il valore del campo dati è un archivio .gzip con codifica Base64.

Example CloudWatch Registra l'evento del messaggio

```
{
  "awslogs": {
    "data":
 "ewogICAgIm1lc3NhZ2VUeXBlIjogIkRBVEFfTUVTU0FHRSIsCiAgICAib3duZXIiOiAiMTIzNDU2Nzg5MDEyIiwKICAgI
  }
}
```

Una volta decodificati e decompressi, i dati di log costituiscono un documento JSON con la struttura seguente:

Example CloudWatch Registra i dati dei messaggi (decodificati)

```
{
    "messageType": "DATA_MESSAGE",
    "owner": "123456789012",
    "logGroup": "/aws/lambda/echo-nodejs",
    "logStream": "2019/03/13/[$LATEST]94fa867e5374431291a7fc14e2f56ae7",
    "subscriptionFilters": [
        "LambdaStream_cloudwatchlogs-node"
    ],
    "logEvents": [
        {
            "id": "34622316099697884706540976068822859012661220141643892546",
            "timestamp": 1552518348220,
            "message": "REPORT RequestId: 6234bffe-149a-b642-81ff-2e8e376d8aff
\tDuration: 46.84 ms\tBilled Duration: 47 ms \tMemory Size: 192 MB\tMax Memory Used: 72
 MB\t\n"
        }
    ]
```

```
}
```

Utilizzo AWS Lambda con AWS 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 altri AWS servizi 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 AWS CloudFormation richiama quando crei, aggiorni o elimini lo stack. Puoi anche includere proprietà aggiuntive come`FunctionName`, che vengono AWS CloudFormation passate alla funzione così com'è.

AWS CloudFormation richiama la funzione Lambda in modo asincrono con un evento che include un URL di callback.

Example — AWS CloudFormation evento del messaggio

```
{
    "RequestType": "Create",
    "ServiceToken": "arn:aws:lambda:us-east-1:123456789012:function:lambda-error-
processor-primer-14ROR2T3JKU66",
    "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-14ROR2T3JKU66",
        "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 AWS 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"
}
```

AWS 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. AWS 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 AWS 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 AWS CloudFormation e l'aggiornamento dello stack continua. Il modello utilizza il tipo di AWS::Serverless::Functionrisorsa fornito da. AWS Serverless Application Model

Example — Funzione di risorse personalizzata

```
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, è possibile ottenere il codice sorgente `cfn-response` dal modulo cfn-response nella Guida per l'utente. AWS CloudFormation

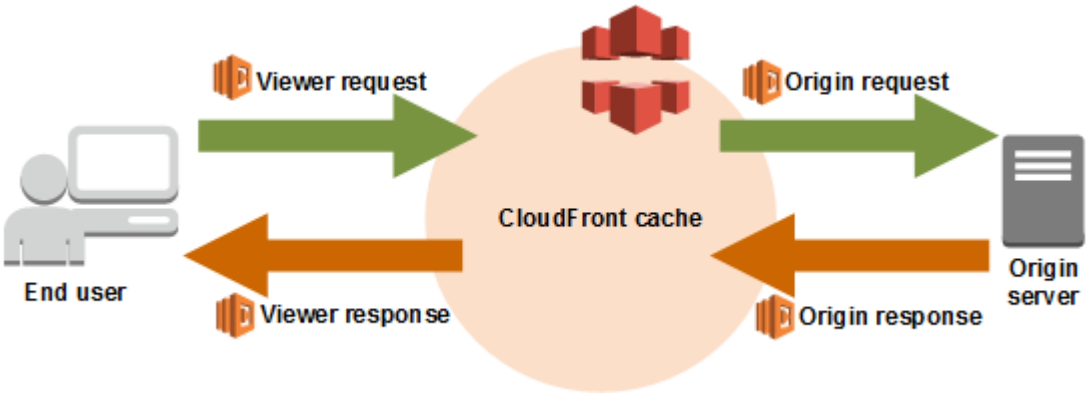
Per ulteriori informazioni sulle risorse personalizzate, consulta Risorse personalizzate nella Guida per l'utente di AWS CloudFormation .

Utilizzo AWS Lambda con CloudFront Lambda @Edge

Lambda @Edge è un'estensione AWS Lambda che consente di distribuire funzioni Python e Node.js nelle edge location di Amazon. CloudFront Un caso d'uso comune di Lambda @Edge consiste nell'utilizzare funzioni per personalizzare il contenuto che la CloudFront distribuzione fornisce agli utenti finali. Il richiamo di queste funzioni più vicine al visualizzatore invece che al server di origine consente di ridurre significativamente la latenza e migliorare l'esperienza dell'utente.

Quando associ una CloudFront distribuzione a una funzione Lambda @Edge, CloudFront intercetta le richieste e le risposte nelle edge location. CloudFront CloudFront quindi richiama la funzione Lambda inviando un evento. Puoi CloudFront invocare la tua funzione Lambda quando si verificano i seguenti eventi:

- Quando CloudFront riceve una richiesta da un visualizzatore (richiesta del visualizzatore)
- Prima CloudFront inoltra una richiesta all'origine (richiesta di origine)
- Quando CloudFront riceve una risposta dall'origine (origin response)
- Before CloudFront restituisce la risposta allo spettatore (risposta del visualizzatore)



> **Note**
>
> Lambda@Edge supporta un insieme limitato di runtime e funzionalità. Per i dettagli, consulta Requisiti e restrizioni sulle funzioni Lambda nella Amazon CloudFront Developer Guide.

Di seguito è riportato un esempio di CloudFront evento.

Example CloudFront evento di messaggio

```
{
  "Records": [
    {
      "cf": {
        "config": {
          "distributionId": "EDFDVBD6EXAMPLE"
        },
        "request": {
          "clientIp": "2001:0db8:85a3:0:0:8a2e:0370:7334",
          "method": "GET",
          "uri": "/picture.jpg",
          "headers": {
            "host": [
              {
                "key": "Host",
                "value": "d111111abcdef8.cloudfront.net"
              }
            ],
            "user-agent": [
              {
                "key": "User-Agent",
                "value": "curl/7.51.0"
              }
            ]
          }
        }
      }
    }
  ]
}
```

Per ulteriori informazioni sull'utilizzo di Lambda @Edge, consulta Using with CloudFront Lambda @Edge.

Uso di AWS Lambda con AWS CodeCommit

È possibile creare un trigger per un repository AWS CodeCommit in modo che gli eventi nel repository invochino una funzione Lambda. È possibile ad esempio invocare una funzione Lambda quando un ramo o un tag viene creato o quando un'istruzione push viene eseguita su un ramo esistente.

Example Evento messaggio di AWS CodeCommit

```
{
    "Records": [
        {
            "awsRegion": "us-east-2",
            "codecommit": {
                "references": [
                    {
                        "commit": "5e493c6f3067653f3d04eca608b4901eb227078",
                        "ref": "refs/heads/master"
                    }
                ]
            },
            "eventId": "31ade2c7-f889-47c5-a937-1cf99e2790e9",
            "eventName": "ReferenceChanges",
            "eventPartNumber": 1,
            "eventSource": "aws:codecommit",
            "eventSourceARN": "arn:aws:codecommit:us-east-2:123456789012:lambda-
pipeline-repo",
            "eventTime": "2019-03-12T20:58:25.400+0000",
            "eventTotalParts": 1,
            "eventTriggerConfigId": "0d17d6a4-efeb-46f3-b3ab-a63741badeb8",
            "eventTriggerName": "index.handler",
            "eventVersion": "1.0",
            "userIdentityARN": "arn:aws:iam::123456789012:user/intern"
        }
    ]
}
```

Per ulteriori informazioni, consulta Gestione dei trigger per un repository AWS CodeCommit.

Utilizzo di AWS Lambda con Amazon Cognito

La funzionalità Amazon Cognito Events consente di eseguire le funzioni Lambda in risposta a eventi in Amazon Cognito. Amazon Cognito fornisce autenticazione, autorizzazione e gestione degli utenti per le app Web e per dispositivi mobili. È possibile richiamare una funzione Lamba in risposta a eventi importanti di Amazon Cognito. Ad esempio, grazie agli eventi di attivazione sincronizzazione è possibile invocare una funzione Lambda che viene pubblicata ogni volta che viene sincronizzato un set di dati. Per ulteriori informazioni e dettagli sull'esempio, consultare la sezione Introduzione di Amazon Cognito Events: sincronizzazione dei trigger nel blog per lo sviluppo di dispositivi mobili.

Example Evento messaggio di Amazon Cognito

```
{
  "datasetName": "datasetName",
  "eventType": "SyncTrigger",
  "region": "us-east-1",
  "identityId": "identityId",
  "datasetRecords": {
    "SampleKey2": {
      "newValue": "newValue2",
      "oldValue": "oldValue2",
      "op": "replace"
    },
    "SampleKey1": {
      "newValue": "newValue1",
      "oldValue": "oldValue1",
      "op": "replace"
    }
  },
  "identityPoolId": "identityPoolId",
  "version": 2
}
```

La mappatura delle origini eventi viene eseguita tramite la configurazione delle sottoscrizioni agli eventi Amazon Cognito. Per informazioni sulla mappatura delle origini evento e per un evento di esempio, consulta Eventi Amazon Cognito nella Guida per gli sviluppatori di Amazon Cognito.

Utilizzo di Lambda con Amazon Connect

È possibile utilizzare una funzione Lambda per elaborare le richieste da Amazon Connect. È possibile utilizzare Amazon Connect per creare un contact center cloud.

Amazon Connect richiama la funzione Lambda in modo sincrono con un evento contenente il corpo della richiesta e i metadati.

Example Evento richiesta Amazon Connect

```
{
  "Details": {
      "ContactData": {
          "Attributes": {},
          "Channel": "VOICE",
          "ContactId": "4a573372-1f28-4e26-b97b-XXXXXXXXXXX",
          "CustomerEndpoint": {
              "Address": "+1234567890",
              "Type": "TELEPHONE_NUMBER"
          },
          "InitialContactId": "4a573372-1f28-4e26-b97b-XXXXXXXXXXX",
          "InitiationMethod": "INBOUND | OUTBOUND | TRANSFER | CALLBACK",
          "InstanceARN": "arn:aws:connect:aws-region:1234567890:instance/
c8c0e68d-2200-4265-82c0-XXXXXXXXXX",
          "PreviousContactId": "4a573372-1f28-4e26-b97b-XXXXXXXXXXX",
          "Queue": {
             "ARN": "arn:aws:connect:eu-west-2:111111111111:instance/cccccccc-bbbb-
dddd-eeee-ffffffffffff/queue/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
             "Name": "PasswordReset"
           },
          "SystemEndpoint": {
              "Address": "+1234567890",
              "Type": "TELEPHONE_NUMBER"
          }
      },
      "Parameters": {
          "sentAttributeKey": "sentAttributeValue"
      }
  },
  "Name": "ContactFlowEvent"
}
```

Per informazioni su come utilizzare Amazon Connect con Lambda, consulta Richiamare funzioni Lambda nella Guida per l'amministratore di Amazon Connect.

Utilizzo AWS Lambda con Amazon EC2

Puoi utilizzarlo AWS Lambda per elaborare 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 completata una snapshot del volume Amazon Elastic Block Store o quando è pianificata la chiusura di un'istanza spot. Si configura EventBridge (CloudWatch Eventi) per inoltrare tali eventi a una funzione Lambda per l'elaborazione.

EventBridge (CloudWatch Events) richiama la funzione Lambda in modo asincrono con il documento 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 dettagli sulla configurazione degli eventi, consulta. Utilizzo di Lambda con Amazon Scheduler EventBridge Per una funzione di esempio che elabora le notifiche di snapshot di Amazon EBS, consulta EventBridge Scheduler for Amazon EBS.

Puoi anche utilizzare l' AWS SDK per gestire istanze e altre risorse con l'API Amazon EC2.

Autorizzazioni

Per elaborare gli eventi del ciclo di vita da Amazon EC2 EventBridge , CloudWatch (Events) necessita dell'autorizzazione per richiamare la tua funzione. Questa autorizzazione proviene dalla

policy basata sulle risorse della funzione. Se utilizzi la console EventBridge (CloudWatch Events) per configurare l'attivazione di un evento, la console aggiorna la politica 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 un'istruzione, 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.

Tutorial: Configurazione di una funzione Lambda per accedere ad Amazon in un ElastiCache Amazon VPC

Per informazioni su come configurare Lambda per accedere ad Amazon ElastiCache in un Amazon VPC, consulta il tutorial Lambda nella Guida per l'utente di Redis. ElastiCache

Elabora 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 instradi 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.

Utilizzo di Amazon EFS con Lambda

Lambda si integra con Amazon Elastic File System (Amazon EFS) per supportare l'accesso sicuro e condiviso al file system per le applicazioni Lambda. È possibile configurare le funzioni per montare un file system durante l'inizializzazione con il protocollo NFS sulla rete locale all'interno di un VPC. Lambda gestisce la connessione e crittografa tutto il traffico da e verso il file system.

Il file system e la funzione Lambda devono trovarsi nella stessa regione. Una funzione Lambda in un account può montare un file system in un account diverso. Per questo scenario, è possibile configurare il peering VPC tra il VPC della funzione e il VPC del file system.

> **Note**
>
> Per configurare una funzione per la connessione a un file system, consulta Configurazione dell'accesso al file system per le funzioni Lambda.

Amazon EFS supporta il blocco dei file per prevenire il danneggiamento se più funzioni tentano di scrivere nello stesso file system contemporaneamente. Il blocco in Amazon EFS segue il protocollo NFS v4.1 per ottenere l'advisory locking e permette alle applicazioni di impiegare sia blocchi di file completi sia blocchi di intervalli di byte.

Amazon EFS offre opzioni per personalizzare il file system in base alla necessità dell'applicazione di mantenere elevate prestazioni su larga scala. Ci sono tre fattori principali da considerare: il numero di connessioni, il throughput (in MiB al secondo) e IOPS.

> **Quote**
>
> Per ulteriori informazioni sulle quote e sui limiti del file system, consulta l'argomento relativo alle quote per i file system di Amazon EFS nell'Amazon Elastic File System User Guide.

Per evitare problemi di scalabilità, throughput e IOPS, monitora i parametri che Amazon EFS invia ad Amazon. CloudWatch Per una panoramica del monitoraggio in Amazon EFS, consulta Monitoraggio Amazon EFS nell'Amazon Elastic File System User Guide.

Sections

- IOPS

Connessioni

Amazon EFS supporta fino a 25.000 connessioni per file system. Durante l'inizializzazione, ogni istanza di una funzione crea una singola connessione al proprio file system che persiste tra le chiamate. Ciò significa che è possibile raggiungere la simultaneità 25.000 attraverso una o più funzioni collegate a un file system. Per limitare il numero di connessioni create da una funzione, utilizzare la simultaneità riservata.

Tuttavia, quando si apportano modifiche al codice o alla configurazione della funzione scalabile, si verifica un aumento temporaneo del numero di istanze della funzione oltre la simultaneità corrente. Lambda effettua il provisioning di nuove istanze per gestire le nuove richieste e si verifica qualche ritardo prima che le vecchie istanze chiudano le loro connessioni al file system. Per evitare di raggiungere il limite massimo di connessioni durante una distribuzione, utilizzare le distribuzioni in sequenza. Con le distribuzioni in sequenza, si sposta gradualmente il traffico alla nuova versione ogni volta che si apporta una modifica.

Se ci si connette allo stesso file system da altri servizi, ad esempio Amazon EC2, è necessario essere consapevoli del comportamento di dimensionamento delle connessioni in Amazon EFS. Un file system supporta la creazione di un massimo di 3.000 connessioni in un burst, dopodiché supporta 500 nuove connessioni al minuto.

Per monitorare e attivare un allarme sulle connessioni, utilizzare il parametro `ClientConnections`.

Prestazioni

Su scala, è anche possibile superare il throughput massimo per un file system. In modalità Bursting (impostazione predefinita), un file system ha un throughput di base ridotto che viene dimensionato in modo lineare in base alle dimensioni. Per consentire picchi di attività, al file system vengono concessi crediti di burst che consentono di utilizzare 100 MIb/s o più di throughput. I crediti si accumulano continuamente e vengono spesi con ogni operazione di lettura e scrittura. Se il file system esaurisce i crediti, limita le operazioni di lettura e scrittura oltre il throughput di base, il che può causare il timeout delle chiamate.

> **Note**
>
> Se si utilizza la simultaneità fornita, la funzione può consumare crediti di burst anche quando è inattiva. Con la simultaneità fornita, Lambda inizializza le istanze della funzione prima che

venga richiamata e ricicla le istanze ogni poche ore. Se durante l'inizializzazione si utilizzano file in un file system collegato, questa attività può utilizzare tutti i crediti di burst.

Per monitorare e attivare un allarme sul throughput, utilizzare il parametro `BurstCreditBalance`. Dovrebbe aumentare quando la simultaneità della funzione è bassa e diminuire quando è alta. Se diminuisce sempre o non accumula abbastanza durante l'attività meno intensa per coprire il picco di traffico, potrebbe essere necessario limitare la simultaneità della funzione o abilitare il throughput assegnato.

IOPS

Operazioni di input/output al secondo (IOPS) è una misurazione del numero di operazioni di lettura e scrittura elaborate dal file system. In modalità per uso generico, IOPS è limitato a favore di una latenza più bassa, il che è vantaggioso per la maggior parte delle applicazioni.

Per il monitoraggio e per gli allarmi su IOPS in modalità generica, utilizzare il parametro `PercentIOLimit`. Se questa parametro raggiunge il 100%, la funzione può andare in timeout in attesa del completamento delle operazioni di lettura e scrittura.

Utilizzo di Lambda con Amazon Scheduler EventBridge

Amazon EventBridge Scheduler è uno strumento di pianificazione senza server che consente di creare, eseguire e gestire attività da un unico servizio gestito centralizzato. Con EventBridge Scheduler, puoi creare pianificazioni utilizzando le espressioni cron e rate per schemi ricorrenti o configurare chiamate 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 EventBridge Scheduler con Lambda EventBridge , Scheduler richiama la funzione Lambda in modo asincrono. Questa pagina spiega come usare EventBridge Scheduler per richiamare una funzione Lambda in base a una pianificazione.

Configurare il ruolo di esecuzione

Quando crei una nuova EventBridge pianificazione, Scheduler deve avere l'autorizzazione a richiamare l'operazione 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 si desidera che EventBridge Scheduler richiami.

Quando si utilizza la console EventBridge Scheduler per creare una pianificazione, come nella procedura seguente, EventBridge Scheduler imposta automaticamente un ruolo di esecuzione in base all'obiettivo selezionato. Se si desidera creare una pianificazione utilizzando uno degli EventBridge Scheduler SDK AWS CLI AWS CloudFormation, oppure è necessario disporre di un ruolo di esecuzione esistente che conceda le autorizzazioni richieste da EventBridge Scheduler per richiamare una destinazione. Per ulteriori informazioni sull'impostazione manuale di un ruolo di esecuzione per la pianificazione, consulta Configurazione di un ruolo di esecuzione nella Guida per l'utente di Scheduler. EventBridge

Creare una pianificazione.

Per creare una pianificazione utilizzando la console

1. Apri la console Amazon EventBridge Scheduler all'indirizzo https://console.aws.amazon.com/scheduler/home.
2. Nella pagina Pianificazioni, scegli Crea pianificazione.

3. Nella pagina Specifica i dettagli della pianificazione, nella sezione Nome e descrizione della pianificazione, effettua le seguenti operazioni:

   a. Per Nome pianificazione, inserisci un nome per la pianificazione. Ad esempio, **`MyTestSchedule`**.

   b. (Facoltativo) Per Descrizione, inserisci una descrizione per la pianificazione. Ad esempio, **`My first schedule`**.

   c. Per Gruppo di pianificazioni, scegli un gruppo di pianificazioni dall'elenco a discesa. Se non hai un gruppo, scegli predefinito. Per creare un gruppo di pianificazioni, scegli crea la tua pianificazione.

      I gruppi di pianificazione vengono utilizzati per aggiungere tag a gruppi di pianificazioni.

4. • Scegli le opzioni di pianificazione.

Ricorrenza	Esegui questa operazione…	
Pianificazione una tantum Una pianificazione unica richiama una destinazione solo una volta alla data e all'ora specificate.	Per Data e ora, effettua le seguenti operazioni: • Inserisci una data valida in formato `YYYY/MM/DD`. • Inserisci un timestamp in formato `hh:mm` 24 ore. • Per Fuso orario, scegli il fuso orario.	
Pianificazione ricorrente Una pianificazione ricorrente richiama una destinazione con una frequenza specificata utilizzando un'espressione cron o un'espressione rate.	a. Per Tipo di pianificazione, esegui una delle seguenti operazioni: • Per utilizzare un'espressione Cron per definire la pianificazione, scegli Pianificazione basata su cron e	

Ricorrenza	Esegui questa operazione…	
	immetti l'espressione Cron. • Per utilizzare un'espressione di frequenza per definire la pianificazione, scegli Pianificazione basata su frequenza e inserisci l'espressione di frequenza. Per ulteriori informazioni sulle espressioni cron e rate, consulta Schedule types on EventBridge Scheduler nella Amazon EventBridge Scheduler User Guide. b. Per Finestra temporale flessibile, scegli Disattivata per disattivare l'opzione o scegli una delle finestre temporali predefinite. Ad esempio, se scegli 15 minuti e imposti una pianificazione ricorrente per il richiamo della destinazione ogni ora, la pianificazione viene eseguita entro 15 minuti dall'inizio di ogni ora.	

5. (Facoltativo) Se hai scelto Pianificazione ricorrente nel passaggio precedente, nella sezione Intervallo di tempo effettua le seguenti operazioni:

   a. Per Fuso orario, scegli un fuso orario.

   b. Per Data e ora di inizio, inserisci una data valida in formato `YYYY/MM/DD`, quindi specifica un timestamp in formato `hh:mm` 24 ore.

   c. Per Data e ora di fine, inserisci una data valida in formato `YYYY/MM/DD`, quindi specifica un timestamp in formato `hh:mm` 24 ore.

6. Seleziona Avanti.

7. Nella pagina Seleziona destinazione, scegli l'operazione AWS API richiamata da Scheduler: EventBridge

   a. Scegli Richiama AWS Lambda .

   b. Nella sezione Richiama, seleziona una funzione o scegli Crea una nuova funzione Lambda.

   c. (Facoltativo) Inserisci un payload JSON. Se non inserisci un payload, EventBridge Scheduler utilizza un evento vuoto per richiamare la funzione.

8. Seleziona Avanti.

9. Nella pagina Settings (Impostazioni), eseguire le operazioni descritte di seguito.

   a. Per attivare la pianificazione, in Stato della pianificazione, attiva Abilita pianificazione.

   b. Per configurare una policy di ripetizione per la tua pianificazione, in Policy di ripetizione e coda DLQ (Dead-Letter Queue) effettua le seguenti operazioni:

      - Attiva/disattiva Riprova.
      - Per Età massima dell'evento, inserisci il numero massimo di ore e minuti in cui EventBridge Scheduler deve conservare un evento non elaborato.
      - La durata massima è 24 ore.
      - Per Numero massimo di tentativi, inserisci il numero massimo di volte in cui EventBridge Scheduler riprova la pianificazione se la destinazione restituisce un errore.

        Il valore massimo è 185 tentativi.

      Con le politiche di ripetizione dei tentativi, se una pianificazione non riesce a richiamare l'obiettivo, EventBridge Scheduler esegue nuovamente la pianificazione. Se configurato,

è necessario impostare il tempo di conservazione massimo e i nuovi tentativi per la pianificazione.

c. Scegli dove EventBridge Scheduler archivia gli eventi non consegnati.

Opzione Dead-letter queue (DLQ)	Esegui questa operazione…	
Non conservare	Scegliere None (Nessuno).	
Archivia l'evento nello stesso spazio in Account AWS cui stai creando il programma	a. Scegli Seleziona una coda Amazon SQS in my Account AWS as a DLQ. b. Scegli il nome della risorsa Amazon (ARN) della coda di Amazon SQS.	
Archivia l'evento in un luogo diverso Account AWS da quello in cui stai creando il programma	a. Scegli Specificare una coda Amazon SQS tra le altre Account AWS come DLQ. b. Inserisci il nome della risorsa Amazon (ARN) della coda di Amazon SQS.	

d. Per utilizzare una chiave gestita dal cliente per crittografare l'input di destinazione, in Crittografia scegli Personalizza le impostazioni di crittografia (avanzate).

   Se scegli questa opzione, inserisci l'ARN di una chiave KMS esistente scegli Crea una AWS KMS key per accedere alla console AWS KMS . Per ulteriori informazioni su come EventBridge Scheduler crittografa i dati inattivi, consulta Encryption at rest nella Amazon EventBridge Scheduler User Guide.

e. 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 assegna al ruolo le autorizzazioni necessarie per la destinazione basata sul modello.

10. Seleziona Avanti.
11. Nella pagina Rivedi e crea pianificazione, rivedi i dettagli della pianificazione. In ogni sezione, scegli Modifica per tornare a tale passaggio e modificarne i dettagli.
12. Scegli Crea pianificazione.

    Puoi visualizzare un elenco delle pianificazioni nuove ed esistenti nella pagina Pianificazioni. Nella colonna Stato, accertati che la nuova pianificazione sia Abilitata.

Per confermare che EventBridge Scheduler ha richiamato la funzione, controlla i log Amazon CloudWatch della funzione.

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

Uso di AWS Lambda con AWS IoT

AWS IoT fornisce una comunicazione sicura tra i dispositivi connessi a Internet (come i sensori) e AWS Cloud. Questo consente raccogliere, archiviare e analizzare i dati di telemetria da più dispositivi.

È possibile creare regole AWS IoT affinché i dispositivi interagiscano con i servizi AWS. AWS IoT Rules Engine fornisce un linguaggio basato su SQL, per selezionare i dati dai payload dei messaggi e inviare i dati ad altri servizi, ad esempio Amazon S3, Amazon DynamoDB e AWS Lambda. Si definisce una regola per richiamare una funzione Lambda quando si desidera richiamare un altro servizio AWS o un servizio di terze parti.

Quando un messaggio IoT in ingresso attiva la regola, AWS IoTrichiama 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 Evento messaggio di AWS IoT

```
{
    "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 servizio AWS IoT per richiamare la funzione Lambda. Utilizza il comando `add-permission` per aggiungere un'istruzione di autorizzazione alla policy basata sulle risorse della funzione.

```
aws lambda add-permission --function-name my-function \
--statement-id iot-events --action "lambda:InvokeFunction" --principal
 iot.amazonaws.com
```

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\"}"
}
```

Per ulteriori informazioni su come utilizzare Lambda con AWS IoT, consulta Creazione di una regola AWS Lambda.

Utilizzo AWS Lambda con Amazon Data Firehose

Amazon Data Firehose acquisisce, trasforma e carica i dati in streaming in servizi downstream come Managed Service for Apache Flink o Amazon S3. È possibile scrivere funzioni Lambda per richiedere un'elaborazione aggiuntiva e personalizzata dei dati prima che vengano inviati a valle.

Example Evento messaggio Amazon Data Firehose


```
{
  "invocationId": "invoked123",
  "deliveryStreamArn": "aws:lambda:events",
  "region": "us-west-2",
  "records": [
    {
      "data": "SGVsbG8gV29ybGQ=",
      "recordId": "record1",
      "approximateArrivalTimestamp": 1510772160000,
      "kinesisRecordMetadata": {
        "shardId": "shardId-000000000000",
        "partitionKey": "4d1ad2b9-24f8-4b9d-a088-76e9947c317a",
        "approximateArrivalTimestamp": "2012-04-23T18:25:43.511Z",
        "sequenceNumber": "49546986683135544286507457936321625675700192471156785154",
        "subsequenceNumber": ""
      }
    },
    {
      "data": "SGVsbG8gV29ybGQ=",
      "recordId": "record2",
      "approximateArrivalTimestamp": 151077216000,
      "kinesisRecordMetadata": {
        "shardId": "shardId-000000000001",
        "partitionKey": "4d1ad2b9-24f8-4b9d-a088-76e9947c318a",
        "approximateArrivalTimestamp": "2012-04-23T19:25:43.511Z",
        "sequenceNumber": "49546986683135544286507457936321625675700192471156785155",
        "subsequenceNumber": ""
      }
    }
  ]
}
```


Per ulteriori informazioni, consulta Amazon Data Firehose data transformation nella Firehose Developer Guide.

Uso di AWS Lambda con Amazon Lex

Puoi utilizzare Amazon Lex per integrare un bot conversazionale nell'applicazione. Il bot Amazon Lex fornisce un'interfaccia di conversazione con i tuoi utenti. Amazon Lex fornisce l'integrazione precostruita con Lambda, che consente di utilizzare una funzione Lambda con il bot Amazon Lex.

Quando configuri un bot Amazon Lex, puoi specificare una funzione Lambda per eseguire la convalida, la gestione o entrambi. Per la convalida, Amazon Lex richiama la funzione Lambda dopo ogni risposta da parte dell'utente. La funzione Lambda può convalidare la risposta e fornire un feedback correttivo all'utente, se necessario. Per il soddisfacimento, Amazon Lex richiama la funzione Lambda per soddisfare la richiesta dell'utente dopo che il bot ha raccolto correttamente tutte le informazioni richieste e ha ricevuto la conferma da parte dell'utente.

Puoi gestire la concorrenza della tua funzione Lambda per controllare il numero massimo di bot conversazionali simultanei che servi. L'API Amazon Lex restituisce un codice di stato HTTP 429 (Too Many Requests) se la funzione ha raggiunto il livello di concorrenza massima.

L'API restituisce un codice di stato HTTP 424 (Dependency Failed Exception) se la funzione Lambda genera un'eccezione.

Il bot Amazon Lex invoca la funzione Lambda in modo sincrono. Il parametro evento contiene informazioni sul bot e sul valore di ogni slot nel dialogo. Per le definizioni dei campi evento e risposta, consulta Formato eventi e risposte di Lambda nella Guida per gli sviluppatori di Amazon Lex. Il `invocationSource` parametro nell'evento del messaggio Amazon Lex indica se la funzione Lambda deve convalidare gli input (DialogCodeHook) o soddisfare l'intento (). FulfillmentCodeHook

Per un'esercitazione di esempio che mostra come utilizzare Lambda con Amazon Lex, consulta Esercizio 1: creazione di un bot Amazon Lex utilizzando un piano nella Guida per gli sviluppatori di Amazon Lex.

Ruoli e autorizzazioni

È necessario configurare un ruolo collegato ai servizi come ruolo di esecuzione della funzione. Amazon Lex definisce il ruolo collegato ai servizi con autorizzazioni predefinite. Quando crei un bot Amazon Lex utilizzando la console, il ruolo collegato al servizio viene creato automaticamente. Per creare un ruolo collegato al servizio con l'interfaccia della riga di comando AWS CLI, utilizza il comando `create-service-linked-role`.

```
aws iam create-service-linked-role --aws-service-name lex.amazonaws.com
```

Questo comando crea il ruolo seguente.

```
{
  "Role": {
      "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
              {
                  "Action": "sts:AssumeRole",
                  "Effect": "Allow",
                  "Principal": {
                      "Service": "lex.amazonaws.com"
                  }
              }
          ]
      },
      "RoleName": "AWSServiceRoleForLexBots",
      "Path": "/aws-service-role/lex.amazonaws.com/",
      "Arn": "arn:aws:iam::account-id:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
}
```

Se la funzione Lambda utilizza altri servizi AWS, è necessario aggiungere le autorizzazioni corrispondenti al ruolo collegato al servizio.

Per consentire al bot Amazon Lex di richiamare la funzione Lambda, viene utilizzata una policy di autorizzazione basata sulle risorse. Se utilizzi la console Amazon Lex, la policy di autorizzazione viene creata automaticamente. In AWS CLI, utilizza il comando `add-permission` Lambda per impostare l'autorizzazione.

Per Amazon Lex V2, emetti il comando seguente. Nell'ARN di origine, sostituisci `us-east-1` con la Regione AWS in cui si trova il bot Amazon Lex e usa il tuo numero di Account AWS e alias bot.

```
aws lambda add-permission \
    --function-name LexCodeHook \
    --statement-id LexInvoke-MyBot \
    --action lambda:InvokeFunction \
    --principal lex.amazonaws.com \
    --source-arn "arn:aws:lex:us-east-1:123456789012:bot-alias/MYBOT/MYBOTALIAS"
```

Puoi utilizzare Amazon Lex V1 anche per richiamare una funzione Lambda. Per Amazon Lex V1, emetti il comando seguente. Nell'ARN di origine, sostituisci `us-east-1` con la Regione AWS in cui desideri si trovi Amazon Lex e usa il tuo numero di Account AWS e alias bot.

```
aws lambda add-permission \
    --function-name LexCodeHook \
    --statement-id LexInvoke-MyIntent \
    --action lambda:InvokeFunction \
    --principal lex.amazonaws.com \
    --source-arn "arn:aws:lex:us-east-1:123456789012 ID:intent:MYINTENT:*"
```

Tieni presente che Amazon Lex V1 non è più sottoposto a manutenzione. Ti consigliamo di utilizzare Amazon Lex V2.

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.

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.

Configurazione di una funzione

Nella console Lambda, puoi effettuare il provisioning e configurare istanze di database Amazon RDS e risorse proxy. Per ulteriori informazioni, consulta i database RDS nella scheda Configurazione. In alternativa, puoi anche creare e configurare connessioni alle funzioni Lambda nella console Amazon RDS.

- 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 responsabile IAM che utilizza la console per configurare la risorsa Amazon RDS e connetterla alla tua funzione deve disporre delle seguenti autorizzazioni:

> **Note**
>
> Le autorizzazioni del proxy Amazon RDS sono necessarie solo se configuri un proxy Amazon RDS per gestire un pool di connessioni al database.

Example policy di autorizzazione

```
{
  "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:AttachPolicy",
        "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.

> **Impostazione Lambda e Amazon RDS**
>
> Le console Lambda e Amazon RDS ti aiuteranno a configurare in automatico alcune risorse necessarie a stabilire una connessione tra Lambda e Amazon RDS.

Connect a un database Amazon RDS in una funzione Lambda

Il seguente esempio di codice mostra 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.

Go

SDK per Go V2

> **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 Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Golang v2 code here.
*/

package main

import (
 "context"
 "database/sql"
 "encoding/json"
 "fmt"

 "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 = "DatabaseName"
 var dbUser string = "DatabaseUser"
 var dbHost string = "mysqldb.123456789012.us-east-1.rds.amazonaws.com"
 var dbPort int = 3306
 var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
 var region string = "us-east-1"

 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)
}
```

JavaScript

SDK per (v2 JavaScript)

> **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 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)
  }
};
```

Elaborare 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&amp;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"
            }
          }
        ]
```

```
    }
```

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 funzione Lambda leggerà i record da una coda di Amazon SQS e scriverà nuove voci in una tabella del database ogni volta che verrà aggiunto un messaggio.

Elabora le notifiche degli eventi di 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":
 "vlR7PnpV2Ce81l0PRw6jlUpck7Jo5ZsQjryTjKlc5aLWGVHPZLj5NeC6qMa0emYBDXOo6QBU0Wo="
```

```
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "828aa6fc-f7b5-4305-8584-487c791949c1",
        "bucket": {
          "name": "DOC-EXAMPLE-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.

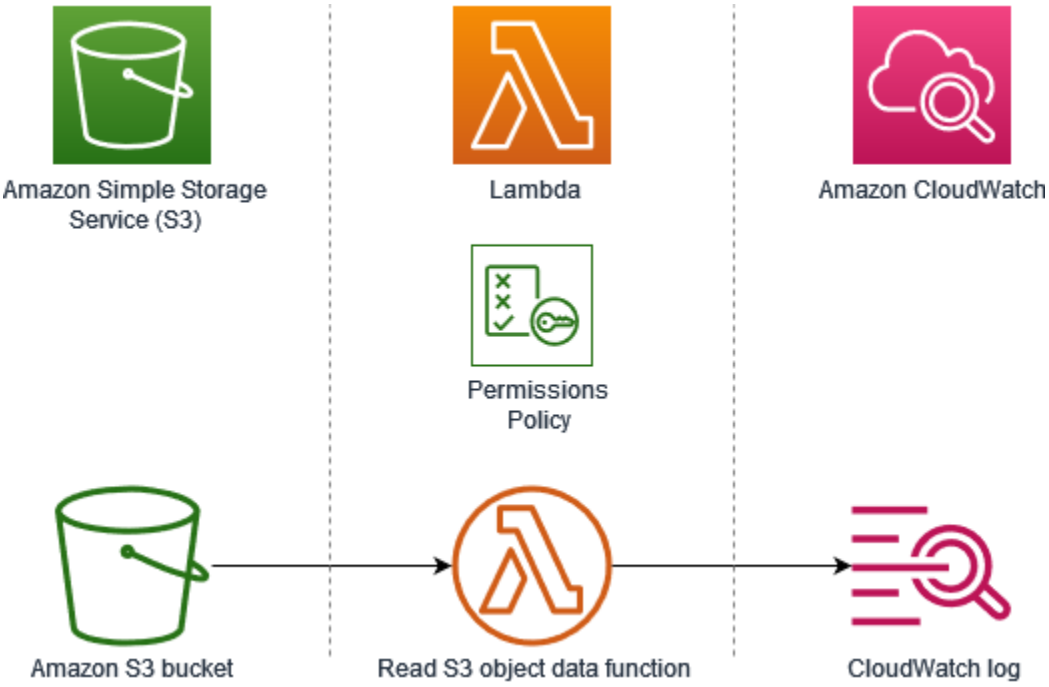
Se la tua funzione utilizza l' AWS SDK per gestire le risorse Amazon S3, necessita anche delle autorizzazioni Amazon S3 nel suo ruolo di esecuzione.

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 in 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 AWS Management Console.

Prerequisiti

Registrati per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come procedura consigliata in materia di sicurezza, assegnate l'accesso amministrativo a un utente e utilizzate solo l'utente root per eseguire attività che richiedono l'accesso da parte dell'utente root.

AWS ti invia un'e-mail di conferma dopo il completamento della procedura di registrazione. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .

2. In IAM Identity Center, concedi l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con le impostazioni predefinite IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accedi come utente con accesso amministrativo

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso ad altri utenti

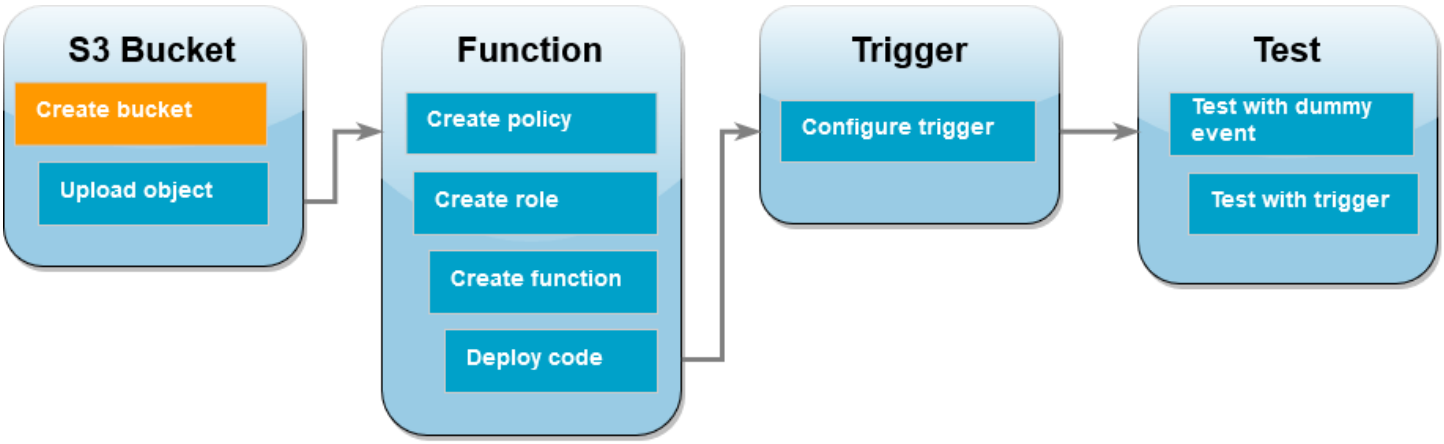
1. In IAM Identity Center, crea un set di autorizzazioni che segua la migliore pratica di applicazione delle autorizzazioni con privilegi minimi.

   Per istruzioni, consulta Creare un set di autorizzazioni nella Guida per l'utente.AWS IAM Identity Center

2. Assegna gli utenti a un gruppo, quindi assegna l'accesso Single Sign-On al gruppo.

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente.AWS IAM Identity Center

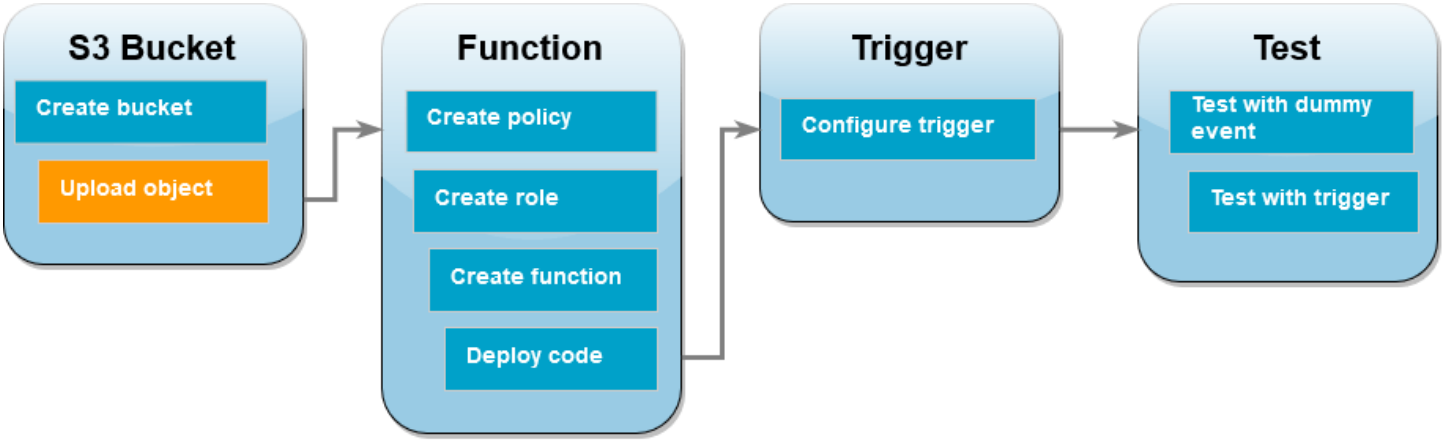
Creazione di un bucket Amazon S3



Come creare un bucket Amazon S3.

1. Apri la console Amazon S3 e seleziona la pagina Bucket.
2. Scegliere Create bucket (Crea bucket).
3. In General configuration (Configurazione generale), eseguire le operazioni seguenti:
   a. 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 (-).
   b. In AWS Region (Regione), scegliere una Regione. Più avanti nel tutorial, è necessario creare la funzione Lambda nella stessa regione.
4. Lascia tutte le altre opzioni impostate sui valori predefiniti e scegli Crea bucket.

Caricamento di un oggetto di test in un bucket

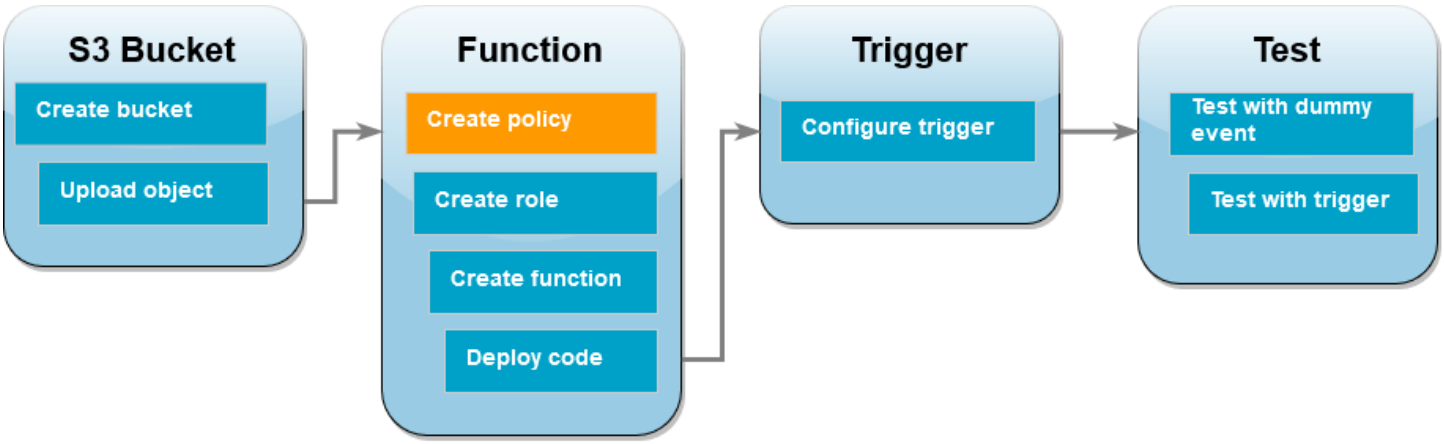


Caricamento di un oggetto di test

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 che desideri caricare. Puoi selezionare qualsiasi file (ad esempio,`HappyFace.jpg`).
4. Seleziona Apri, quindi Carica.

Più avanti nel tutorial, testerai la tua funzione Lambda utilizzando questo oggetto.

Creazione di una policy di autorizzazione



Crea una politica di autorizzazioni che consenta a Lambda di ottenere oggetti da un bucket Amazon S3 e di scriverli su Amazon Logs. CloudWatch

Come creare la policy

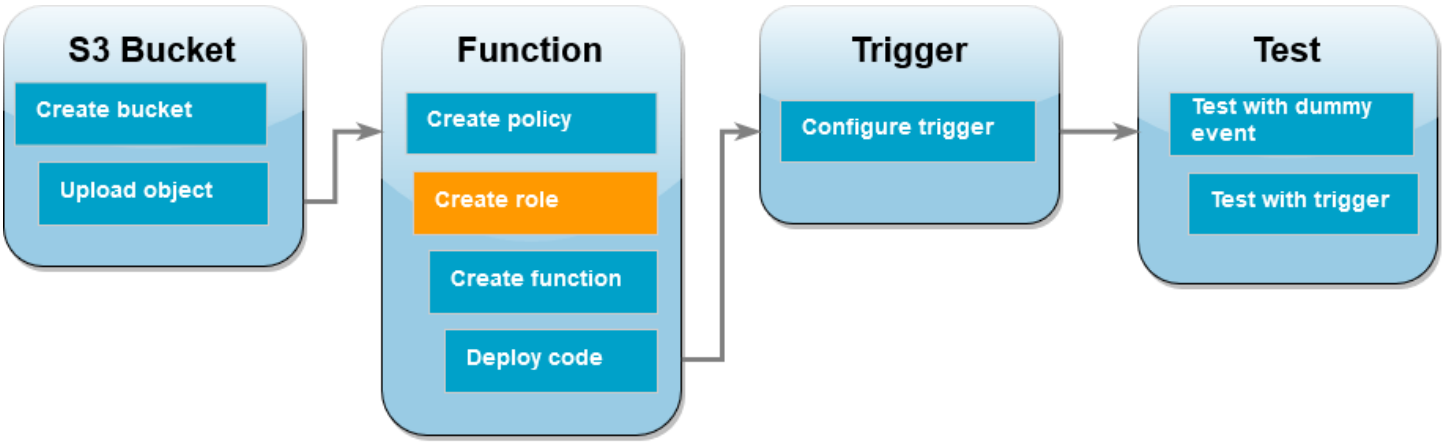
1. Apri la pagina Policies (Policy) nella console IAM.
2. Scegliere Create Policy (Crea policy).
3. Scegliere la scheda JSON e quindi incollare la seguente policy personalizzata nell'editor JSON.

```
{
    "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:::*/*"
        }
    ]
}
```

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



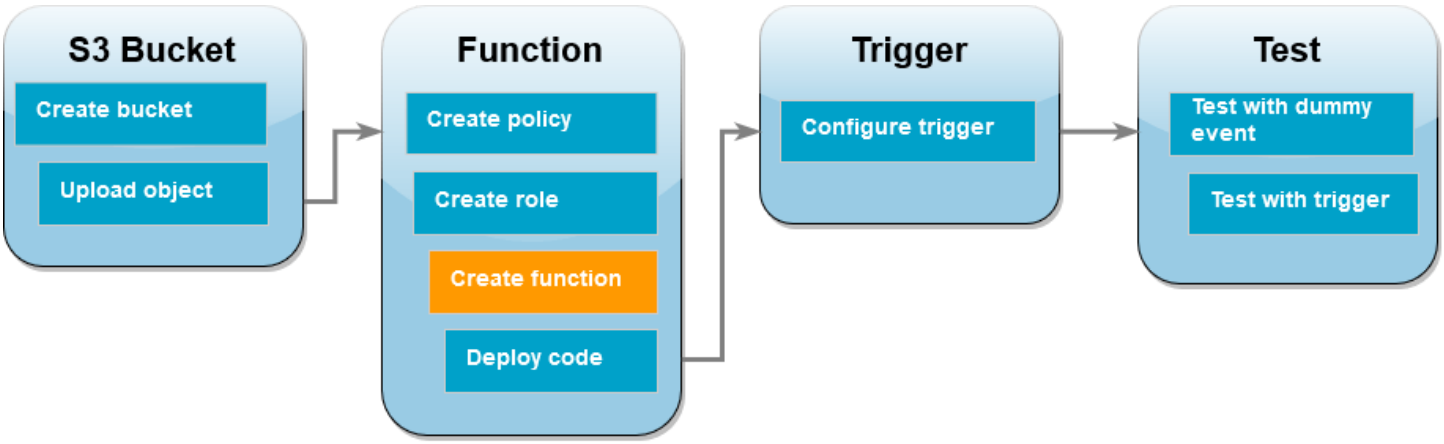
Un ruolo di esecuzione è un ruolo AWS Identity and Access Management (IAM) che concede a una funzione Lambda l'autorizzazione ad AWS accedere a servizi e risorse. In questo passaggio, crea un ruolo di esecuzione utilizzando la politica di autorizzazioni creata nel passaggio precedente.

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 Create role (Crea ruolo).
3. Per il tipo di entità attendibile, scegli Servizio AWS , quindi per il caso d'uso seleziona Lambda.
4. Seleziona Successivo.
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).

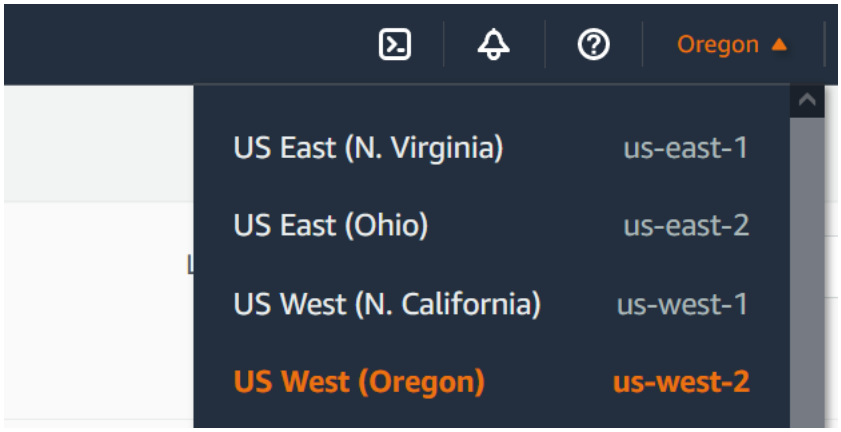
Creazione della funzione Lambda



Crea una funzione Lambda nella console usando il runtime Python 3.12.

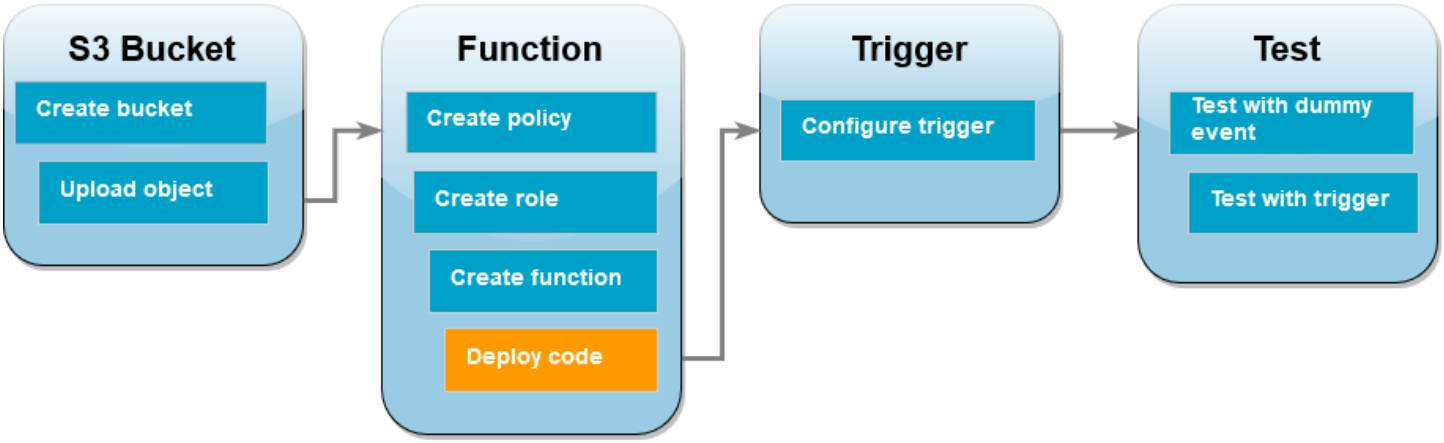
Creazione della funzione Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Assicurati di lavorare nello stesso bucket in Regione AWS cui hai creato il tuo 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 3.12.
   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 di Python 3.12, ma abbiamo anche fornito file di codice di esempio per altri runtime. Per visualizzare il codice per il runtime che ti interessa, seleziona la scheda corrispondente nella casella seguente.

La funzione Lambda recupera il nome chiave dell'oggetto caricato e il nome del bucket dal `event` parametro che riceve da Amazon S3. La funzione utilizza quindi il metodo get_object di AWS SDK for 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

   AWS SDK for .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.DefaultLambdaJson

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 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.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

exports.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.

Consumo di un evento S3 con Lambda utilizzando 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\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 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 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.

Consumo di un evento S3 con Lambda utilizzando 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 da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

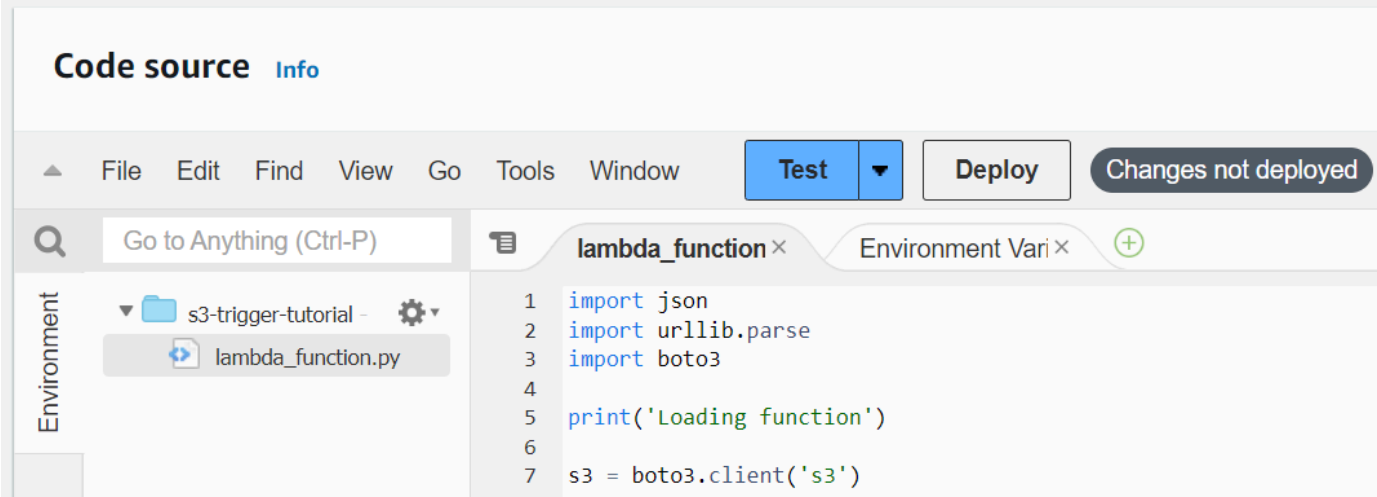
Utilizzo di un evento S3 con Lambda tramite Rust.

```
// 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;
```

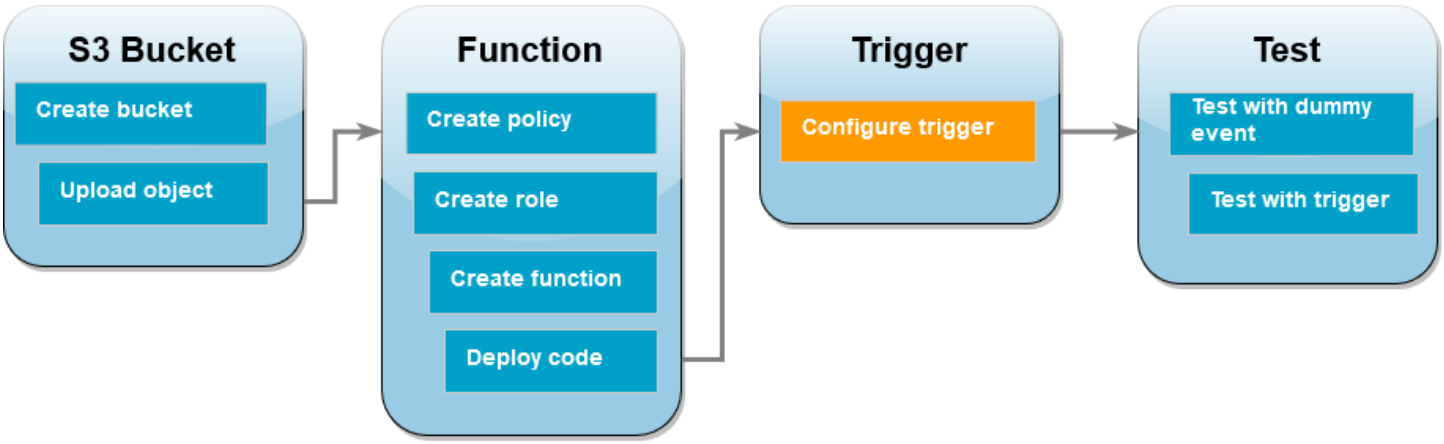
```
    match s3_get_object_result {
        Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
 contains a 'body' property of type ByteStream"),
        Err(_) => tracing::info!("Failure with S3 Get Object request")
    }

    Ok(())
}
```

2. Nel riquadro Codice sorgente della console Lambda, incolla il codice nel file lambda_function.py.



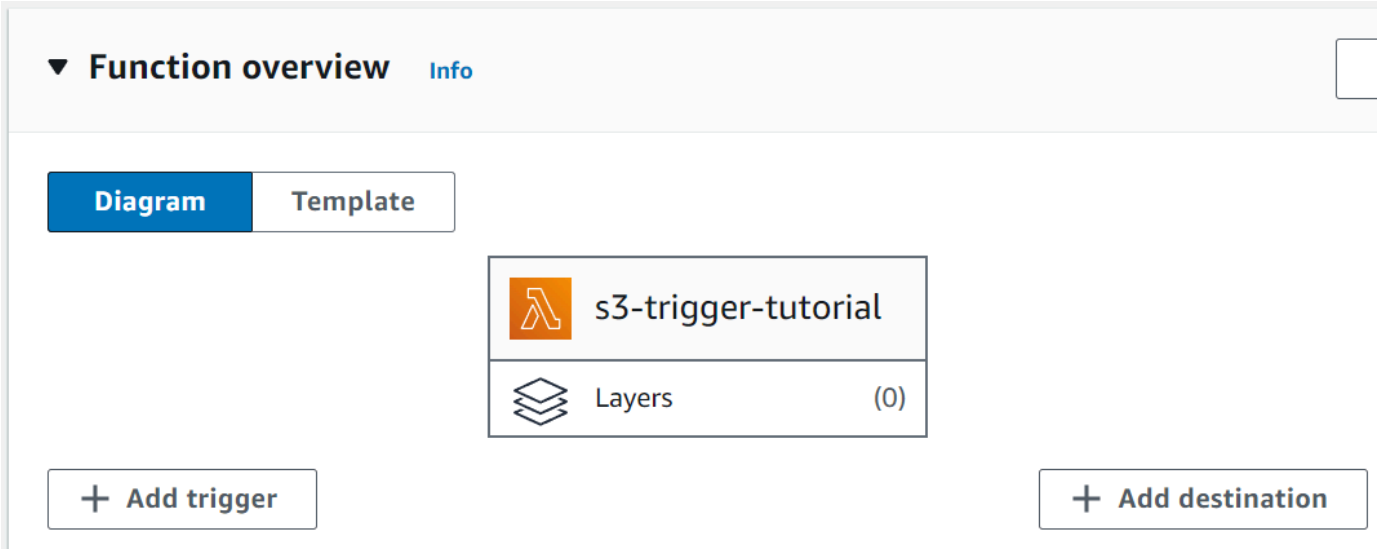
3. Seleziona Deploy (Implementa).

Creazione del trigger Amazon S3



Creazione del trigger Amazon S3

1. Nel riquadro Panoramica della funzione, scegli Aggiungi trigger.

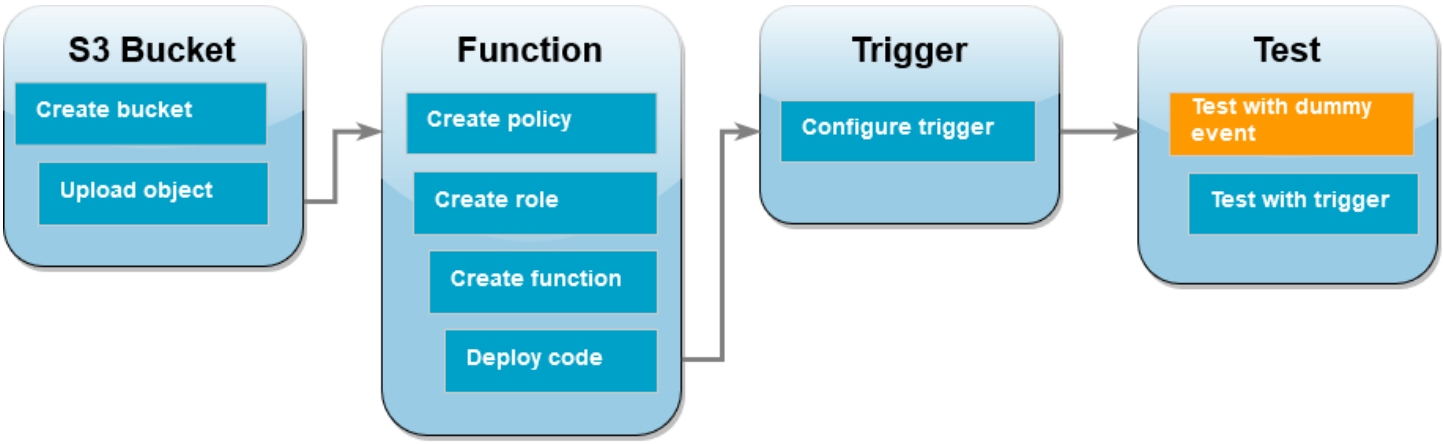


2. Seleziona S3.
3. In Bucket, seleziona il bucket che hai creato in precedenza nel tutorial.
4. In Tipi di evento, assicurati che sia selezionata l'opzione Tutti gli eventi di creazione degli oggetti.
5. In Invocazione ricorsiva, seleziona la casella di controllo per confermare che non è consigliabile utilizzare lo stesso bucket Amazon S3 per input e output.
6. Scegli 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 politiche 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 politiche 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.
> ```

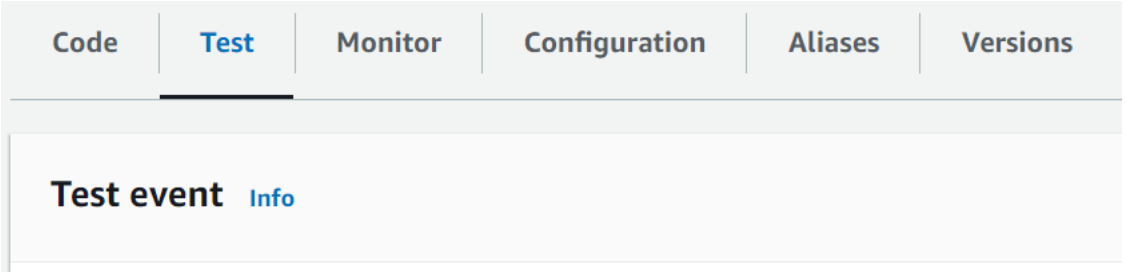
Puoi visualizzare questo errore se in precedenza hai configurato un trigger per un'altra funzione Lambda utilizzando lo stesso bucket e da allora hai eliminato la funzione o modificato le sue politiche di autorizzazione.

Test di una funzione Lambda con un evento fittizio



Test della funzione Lambda con un evento fittizio

1. Nella pagina della console Lambda relativa alla tua funzione, scegli la scheda Test.



2. Per Event name (Nome evento) immettere `MyTestEvent`.
3. Nella sezione Event JSON, incolla il seguente evento di test. Assicurati di sostituire questi valori:

   - Sostituisci `us-east-1` con la regione in cui è stato creato il bucket Amazon S3.
   - Sostituisci entrambe le istanze di `DOC-EXAMPLE-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`).

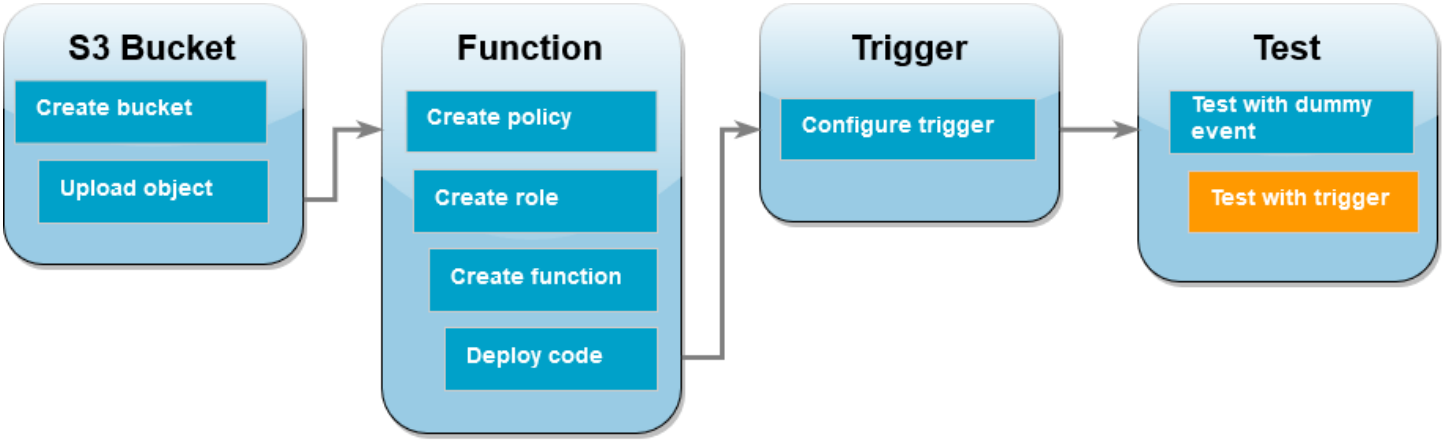
```
{
  "Records": [
```

```
    {
      "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": "DOC-EXAMPLE-BUCKET",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
        },
        "object": {
          "key": "test%2Fkey",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
  ]
}
```

4. Scegli Save (Salva).
5. Scegli Test (Esegui test).
6. Se la tua funzione viene eseguita correttamente, vedrai un output simile al seguente nella scheda Risultati dell'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: 'DOC-EXAMPLE-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 testare la tua funzione con il trigger configurato, carica un oggetto nel tuo bucket Amazon S3 utilizzando la console. Per verificare che la funzione Lambda sia stata eseguita come previsto, usa CloudWatch Logs per visualizzare l'output della funzione.

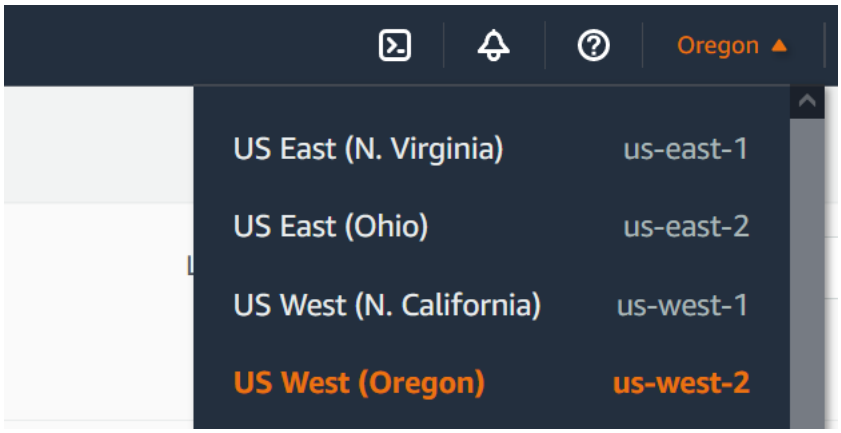
Caricamento di un oggetto nel bucket Amazon S3

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 nella stessa modalità in Regione AWS cui hai creato la 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 output 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 addebiti inutili ai tuoi. 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 **`delete`** 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 Delete (Elimina).
4. Inserisci il nome del bucket nel campo di immissione testo.
5. Scegli Delete Bucket (Elimina bucket).

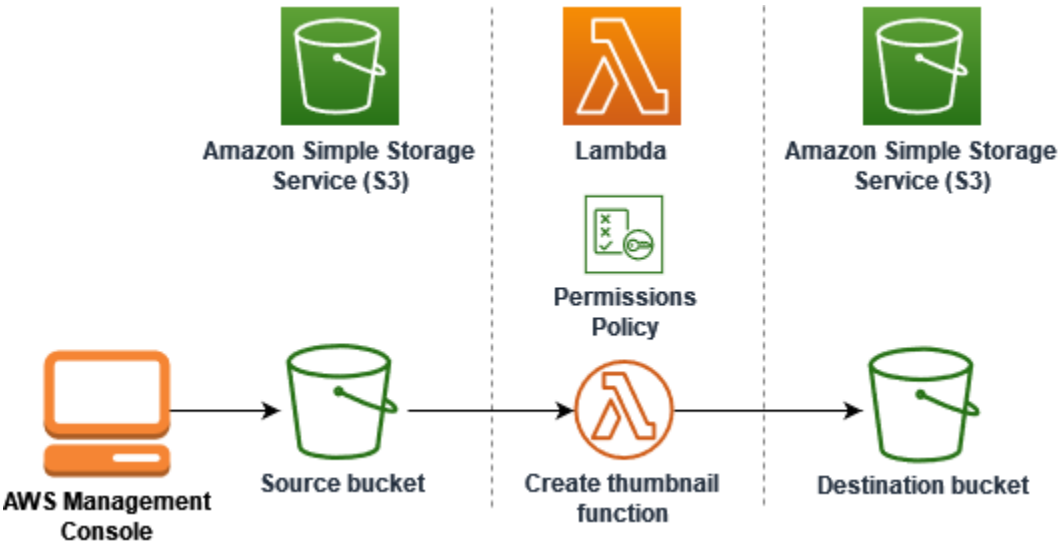
Passaggi successivi

InTutorial: uso di un trigger Amazon S3 per creare immagini in miniatura, il trigger di Amazon S3 richiama una funzione che crea un'immagine in miniatura per ogni file di immagine caricato in un 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 di archivi 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 usando il AWS Command Line Interface (AWS CLI) o il AWS Management Console.

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

Iscriviti per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come procedura consigliata in materia di sicurezza, assegnate l'accesso amministrativo a un utente e utilizzate solo l'utente root per eseguire attività che richiedono l'accesso da parte dell'utente root.

AWS ti invia un'e-mail di conferma dopo il completamento della procedura di registrazione. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .
2. In IAM Identity Center, concedi l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con le impostazioni predefinite IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accedi come utente con accesso amministrativo

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso ad altri utenti

1. In IAM Identity Center, crea un set di autorizzazioni che segua la migliore pratica di applicazione delle autorizzazioni con privilegi minimi.

Per istruzioni, consulta Creare un set di autorizzazioni nella Guida per l'utente.AWS IAM Identity Center

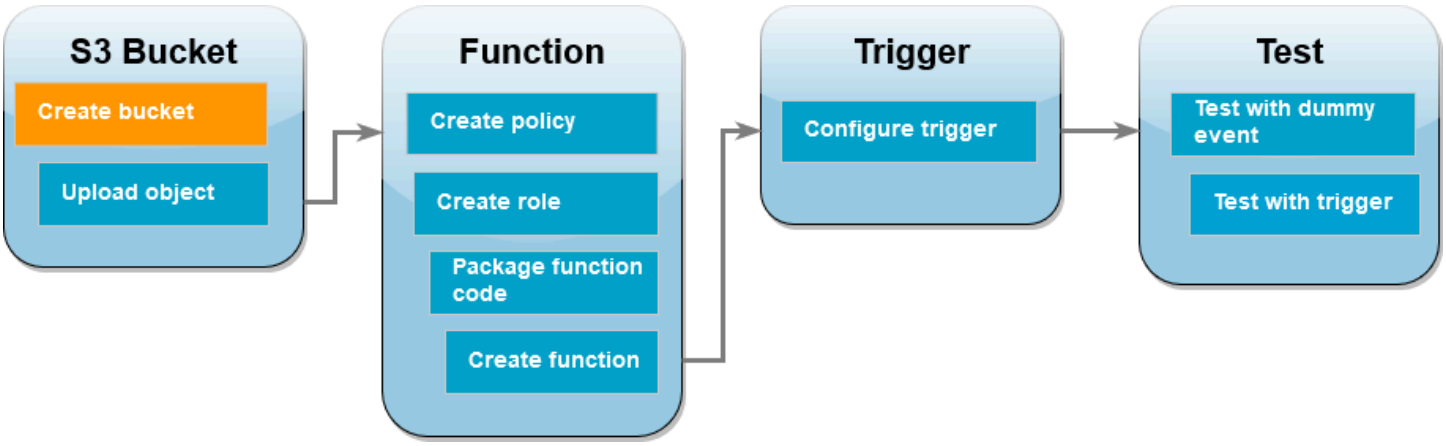
2. Assegna gli utenti a un gruppo, quindi assegna l'accesso Single Sign-On al gruppo.

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente.AWS IAM Identity Center

Se desideri utilizzare il per AWS CLI completare il tutorial, installa la versione più recente di AWS Command Line Interface.

Per il codice della funzione Lambda, puoi utilizzare Python o Node.js. Installa gli strumenti di supporto linguistico e un gestore di pacchetti per il linguaggio che desideri utilizzare.

Creazione di due bucket Amazon S3



Per prima cosa, crea due bucket Amazon S3. Il primo bucket è il bucket di origine in cui caricherai le tue immagini. Il secondo bucket viene utilizzato da Lambda per salvare la miniatura ridimensionata quando richiami la tua funzione.

AWS Management Console

Creazione di bucket Amazon S3 (console)

1. Nella console Amazon S3, apri la pagina Bucket.
2. Scegliere Create bucket (Crea bucket).
3. In General configuration (Configurazione generale), eseguire le operazioni seguenti:

a. 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 (-).

b. Per Regione AWS, scegli la Regione AWS più vicina alla tua posizione geografica. Più avanti nel tutorial, devi creare la tua funzione Lambda nella stessa Regione AWS, quindi prendi nota della regione che hai scelto.

4. Lascia tutte le altre opzioni impostate sui valori predefiniti e scegli Crea bucket.

5. Ripeti i passaggi da 1 a 4 per creare il bucket di destinazione. Per Nome del bucket, inserisci **`DOC-EXAMPLE-SOURCE-BUCKET-resized`**, dove **`DOC-EXAMPLE-SOURCE-BUCKET`** è il nome del bucket di origine che hai appena creato.

AWS CLI

Creazione dei 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.

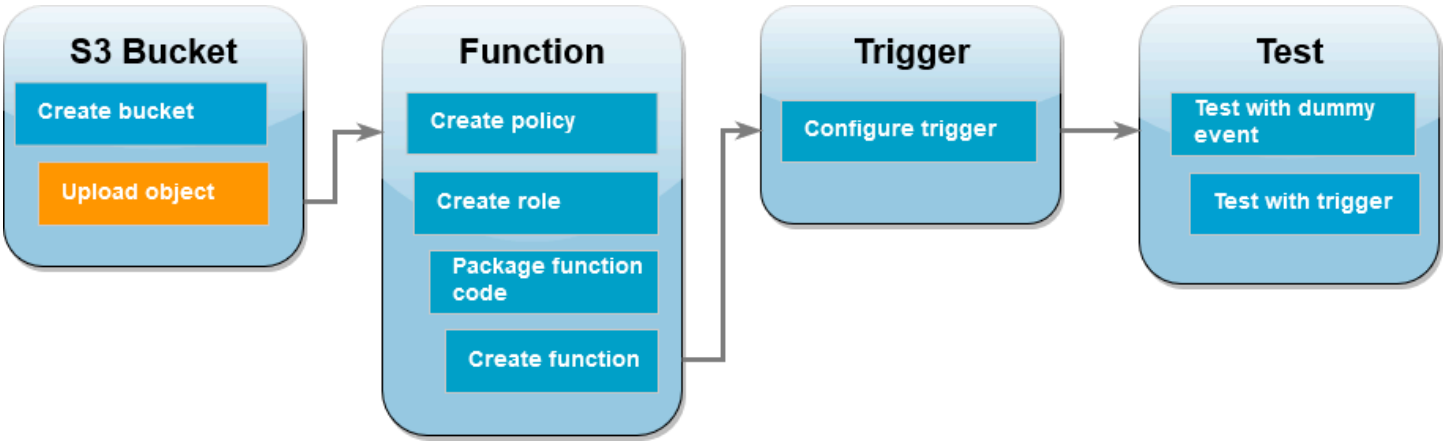
```
aws s3api create-bucket --bucket DOC-EXAMPLE-SOURCE-BUCKET --region us-west-2 \
--create-bucket-configuration LocationConstraint=us-west-2
```

Più avanti nel tutorial, devi creare la tua funzione Lambda nello Regione AWS stesso 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 **`DOC-EXAMPLE-SOURCE-BUCKET-resized`**, dove **`DOC-EXAMPLE-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 bucket sorgente.

```
aws s3api create-bucket --bucket DOC-EXAMPLE-SOURCE-BUCKET-resized --region us-
west-2 \
--create-bucket-configuration LocationConstraint=us-west-2
```

Caricamento di un'immagine di test nel bucket di origine



Più avanti nel tutorial, testerai la tua funzione Lambda invocandola utilizzando la console o AWS CLI 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.

AWS Management Console

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.

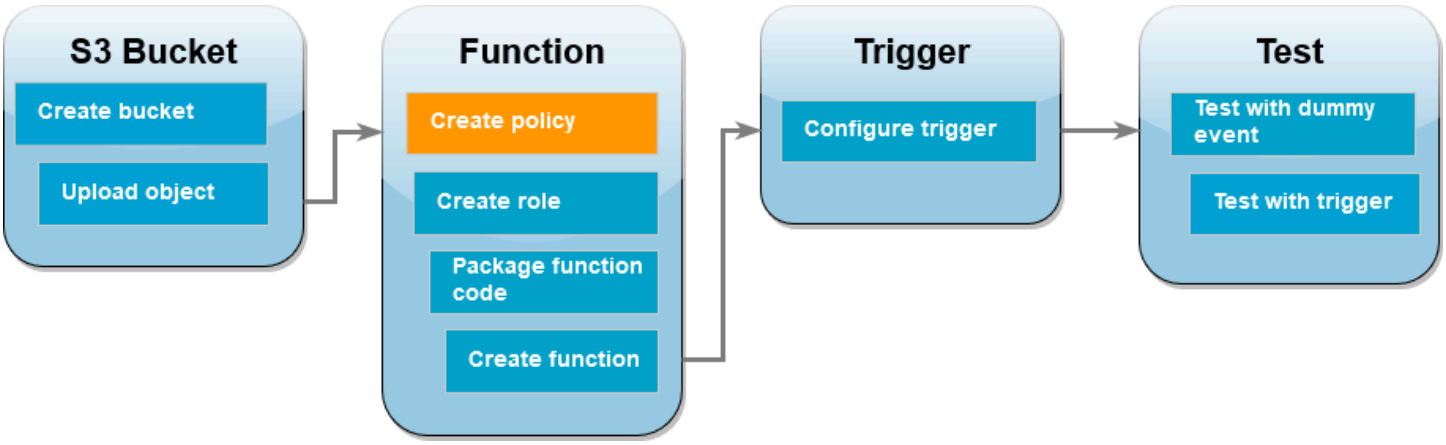
AWS CLI

Caricamento di un'immagine di test nel bucket di origine (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 DOC-EXAMPLE-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 funzione le autorizzazioni necessarie per accedere ad altre risorse. AWS Per questo tutorial, la policy fornisce le autorizzazioni di lettura e scrittura Lambda per i bucket Amazon S3 e consente di scrivere su Amazon Logs. CloudWatch

AWS Management Console

Creazione della policy (console)

1. Apri la pagina Policies della console (IAM). AWS Identity and Access Management
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. Seleziona Successivo.
5. In Dettagli sulla policy, per Nome policy, inserisci **`LambdaS3Policy`**.
6. Scegli Crea policy.

AWS CLI

Creazione della policy (AWS CLI)

1. Salva il seguente JSON in un file denominato `policy.json`.

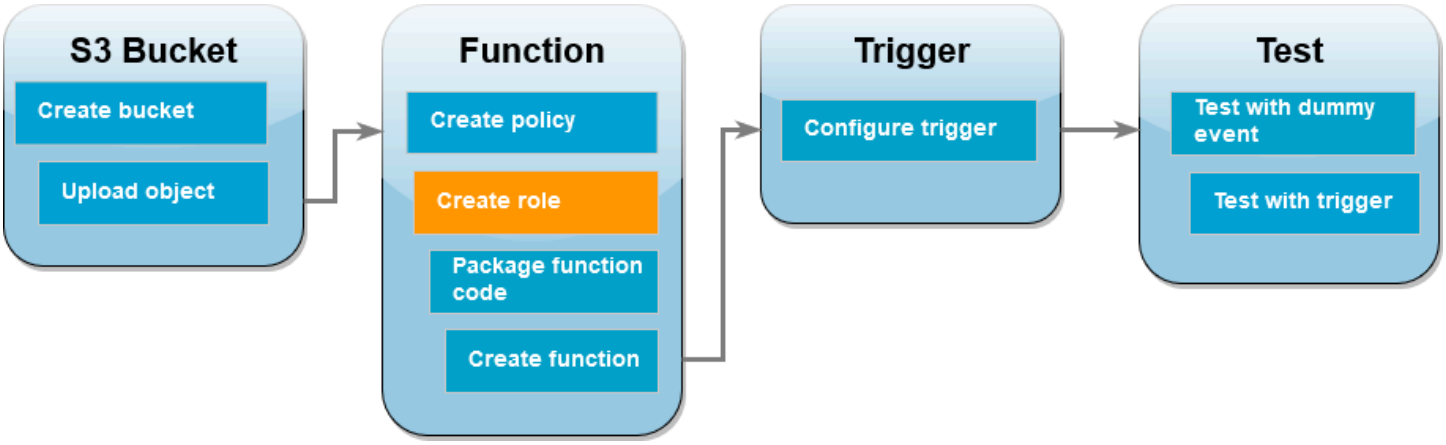
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "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' Servizi AWS accesso 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.

AWS Management Console

Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione (console)

1. Apri la pagina Ruoli della console (IAM).

2. Scegli Crea ruolo.
3. Per Tipo di entità attendibile, seleziona Servizio AWS, mentre per Caso d'uso, seleziona Lambda.
4. Seleziona Successivo.
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. Seleziona Successivo.
6. In Dettagli ruolo, per Nome ruolo, inserisci **`LambdaS3Role`**.
7. Scegli Crea ruolo.

AWS CLI

Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione (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 servizio principale l'`lambda.amazonaws.com`autorizzazione a chiamare l'azione AWS Security Token Service ()AWS STS. `AssumeRole`

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

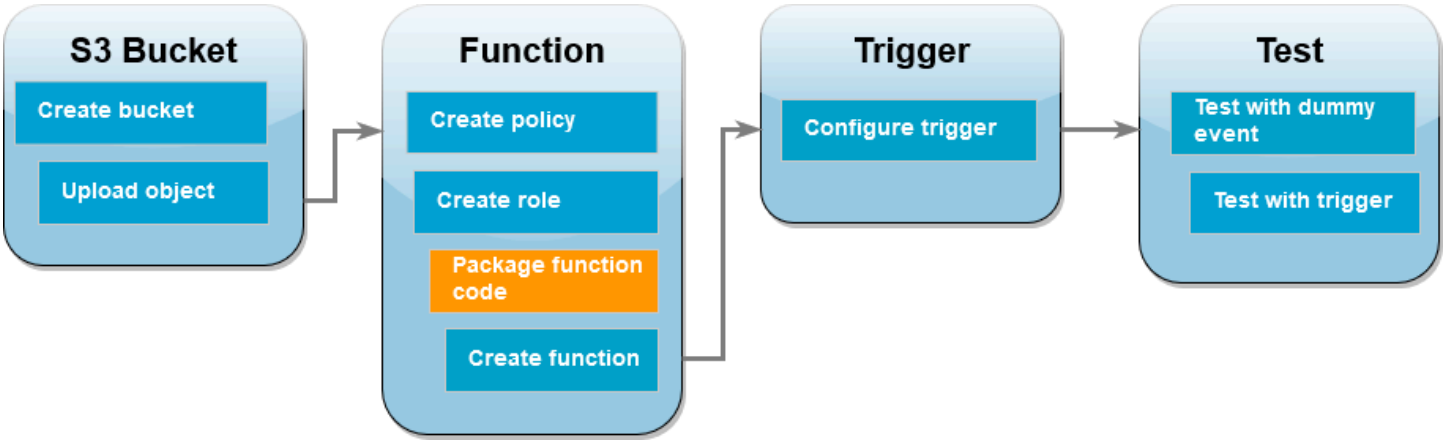
2. Nella directory in cui hai salvato il documento della policy di attendibilità JSON, esegui il comando della CLI sotto riportato per creare il ruolo di esecuzione.

```
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 polizza con il tuo numero di conto.

```
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

Creazione del pacchetto di implementazione (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. Salvare il codice della funzione seguente in un file denominato `index.mjs`. Assicurati di sostituirlo `'us-west-2'` con quello 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-west-2'});

// 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);
  };
```

3. 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
```

4. 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 for 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 3.9 \
--only-binary=:all: --upgrade \
pillow boto3
```

La libreria Pillow contiene codice C/C++. Utilizzando le opzioni `--platform manylinux_2014_x86_64` e `--only-binary=:all:`, pip scaricherà e installerà una versione di Pillow che contiene file binari precompilati compatibili con il sistema operativo Amazon Linux 2. Ciò garantisce che il pacchetto di implementazione funzioni nell'ambiente di esecuzione Lambda, indipendentemente dal sistema operativo e dall'architettura del computer di compilazione locale.

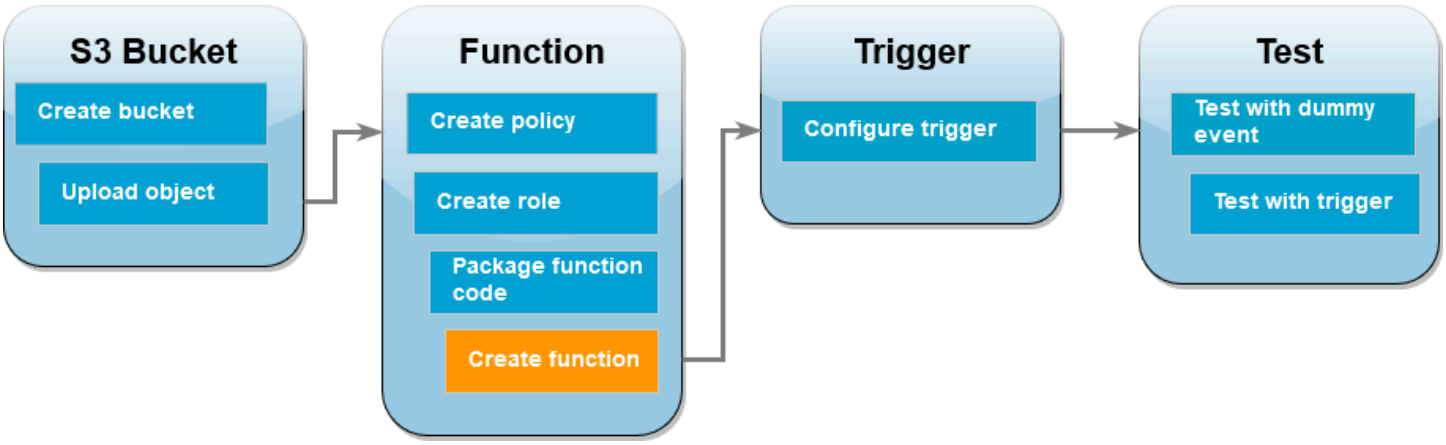
3. Crea un file .zip contenente il codice dell'applicazione e le librerie Pillow e Boto3. Su Linux o MacOS, esegui i comandi riportati di seguito dall'interfaccia della linea di comando.

```
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



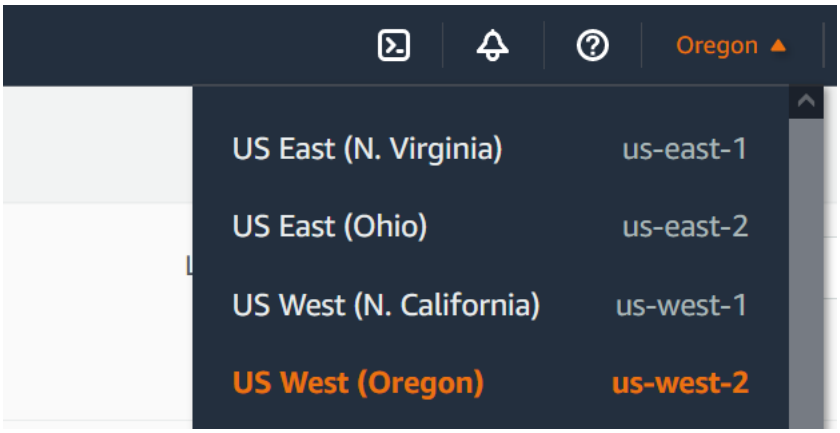
Puoi creare la tua funzione Lambda utilizzando la console AWS CLI o la console Lambda. Segui le istruzioni per il linguaggio scelto per creare la funzione.

AWS Management Console

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 bucket in Regione AWS cui hai creato il tuo 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 18.x o Python 3.9, a seconda del linguaggio che hai scelto per la 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

Creazione della 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 ID. Account AWS Per il parametro `region`, sostituisci `us-west-2` con la regione in cui hai creato i bucket Amazon S3.

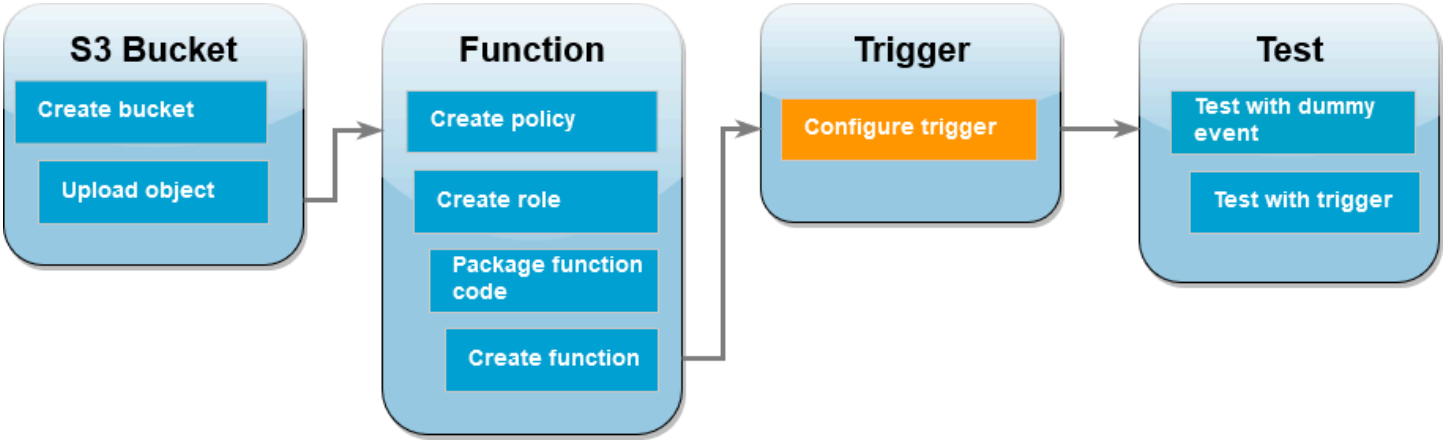
  - Per Node.js, esegui il comando sotto riportato dalla directory contenente il file `function.zip`.

```
aws lambda create-function --function-name CreateThumbnail \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs18.x \
--timeout 10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-west-2
```

- 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 python3.9 --timeout 10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-west-2
```

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.

AWS Management Console

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. Scegli 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

Configurazione del trigger 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 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 del 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:::DOC-EXAMPLE-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 lo Account AWS crei 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.

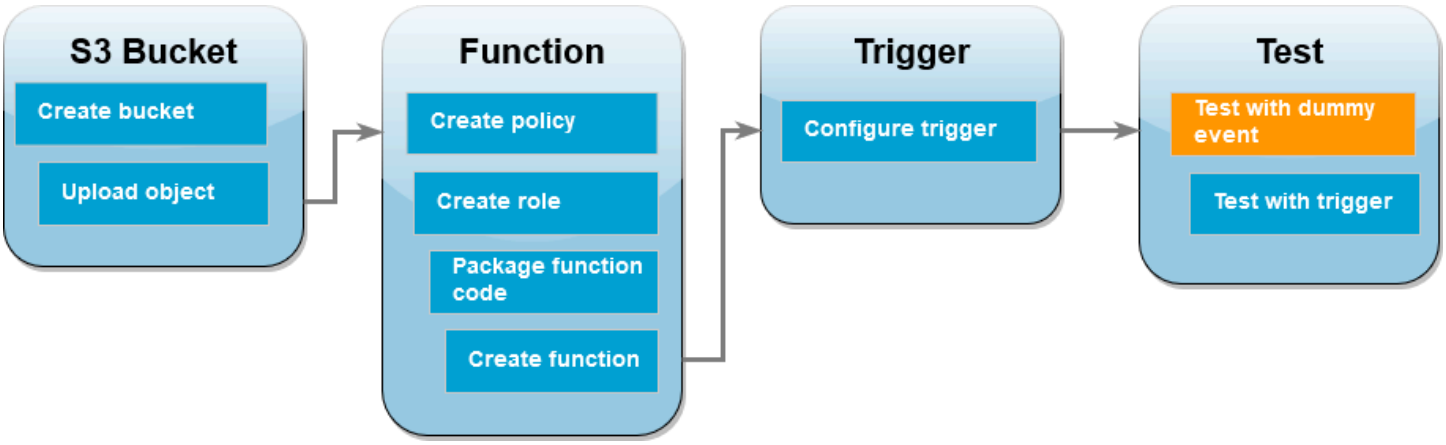
```
{
"LambdaFunctionConfigurations": [
    {
      "Id": "CreateThumbnailEventConfiguration",
      "LambdaFunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:CreateThumbnail",
      "Events": [ "s3:ObjectCreated:Put" ]
    }
  ]
}
```

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 `DOC-EXAMPLE-SOURCE-BUCKET` con il nome del tuo bucket di origine.

```
aws s3api put-bucket-notification-configuration --bucket DOC-EXAMPLE-SOURCE-
BUCKET \
--notification-configuration file://notification.json
```

Per ulteriori informazioni sul `put-bucket-notification-configuration` comando e sull'`notification-configuration`opzione, consulta put-bucket-notification-configurationla AWS CLI Command Reference.

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 documento in formato JSON che contiene i dati che una 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.

AWS Management Console

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.
      - Infatti`awsRegion`, `us-east-1` sostituiscilo con quello in Regione AWS cui hai creato i bucket Amazon S3.
      - Per `name`, sostituisci `DOC-EXAMPLE-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": "DOC-EXAMPLE-BUCKET",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::DOC-EXAMPLE-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

Test di una 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:

   1. Infatti`awsRegion`, `us-west-2` sostituiscilo con quello in Regione AWS cui hai creato i bucket Amazon S3.
   2. Per `name`, sostituisci `DOC-EXAMPLE-SOURCE-BUCKET` con il nome del bucket di origine Amazon S3.
   3. Per `key`, sostituisci `HappyFace.jpg` 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-west-2",
      "eventTime":"1970-01-01T00:00:00.000Z",
      "eventName":"ObjectCreated:Put",
      "userIdentity":{
        "principalId":"AIDAJDPLRKLG7UEXAMPLE"
      },
      "requestParameters":{
```

```
        "sourceIPAddress":"127.0.0.1"
      },
      "responseElements":{
        "x-amz-request-id":"C3D13FE58DE4C810",
        "x-amz-id-2":"FMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/
JRWeUWerMUE5JgHvANOjpD"
      },
      "s3":{
        "s3SchemaVersion":"1.0",
        "configurationId":"testConfigRule",
        "bucket":{
          "name":"DOC-EXAMPLE-SOURCE-BUCKET",
          "ownerIdentity":{
            "principalId":"A3NL1KOZZKExample"
          },
          "arn":"arn:aws:s3:::DOC-EXAMPLE-SOURCE-BUCKET"
        },
        "object":{
          "key":"HappyFace.jpg",
          "size":1024,
          "eTag":"d41d8cd98f00b204e9800998ecf8427e",
          "versionId":"096fKKXTRTtl3on89fVO.nfljtsv6qko"
        }
      }
    }
  ]
}
```

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' cli-binary-format opzione è obbligatoria se si utilizza la versione 2 di. 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.

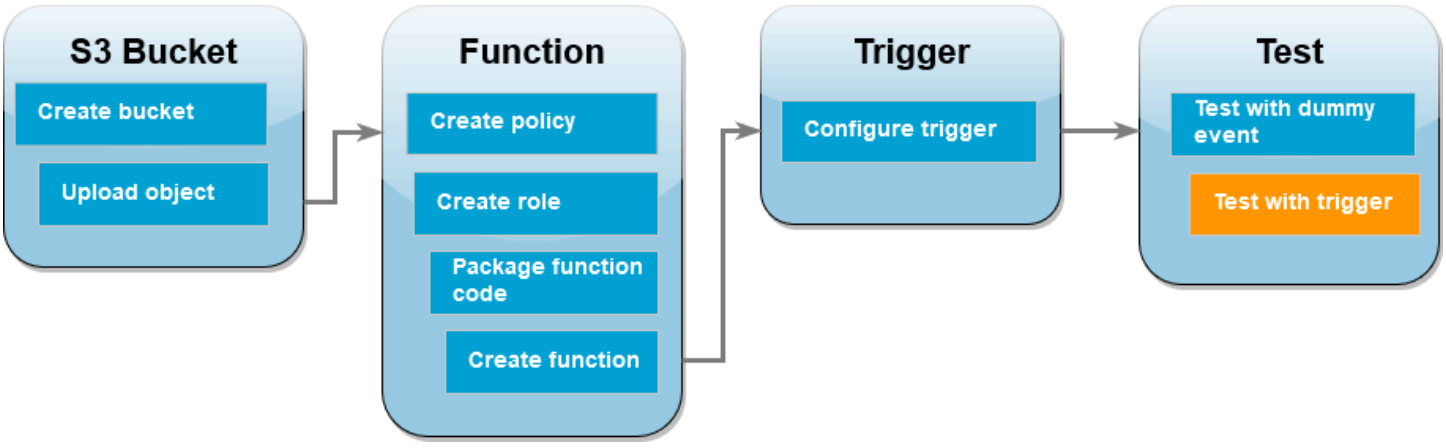
3. Verifica che la funzione abbia creato una versione in miniatura dell'immagine e l'abbia salvata nel bucket Amazon S3 di destinazione. Esegui il comando della CLI sotto riportato sostituendo `DOC-EXAMPLE-SOURCE-BUCKET-resized` con il nome del tuo bucket di destinazione.

```
aws s3api list-objects-v2 --bucket DOC-EXAMPLE-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, puoi testare la configurazione completa aggiungendo un file di immagine al tuo bucket di origine 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.

AWS Management Console

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

Test della funzione Lambda tramite 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 DOC-EXAMPLE-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 `DOC-EXAMPLE-SOURCE-BUCKET-resized` con il nome del tuo bucket di destinazione.

```
aws s3api list-objects-v2 --bucket DOC-EXAMPLE-SOURCE-BUCKET-resized
```

Se la tua funzione viene eseguita correttamente, vedrai un output 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 a tuo Account AWS carico.

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 **`delete`** nel campo di immissione testo e scegli Delete (Elimina).

Per eliminare la policy creata

1. Aprire la pagina Policies (Policy) nella console IAM.
2. Seleziona la politica che hai creato (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 Delete (Elimina).
4. Inserisci il nome del bucket nel campo di immissione testo.
5. Scegli Delete Bucket (Elimina bucket).

Utilizzo AWS Lambda con operazioni batch di 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.

L'esempio seguente mostra l'evento che Amazon S3 invia alla funzione Lambda per un oggetto denominato customerImage1.jpg nel bucket DOC-EXAMPLE-BUCKET.

Example Evento di richiesta batch Amazon S3

```
{
"invocationSchemaVersion": "1.0",
    "invocationId": "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
    "job": {
        "id": "f3cc4f60-61f6-4a2b-8a21-d07600c373ce"
    },
    "tasks": [
        {
            "taskId": "dGFza2lkZ29lc2hlcmUK",
            "s3Key": "customerImage1.jpg",
            "s3VersionId": "1",
            "s3BucketArn": "arn:aws:s3:::DOC-EXAMPLE-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" : "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
  "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.

Trasformazione di oggetti S3 con S3 Object Lambda

Con S3 Object Lambda puoi aggiungere il tuo codice alle richieste GET, HEAD e LIST di Amazon S3 per modificare ed elaborare i dati prima che vengano restituiti a un'applicazione. Puoi utilizzare il codice personalizzato per modificare i dati restituiti dalle richieste GET, HEAD o LIST S3 standard per filtrare le righe, ridimensionare le immagini in modo dinamico, oscurare i dati riservati e molto altro. Grazie alle funzioni AWS Lambda, il codice viene eseguito su un'infrastruttura completamente gestita da AWS, eliminando la necessità di creare e archiviare copie derivate dei dati o di eseguire proxy, tutto senza apportare modifiche alle applicazioni.

Per ulteriori informazioni, consulta Trasformazione di oggetti con S3 Object Lambda.

Tutorial

- Trasformazione dei dati per l'applicazione con Amazon S3 Object Lambda
- Rilevamento e redazione dei dati PII con Amazon S3 Object Lambda e Amazon Comprehend
- Utilizzo di Amazon S3 Object Lambda per aggiungere filigrane alle immagini in modo dinamico man mano che vengono recuperate

Utilizzo di AWS Lambda con Secrets Manager

La tua funzione AWS Lambda può interagire con AWS Secrets Manager utilizzando l'API Secrets Manager o uno qualsiasi dei Software Development Kit (SDK) AWS. Puoi utilizzare l'estensione Lambda di AWS Parameters and Secrets per recuperare e memorizzare nella cache i segreti AWS Secrets Manager nelle funzioni Lambda senza utilizzare un SDK. Per ulteriori informazioni, consulta Utilizzo dei AWS Secrets Manager segreti nelle funzioni AWS Lambda.

Uso di AWS Lambda con Amazon SES

Quando si utilizza Amazon SES per ricevere messaggi, è possibile configurare Amazon SES per chiamare la funzione Lambda quando arriva il messaggio. Il servizio può quindi invocare la funzione Lambda passando l'evento e-mail in ingresso, ovvero un messaggio Amazon SES in un evento Amazon SNS, come un parametro.

Example Evento messaggio di Amazon SES

```
{
  "Records": [
    {
      "eventVersion": "1.0",
      "ses": {
        "mail": {
          "commonHeaders": {
            "from": [
              "Jane Doe <janedoe@example.com>"
            ],
            "to": [
              "johndoe@example.com"
            ],
            "returnPath": "janedoe@example.com",
            "messageId": "<0123456789example.com>",
            "date": "Wed, 7 Oct 2015 12:34:56 -0700",
            "subject": "Test Subject"
          },
          "source": "janedoe@example.com",
          "timestamp": "1970-01-01T00:00:00.000Z",
          "destination": [
            "johndoe@example.com"
          ],
          "headers": [
            {
              "name": "Return-Path",
              "value": "<janedoe@example.com>"
            },
            {
              "name": "Received",
              "value": "from mailer.example.com (mailer.example.com [203.0.113.1])
 by inbound-smtp.us-west-2.amazonaws.com with SMTP id o3vrnil0e2ic for
 johndoe@example.com; Wed, 07 Oct 2015 12:34:56 +0000 (UTC)"
            },
```

```
            {
              "name": "DKIM-Signature",
              "value": "v=1; a=rsa-sha256; c=relaxed/relaxed; d=example.com;
 s=example; h=mime-version:from:date:message-id:subject:to:content-type;
 bh=jX3F0bCAI7sIbkHyy3mLYO28ieDQz2R0P8HwQkklFj4=; b=sQwJ+LMe9RjkesGu
+vqU56asvMhrLRRYrWCbV"
            },
            {
              "name": "MIME-Version",
              "value": "1.0"
            },
            {
              "name": "From",
              "value": "Jane Doe <janedoe@example.com>"
            },
            {
              "name": "Date",
              "value": "Wed, 7 Oct 2015 12:34:56 -0700"
            },
            {
              "name": "Message-ID",
              "value": "<0123456789example.com>"
            },
            {
              "name": "Subject",
              "value": "Test Subject"
            },
            {
              "name": "To",
              "value": "johndoe@example.com"
            },
            {
              "name": "Content-Type",
              "value": "text/plain; charset=UTF-8"
            }
          ],
          "headersTruncated": false,
          "messageId": "o3vrnil0e2ic28tr"
        },
        "receipt": {
          "recipients": [
            "johndoe@example.com"
          ],
          "timestamp": "1970-01-01T00:00:00.000Z",
```

```
          "spamVerdict": {
            "status": "PASS"
          },
          "dkimVerdict": {
            "status": "PASS"
          },
          "processingTimeMillis": 574,
          "action": {
            "type": "Lambda",
            "invocationType": "Event",
            "functionArn": "arn:aws:lambda:us-west-2:111122223333:function:Example"
          },
          "spfVerdict": {
            "status": "PASS"
          },
          "virusVerdict": {
            "status": "PASS"
          }
        }
      },
      "eventSource": "aws:ses"
    }
  ]
}
```

Per ulteriori informazioni, consulta Operazione Lambda nella Amazon SES Developer Guide.

Richiamo di funzioni Lambda con le notifiche di 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.

Per l'invocazione asincrona, Lambda mette in coda il messaggio e gestisce i nuovi 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 mappature delle sorgenti degli eventi 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 ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

Argomenti

Aggiungere 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 configura

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 di una funzione per la quale desideri aggiungere il trigger.
3. Scegli Configurazione, quindi scegli Trigger.
4. Selezionare Add trigger (Aggiungi trigger).
5. In Configurazione Trigger, nel menu a discesa, scegli SNS.
6. Per l'argomento SNS, scegli l'argomento SNS a cui iscriverti.

Aggiungere manualmente un trigger di argomento Amazon SNS per una funzione Lambda

Per configurare manualmente un trigger SNS per una funzione Lambda, devi completare i seguenti passaggi:

- Definisci una policy basata sulle risorse per la tua funzione per consentire a SNS di richiamarla.
- Sottoscrivi la tua funzione Lambda all'argomento Amazon SNS.

> **Note**
>
> Se l'argomento SNS e la funzione Lambda si trovano in account AWS diversi, è inoltre necessario concedere autorizzazioni aggiuntive per consentire le sottoscrizioni tra account all'argomento SNS. Per ulteriori informazioni, consulta Concedere l'autorizzazione per più account per l'abbonamento ad Amazon SNS.

Puoi usare il AWS Command Line Interface (AWS CLI) per completare entrambi questi passaggi. Innanzitutto, per definire una politica basata sulle risorse per una funzione Lambda che consenta le chiamate SNS, usa il comando seguente. AWS CLI Assicurati di sostituire il valore di `--function-name` con il nome della funzione Lambda e il valore di `--source-arn` con l'argomento SNS ARN.

```
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 comando seguente. AWS CLI Sostituisci il valore di `--topic-arn` con il tuo argomento SNS ARN e il valore `--notification-endpoint` di con la funzione Lambda ARN.

```
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 dell'evento SNS

Amazon SNS richiama la funzione in modo asincrono con un evento che contiene un messaggio e dei metadati.

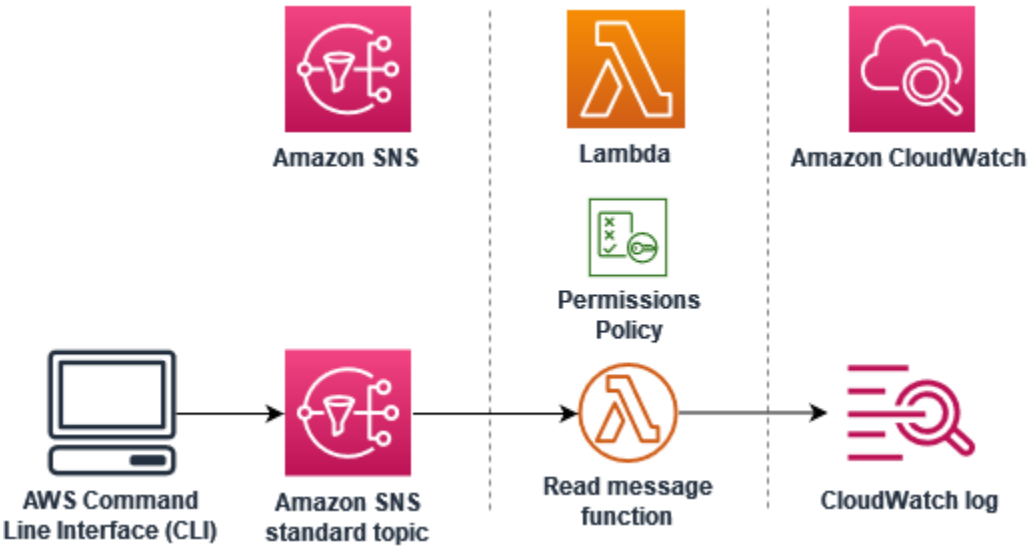
Example Evento messaggio di Amazon SNS

```
{
  "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&amp;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 un'unica funzione Account AWS per abbonarti a un argomento di Amazon Simple Notification Service (Amazon SNS) in un altro argomento. Account AWS Quando pubblichi messaggi sul tuo argomento Amazon SNS, la funzione Lambda legge il contenuto del messaggio e lo invia ad Amazon Logs. CloudWatch Per completare questo tutorial, usa il (). 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 leggerà 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 invii nei log. CloudWatch

Completando questi passaggi, imparerai come configurare un argomento Amazon SNS per richiamare una funzione Lambda. Imparerai anche come creare una policy AWS Identity and Access Management (IAM) che autorizzi una risorsa in un'altra Account AWS a richiamare Lambda.

Nel tutorial, vengono utilizzati due Account AWS separati. I AWS CLI comandi lo illustrano utilizzando due profili denominati chiamati `accountA` e `accountB`, ciascuno configurato per l'uso con un altro. Account AWS Per informazioni su come configurare l'utilizzo AWS CLI di profili diversi, consulta Impostazioni dei file di configurazione e credenziali nella Guida per l'AWS Command Line Interface utente della versione 2. Assicurati di configurare lo stesso valore predefinito Regione AWS per entrambi i profili.

Se i AWS CLI profili creati per i due Account AWS utilizzano nomi diversi o se utilizzi il profilo predefinito e un profilo denominato, modifica i AWS CLI comandi nei passaggi seguenti in base alle esigenze.

Prerequisiti

Iscrivetevi per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come procedura consigliata in materia di sicurezza, assegnate l'accesso amministrativo a un utente e utilizzate solo l'utente root per eseguire attività che richiedono l'accesso da parte dell'utente root.

AWS ti invia un'e-mail di conferma dopo il completamento della procedura di registrazione. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .
2. In IAM Identity Center, concedi l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con le impostazioni predefinite IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accedi come utente con accesso amministrativo

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso ad altri utenti

1. In IAM Identity Center, crea un set di autorizzazioni che segua la migliore pratica di applicazione delle autorizzazioni con privilegi minimi.

   Per istruzioni, consulta Creare un set di autorizzazioni nella Guida per l'utente.AWS IAM Identity Center
2. Assegna gli utenti a un gruppo, quindi assegna l'accesso Single Sign-On al gruppo.

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente.AWS IAM Identity Center

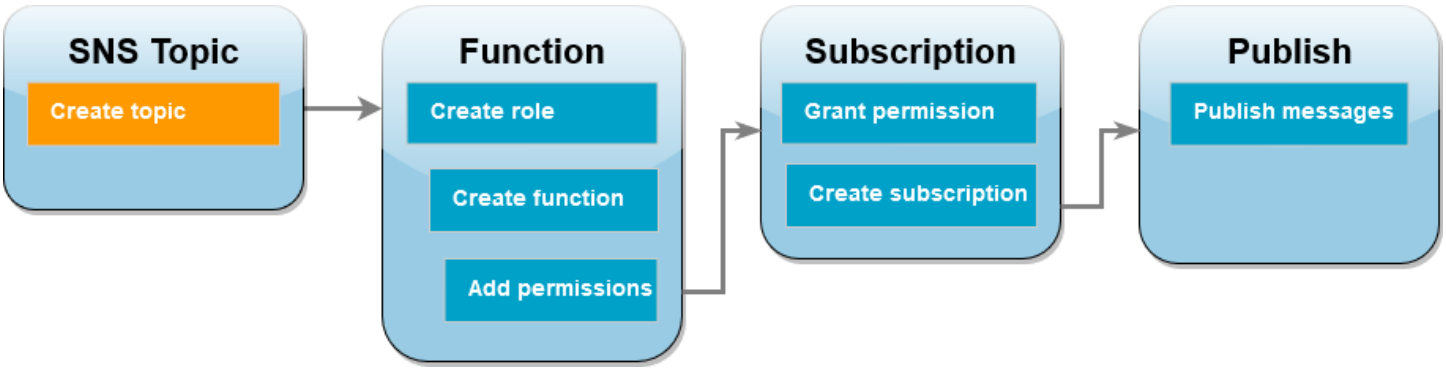
Installa il AWS Command Line Interface

Se non l'hai ancora installato AWS Command Line Interface, segui i passaggi indicati in Installazione o aggiornamento della versione più recente di 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 versione integrata su Windows 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.

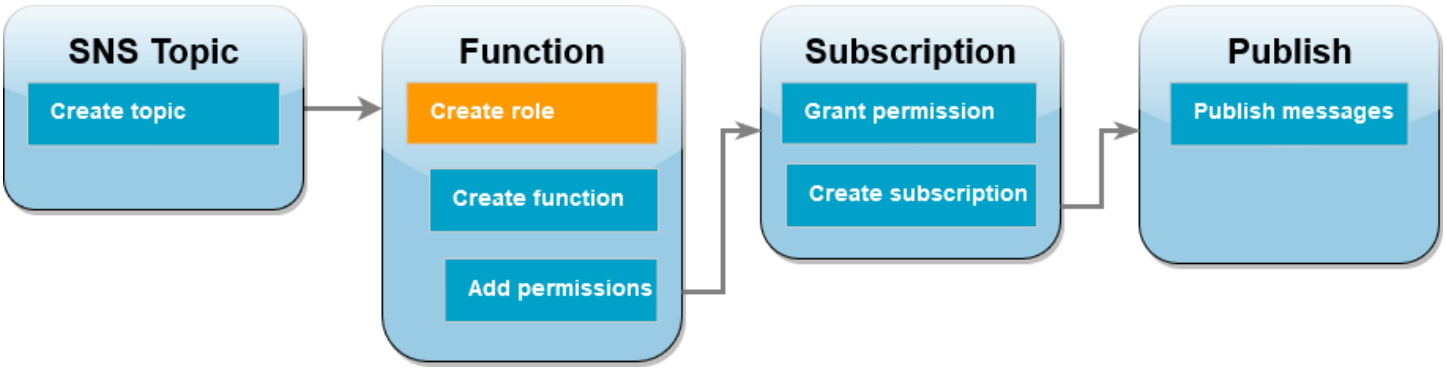
  ```
  aws sns create-topic --name sns-topic-for-lambda --profile accountA
  ```

  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. Sarà necessario in seguito nel tutorial, quando aggiungerai autorizzazioni alla funzione Lambda per effettuare la sottoscrizione all'argomento.

Creazione di un ruolo di esecuzione della funzione (account B)

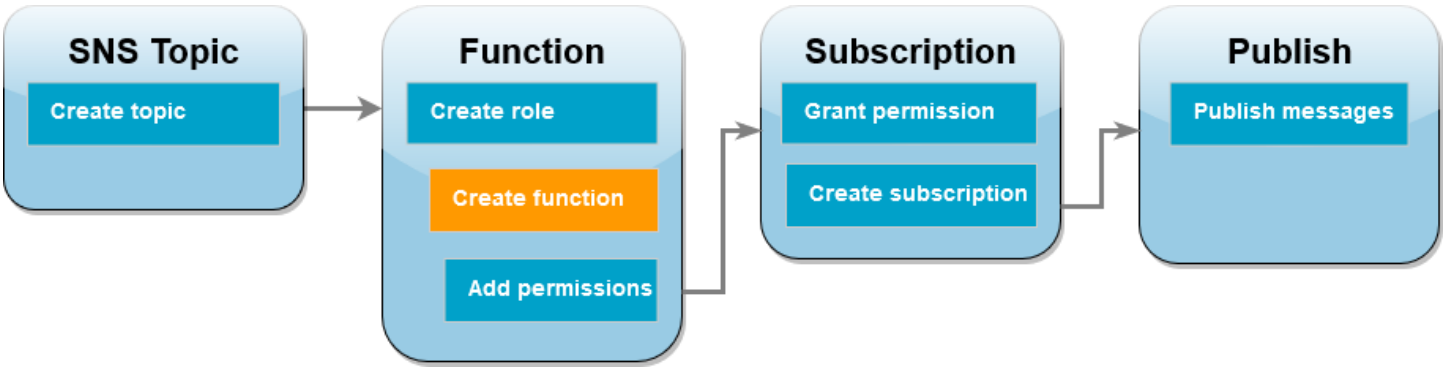


Un ruolo di esecuzione è un ruolo IAM che concede a una funzione Lambda l'autorizzazione ad AWS accedere a servizi e risorse. Prima di creare la funzione nell'account B, crei un ruolo che fornisce alla funzione le autorizzazioni di base per scrivere i log nei registri. CloudWatch Aggiungeremo le autorizzazioni per la lettura dal tuo argomento Amazon SNS in un passaggio successivo.

Creazione di un ruolo di esecuzione

1. Nell'account B apri la pagina dei ruoli nella console IAM.
2. Scegli Crea ruolo.
3. Per Tipo di entità attendibile, scegli Servizio AWS .
4. In Caso d'uso, scegli Lambda.
5. Seleziona Successivo.
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. Seleziona Successivo.
7. Completa la creazione del ruolo effettuando le seguenti operazioni:
    a. In Dettagli ruolo, immetti **`lambda-sns-role`** per Nome ruolo.
    b. Scegli Crea ruolo.

Creare una funzione Lambda (account B)



Crea una funzione Lambda che elabora i messaggi Amazon SNS. Il codice funzione registra il contenuto dei messaggi di ogni record in Amazon CloudWatch Logs.

Questo tutorial utilizza il runtime Node.js 18.x, ma è fornito anche un 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 è nel primo esempio mostrato nella JavaScriptscheda.

.NET

AWS SDK for .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.DefaultLambdaJsonSeri

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.

Consumo di un evento SNS con Lambda utilizzando 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 (v3 JavaScript)

> **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.

Consumo di un evento SNS con Lambda utilizzando 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 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 nodejs18.x \
    --role arn:aws:iam::<AccountB_ID>:role/lambda-sns-role  \
    --timeout 60 --profile accountB
```

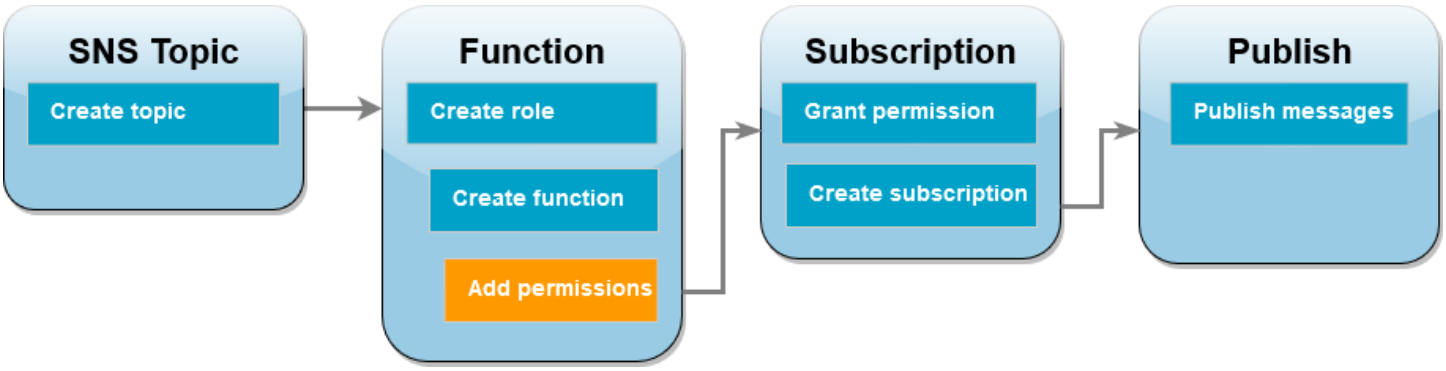
Verrà visualizzato un output simile al seguente.

```
{
    "FunctionName": "Function-With-SNS",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:Function-With-
SNS",
    "Runtime": "nodejs18.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. Sarà necessario in seguito nel tutorial, quando aggiungerai autorizzazioni per consentire ad Amazon SNS di richiamare la 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. Si aggiunge questa dichiarazione utilizzando 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
```

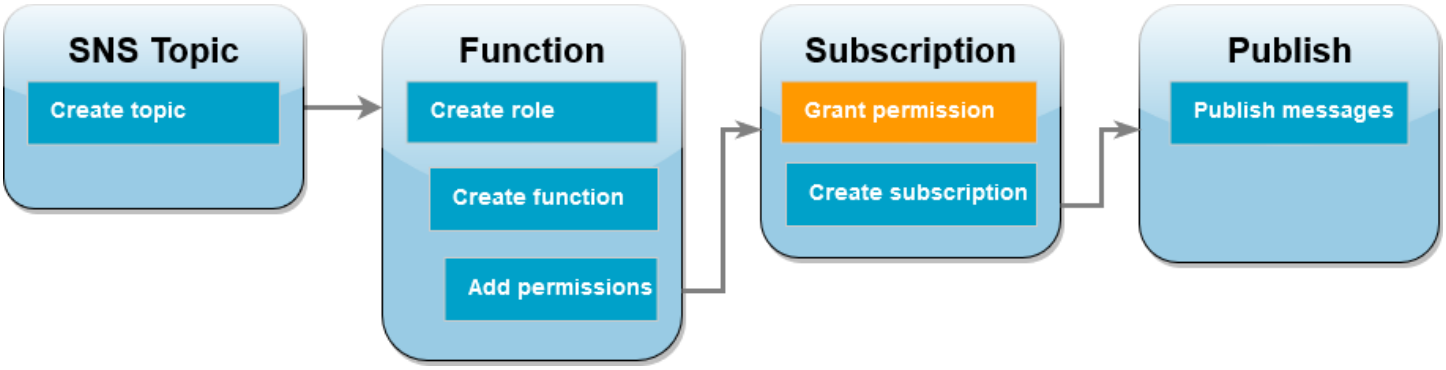
  Verrà visualizzato un output simile al seguente.

```
{
    "Statement": "{\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":
      \"arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda\"}},
```

```
      \"Action\":[\"lambda:InvokeFunction\"],
      \"Resource\":\"arn:aws:lambda:us-east-1:<AccountB_ID>:function:Function-With-
SNS\",
      \"Effect\":\"Allow\",\"Principal\":{\"Service\":\"sns.amazonaws.com\"},
      \"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'area 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)



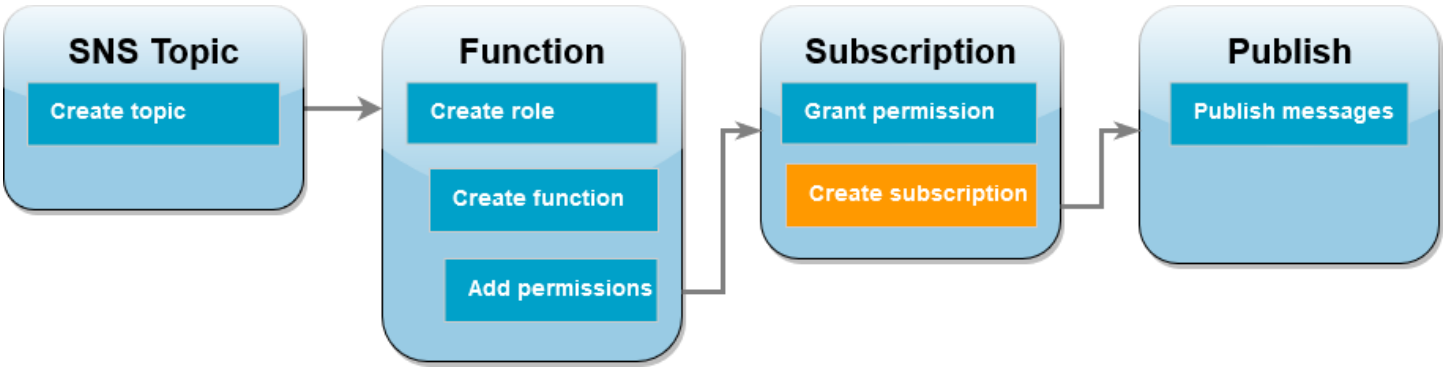
Perché la funzione Lambda nell'account B sottoscriva l'argomento Amazon SNS che hai creato nell'account A, è necessario concedere l'autorizzazione per l'account B in modo da sottoscrivere il tuo argomento. Concedi questa autorizzazione utilizzando il AWS CLI `add-permission` comando.

Concessione dell'autorizzazione per consentire all'account B la sottoscrizione all'argomento

- Nell'account A, esegui il seguente AWS CLI comando. 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

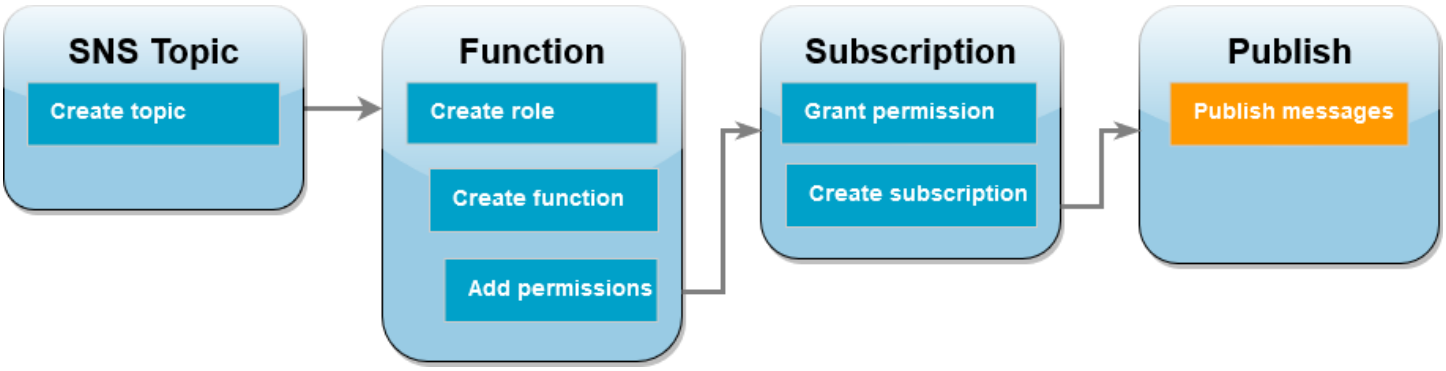
- Nell'account B, esegui il AWS CLI comando seguente. Usa la regione predefinita in cui hai creato l'argomento e gli ARN per l'argomento e la funzione Lambda.

```
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 ha sottoscritto l'argomento Amazon SNS nell'account A, è il momento di testare la configurazione pubblicando messaggi sull'argomento. Per confermare che Amazon SNS ha richiamato la funzione Lambda, utilizza CloudWatch Logs per visualizzare l'output della funzione.

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
   ```

   In questo modo verrà restituito un ID messaggio con un identificatore univoco, indicando che il messaggio è stato accettato da Amazon SNS. Amazon SNS prova quindi a consegnare il messaggio a tutti i sottoscrittori dell'argomento. Per confermare che Amazon SNS ha richiamato la funzione Lambda, usa CloudWatch Logs per visualizzare l'output della funzione:
3. Nell'account B, apri la pagina dei gruppi di log della CloudWatch console Amazon.
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 funzione è stata richiamata correttamente, vedrai un output simile al seguente che mostra il contenuto del messaggio pubblicato sull'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 addebiti inutili ai tuoi 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. Scegli Elimina.
4. Inserisci **`delete me`** nel campo di immissione del testo.
5. Scegli 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 **`delete`** 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).

Procedure consigliate per l'utilizzo AWS Lambda delle funzioni

Di seguito sono indicate le best practice per l'utilizzo di AWS Lambda.

Argomenti

Per ulteriori informazioni sulle best practice per le applicazioni Lambda, consulta Progettazione delle applicazioni in Serverless Land. Puoi anche contattare il team del tuo AWS account e richiedere una revisione dell'architettura.

Codice della funzione

- Separare il gestore Lambda dalla logica principale. In questo modo è possibile creare una funzione di cui è più semplice eseguire l'unit test. In Node.js, l'aspetto è analogo al seguente:

```
exports.myHandler = function(event, context, callback) {
 var foo = event.foo;
 var bar = event.bar;
 var result = MyLambdaFunction (foo, bar);

 callback(null, result);
}

function MyLambdaFunction (foo, bar) {
 // MyLambdaFunction logic here
}
```

- 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, vedere 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.
- Controllare le dipendenze nel pacchetto di distribuzione della funzione. L'ambiente di AWS Lambda esecuzione contiene diverse librerie come l' AWS SDK per i runtime Node.js e Python (un elenco completo è disponibile qui:). Runtime Lambda 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 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. Per le funzioni create in Java o.NET Core, evita di caricare l'intera libreria AWS SDK come parte del pacchetto di distribuzione. Utilizzare invece in modo selettivo i moduli che prelevano i componenti dell'SDK necessari (ad esempio i moduli SDK Dynamo DB e Amazon S3 e le librerie di base Lambda).
- 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 metodo è più rapido rispetto all'inserimento di tutto il codice della funzione in un unico file .jar con un elevato numero di file `.class`. Per istruzioni, consulta Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR.

- 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.
- Evitare di utilizzare codice ricorsivo nella funzione Lambda in base al quale la funzione chiama automaticamente se stessa finché non vengono soddisfatti determinati criteri arbitrari. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se questa operazione viene eseguita in modo accidentale, impostare immediatamente il limite di esecuzioni simultanee della funzione sul valore `0` per interrompere tutte le invocazioni alla 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 funzionalità 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 utilizzi un'altra cache DNS, potrebbero verificarsi dei timeout di connessione.

  La `java.util.logging.Logger` classe può abilitare indirettamente la cache DNS JVM. Per sovrascrivere le impostazioni predefinite, imposta networkaddress.cache.ttl su 0 prima dell'inizializzazione. `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`networkaddress.cache.negative.ttl`, consigliamo di impostare su 0. `UnknownHostException` È possibile impostare questa proprietà per una funzione Lambda con la variabile di `AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0` ambiente.

La disabilitazione della cache DNS JVM non disabilita la cache DNS gestita da Lambda.

Configurazione della funzione

- La verifica delle prestazione 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 tua funzione è determinato per ogni richiamo e può essere visualizzato in Amazon. CloudWatch A ogni invocazione, verrà creata una voce `REPORT:`, come illustrato 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 configurazione di memoria giusta 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 della funzione, è inoltre consigliabile distribuire librerie in grado di sfruttare Advanced Vector Extensions 2 (AVX2). Ciò consente di elaborare carichi di lavoro complessi, tra cui l'inferenza di machine learning, l'elaborazione di elementi multimediali, l'elaborazione HPC (High Performance Computing), le simulazioni scientifiche e la modellazione finanziaria. Per ulteriori informazioni, vedere Creazione di AWS Lambda funzioni più veloci con AVX2.
- Eseguire il test della funzione Lambda per determinare un valore di timeout ottimale. È importante analizzare il tempo di esecuzione della funzione in modo da individuare meglio gli eventuali problemi con un servizio di dipendenza che possa aumentare la simultaneità della funzione oltre le previsioni. Ciò risulta particolarmente utile quando la funzione effettua chiamate di rete a risorse che non sono in grado di gestire il dimensionamento di Lambda.

- 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 tempo di esecuzione.
- 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 vale sia per che. CreateFunctionUpdateFunctionConfiguration
  - Nel caso di CreateFunction, AWS Lambda fallirà il processo di creazione della funzione.
  - Nel caso di UpdateFunctionConfiguration, potrebbe comportare invocazioni duplicate della funzione.

Scalabilità delle funzioni

- Informati sui vincoli di velocità di trasmissione effettiva a monte e a valle. Sebbene le funzioni Lambda si dimensioni perfettamente in base al carico, le dipendenze a monte e a valle potrebbero non avere le stesse capacità di velocità di trasmissione effettiva. Se devi limitare la scalabilità della tua funzione, puoi configurare la concorrenza riservata sulla tua funzione.
- Integra la tolleranza all'acceleratore. 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 all'acceleratore:
  - Usa 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 concorrenza fornita. La concorrenza fornita è il numero di ambienti di esecuzione preinizializzati che Lambda alloca alla tua funzione. Lambda gestisce le richieste in arrivo utilizzando la concorrenza fornita, se disponibile. Lambda può anche scalare la funzione oltre l'impostazione di concorrenza fornita, se necessario. La configurazione della concorrenza fornita comporta costi aggiuntivi per l'account. AWS

Parametri e allarmi

- Usa Utilizzo dei parametri delle funzioni Lambda and CloudWatch Alarms invece di creare o aggiornare una metrica dal 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. È possibile ad esempio configurare un allarme in base alla durata prevista dell'esecuzione della funzione Lambda per individuare colli di bottiglia o latenze attribuibili al codice della funzione.
- Utilizzare la libreria di registrazione e parametri e dimensioni AWS Lambda per rilevare errori di applicazione (ad esempio, ERR, ERROR, WARNING e così via)
- 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 di AWS Cost Explorer, che hanno 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 chiamata. 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**
>
> Le mappature delle sorgenti degli eventi 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 ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

- Aumentare il throughput di elaborazione del flusso Kinesis tramite l'aggiunta di shard. Un flusso Kinesis è costituito da uno o più shard. Lambda esegue il polling su ogni shard con al massimo un'invocazione simultanea. Se ad esempio il flusso è composto da 100 shard attivi, saranno presenti al massimo 100 invocazioni di funzioni Lambda in esecuzione simultaneamente. L'aumento del numero di shard determina direttamente l'aumento del numero massimo di invocazioni di funzioni Lambda simultanee e può aumentare il throughput di elaborazione del flusso Kinesis. 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 IteratorAge per determinare se il tuo stream 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 Security Hub utilizza 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 per valutare le risorse Lambda, consulta AWS Lambda i controlli nella Guida per l' AWS Security Hub utente.
- Monitora i log delle attività di rete Lambda utilizzando Amazon GuardDuty Lambda Protection. 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 legate alle criptovalute. GuardDuty monitora i registri delle attività di rete generati quando viene richiamata una funzione Lambda. Per ulteriori informazioni, consulta la protezione Lambda nella Amazon GuardDuty User Guide.

Gestione delle autorizzazioni in AWS Lambda

Puoi usare 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 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

Le funzioni Lambda spesso devono accedere ad altre AWS risorse ed eseguire varie operazioni API su tali risorse. Ad esempio, potresti avere 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 le informazioni dalle fonti degli eventi. Ogni funzione Lambda deve avere un ruolo di esecuzione. Come minimo, il ruolo di esecuzione deve avere accesso ad Amazon CloudWatch perché le funzioni Lambda accedono ai CloudWatch log per impostazione predefinita. Puoi collegare la policy `AWSLambdaBasicExecutionRole` gestita al tuo ruolo di esecuzione per soddisfare questo requisito.

Per concedere ad altri AWS account, organizzazioni e servizi le autorizzazioni per accedere alle tue risorse Lambda, hai a disposizione alcune opzioni:

- Puoi utilizzare policy basate sull'identità per concedere ad altri utenti l'accesso alle tue 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 AWS servizi le 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 di controllo degli accessi basato sugli attributi (ABAC) per controllare l'accesso alle tue funzioni Lambda. Con ABAC, puoi allegare tag a una funzione Lambda, passarli in determinate richieste API o collegarli al principale IAM che effettua la richiesta. Specificate gli stessi tag nell'elemento condition di una policy IAM per controllare l'accesso alla funzione.

In AWS, è consigliabile concedere solo le autorizzazioni necessarie per eseguire un'attività (autorizzazioni con privilegi minimi). Per implementarlo in Lambda, consigliamo di iniziare con una policy AWS gestita. 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 politiche. 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 il permesso di accedere a AWS servizi e risorse. Ad esempio, potresti creare un ruolo di esecuzione con l'autorizzazione a inviare log ad Amazon CloudWatch e caricare AWS X-Ray dati di traccia su. 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 manualmente `sts:AssumeRole` 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 assuma correttamente il tuo ruolo di esecuzione, la policy di fiducia del ruolo deve specificare il servizio Lambda principal (`lambda.amazonaws.com`) come servizio affidabile.

Argomenti

- Creazione di un ruolo di esecuzione nella console di IAM
- Creazione e gestione di 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 della funzione di origine ARN 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 fornisce alla tua funzione le autorizzazioni di base per registrare gli eventi su Amazon CloudWatch Logs.

Le tue funzioni in genere richiedono autorizzazioni aggiuntive per eseguire attività più significative. Ad esempio, potresti avere una funzione Lambda che risponde a un evento aggiornando le voci in un

database Amazon DynamoDB. Puoi creare un ruolo di esecuzione con le autorizzazioni necessarie utilizzando la console IAM.

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. Seleziona Successivo.
6. Seleziona le politiche AWS gestite che desideri associare al tuo ruolo. Ad esempio, se la tua funzione deve accedere a DynamoDB, seleziona AWSLambdaDynamoDBExecutionRolela policy gestita.
7. Seleziona Successivo.
8. Immettere un Role name (Nome ruolo), quindi selezionare Create role (Crea ruolo).

Per istruzioni dettagliate, consulta Creating a role for an AWS service (console) nella IAM User Guide.

Dopo aver creato il ruolo di esecuzione, associalo alla tua funzione. Quando crei una funzione nella console Lambda, puoi associare qualsiasi ruolo di esecuzione creato in precedenza alla funzione. Se desideri associare un nuovo ruolo di esecuzione a una funzione esistente, segui i passaggi indicati in.

Creazione e gestione di ruoli con AWS CLI

Per creare un ruolo di esecuzione con AWS Command Line Interface (AWS CLI), utilizzate 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. I requisiti per l'escape delle virgolette nella stringa JSON variano in base alla 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

```
{
  "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
```

Verrà visualizzato l'output seguente:

```
{
    "Role": {
        "Path": "/",
        "RoleName": "lambda-ex",
        "RoleId": "AROAQFOXMPL6TZ6ITKWND",
        "Arn": "arn:aws:iam::123456789012:role/lambda-ex",
        "CreateDate": "2020-01-17T23:19:12Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lambda.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

Utilizza il comando attach-policy-to-role per aggiungere le autorizzazioni al ruolo. Il comando seguente aggiunge la politica `AWSLambdaBasicExecutionRole` gestita al ruolo di `lambda-ex` esecuzione.

```
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, associalo alla funzione. Quando crei una funzione nella console Lambda, puoi associare qualsiasi ruolo di esecuzione creato in precedenza alla funzione. Se desideri associare un nuovo ruolo di esecuzione a una funzione esistente, segui i passaggi indicati in.

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 Applicazione delle autorizzazioni del privilegio minimo nella Guida per l'utente di IAM.

Utilizzare IAM Access Analyzer per identificare le autorizzazioni necessarie per la policy del ruolo di esecuzione IAM. IAM Access Analyzer esamina i tuoi 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 Generazione di policy basate sull'attività di accesso nella Guida per l'utente di IAM.

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, è possibile 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 dall'elenco a discesa per visualizzare tutte le autorizzazioni relative a quel 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 tua funzione, esegui anche un semplice aggiornamento del codice o della 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, puoi utilizzare la console Lambda.

Per aggiornare 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, scegli Modifica.
5. Se desideri aggiornare la tua funzione per utilizzare un ruolo diverso come ruolo di esecuzione, scegli il nuovo ruolo nel menu a discesa in 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 politiche che desideri allegare al nuovo ruolo in Modelli di policy.

6. Selezionare 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
AWSLambdaMSKExecutionRole— Lambda ha aggiunto l'autorizzazione kafka: DescribeCluster V2 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 rete elastiche (ENI) e scrivere nei log. CloudWatch	17 giugno 2022
AWSLambdaBasicExecutionRole— Lambda ha iniziato a tracciare le modifiche a questa politica.	`AWSLambdaBasicExecutionRole` concede le autorizzazioni per caricare i log su CloudWatch.	14 febbraio 2022

Modifica	Descrizione	Data
AWSLambdaDynamoDBExecutionRole— Lambda ha iniziato a tracciare le 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
AWSLambdaKinesisExecutionRole— 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
AWSLambdaMSKExecutionRole— Lambda ha iniziato a tracciare le 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 rete elastiche (ENI) e scrivere nei log. CloudWatch	14 febbraio 2022
AWSLambdaSQSQueueExecutionRole— 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

Modifica	Descrizione	Data
AWSLambdaVPCAccessExecutionRole— Lambda ha iniziato a tracciare le modifiche a questa politica.	`AWSLambdaVPCAccessExecutionRole` concede le autorizzazioni per gestire ENI all'interno di un Amazon VPC e scrivere su Logs. CloudWatch	14 febbraio 2022
AWSXRayDaemonWriteAccess— 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
CloudWatchLambdaInsightsExecutionRolePolicy— 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
Politica AmazonS3 ObjectLambdaExecutionRole: 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 AWS Security Token Service.

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 piano, 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 della funzione di origine ARN per controllare il comportamento di accesso alla funzione

È normale che il codice della funzione Lambda effettui richieste API ad altri AWS servizi. 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 gli SDK AWS , 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` , `CreateLogStream` , `PutLogEvents`	Per archiviare i log in un gruppo di CloudWatch log Logs

Servizio	Azione	Motivo
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 Amazon Elastic Compute Cloud (Amazon EC2) per creare interfacce di rete elastiche (ENI) per una funzione che supporta il VPC.
- Chiamate a AWS servizi, come Amazon Simple Queue Service (Amazon SQS), da leggere 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, utilizza la chiave di `lambda:SourceFunctionArn` condizione in una policy basata sull'identità IAM o in una policy di controllo dei servizi (SCP).

> **Note**
>
> Non puoi utilizzare la chiave di condizione `lambda:SourceFunctionArn` nelle policy basate sulle risorse.

Con questa chiave di condizione nelle policy basate sull'identità o negli SCP, puoi 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 altri AWS servizi e 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

```
{
    "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.

È possibile utilizzarla anche in SCP. `lambda:SourceFunctionArn` 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

```
{
    "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 ad altri AWS account, organizzazioni e servizi le autorizzazioni per accedere alle tue risorse Lambda, hai a disposizione alcune opzioni:

- Puoi utilizzare policy basate sull'identità per concedere ad altri utenti l'accesso alle tue 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 AWS servizi le 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 di controllo degli accessi basato sugli attributi (ABAC) per controllare l'accesso alle tue funzioni Lambda. Con ABAC, puoi allegare tag a una funzione Lambda, passarli in determinate richieste API o collegarli al principale IAM che effettua la richiesta. Specificate gli stessi tag nell'elemento condition di una policy IAM per controllare l'accesso alla funzione.

Per aiutarti a ottimizzare le autorizzazioni per l'accesso con privilegi minimi, Lambda fornisce alcune condizioni aggiuntive che puoi includere nelle tue politiche. Per ulteriori informazioni, consulta the section called “Risorse e condizioni”.

Utilizzo di politiche IAM basate sull'identità in Lambda

Puoi utilizzare le policy basate sull'identità in AWS Identity and Access Management (IAM) per concedere agli utenti del tuo account l'accesso a Lambda. Le policy basate su identità possono essere applicate direttamente agli utenti o ai gruppi e ruoli associati a un utente. 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. Questa pagina mostra un esempio di come le policy basate sull'identità possono essere utilizzate per lo sviluppo di funzioni.

Lambda fornisce policy AWS gestite che garantiscono 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.

- AWSLambda_FullAccess— Garantisce l'accesso completo alle azioni Lambda e AWS ad altri servizi utilizzati per sviluppare e gestire le risorse Lambda. Questa politica è stata creata analizzando la politica precedente. AWSLambdaFullAccess
- AWSLambda_ReadOnlyAccess— Garantisce l'accesso in sola lettura alle risorse Lambda. Questa politica è stata creata analizzando la politica precedente. AWSLambdaReadOnlyAccess
- AWSLambdaRole— Concede le autorizzazioni per richiamare le funzioni Lambda.

AWS le politiche 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.

Sections

Scrivere un esempio di policy che conceda le autorizzazioni utente a una funzione

Utilizza le policy basate sull'identità per consentire agli utenti 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 nell'Amazon Elastic Container Registry. Per un esempio, vedere Autorizzazioni 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

```
{
    "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": {
                "StringLike": {
                    "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` – Utilizzare 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`— Utilizzate 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. AWSLambdaFullAccess Alcuni esempi di policy sono disponibili nell'GitHubarchivio di questa guida.

Scrittura di una politica di esempio che conceda le autorizzazioni per l'uso dei livelli

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 AWS regione e con qualsiasi versione del livello, purché il nome del layer inizi con. `test-`

Example Policy per lo sviluppo dei livelli

```
{
    "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`.

```
{
    "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.

Implementazione dell'accesso tra account con politiche basate sull'identità

È possibile applicare una delle policy e dichiarazioni precedenti a un ruolo, che è quindi possibile condividere con un altro account per accedere alle risorse Lambda. A differenza di un utente, un ruolo non dispone di credenziali per l'autenticazione. Al contrario, ha una policy di attendibilità che specifica chi può assumere il ruolo e utilizzare le autorizzazioni.

È possibile utilizzare ruoli tra più account per dimostrare agli account che si dispone di un accesso attendibile a risorse e operazioni Lambda. Se si desidera solo concedere le autorizzazioni per richiamare una funzione o utilizzare un livello, utilizzare le policy basate sulle risorse.

Per ulteriori informazioni, consulta Ruoli IAM nella Guida per l'utente IAM.

Utilizzo di politiche basate sulle risorse in Lambda

Lambda supporta le policy di autorizzazioni basate sulle risorse per le funzioni e i livelli Lambda. Le politiche basate sulle risorse consentono di concedere autorizzazioni di utilizzo ad altri account o organizzazioni in base alla risorsa. AWS Utilizzi anche una politica basata sulle risorse per consentire a un AWS servizio di richiamare la tua funzione per tuo conto.

Per le funzioni Lambda, è possibile concedere un'autorizzazione dell'account per invocare o gestire una funzione. Puoi inoltre utilizzare una singola policy basata su risorse per concedere le autorizzazioni a un'intera organizzazione in AWS Organizations. Puoi anche utilizzare politiche basate sulle risorse per concedere l'autorizzazione di invocazione a un AWS servizio che richiama una funzione in risposta all'attività del tuo account.

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. Scorrere verso il basso fino a Policy basata su risorse, quindi scegliere View Policy Document (Visualizza documento policy). La politica basata sulle risorse mostra le autorizzazioni applicate

quando un altro account o servizio tenta di accedere alla funzione. AWS L'esempio seguente mostra un'istruzione che consente ad Amazon S3 di richiamare una funzione denominata `my-function` per un bucket denominato `DOC-EXAMPLE-BUCKET` nell'account `123456789012`.

Example Policy basata su risorse

```
{
    "Version": "2012-10-17",
    "Id": "default",
    "Statement": [
        {
            "Sid": "lambda-allow-s3-my-function",
            "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:::DOC-EXAMPLE-BUCKET"
              }
            }
        }
     ]
}
```

Per i livelli Lambda, è possibile utilizzare una policy basata sulle risorse solo su una versione del livello specifica anziché sull'intero livello. Oltre alle policy che concedono l'autorizzazione a un singolo account o a più account, per i livelli è anche possibile concedere autorizzazioni a tutti gli account in un'organizzazione.

> **Note**
>
> Puoi aggiornare le politiche basate sulle risorse solo per le risorse Lambda nell'ambito delle azioni e dell'API. AddPermissionAddLayerVersionPermission Attualmente, non è possibile

creare policy per le proprie risorse Lambda in formato JSON o utilizzare condizioni che non sono associate a parametri per tali operazioni.

Le policy basate sulle risorse si applicano a una singola funzione, versione, alias o versione del livello. Concedono le autorizzazioni a uno o più servizi e account. Per gli account attendibili con accesso a più risorse o per l'utilizzo di operazioni API che le policy basate sulle risorse non supportano, è possibile utilizzare ruoli tra più account.

Argomenti

Operazioni API supportate

Le seguenti operazioni dell'API Lambda supportano le policy basate sulle risorse:

- CreateAlias
- DeleteAlias
- DeleteFunction
- DeleteFunctionConcorrenza
- DeleteFunctionEventInvokeConfig
- DeleteProvisionedConcurrencyConfig
- GetAlias
- GetFunction
- GetFunctionConcorrenza
- GetFunctionConfigurazione
- GetFunctionEventInvokeConfig
- GetPolicy

- GetProvisionedConcurrencyConfig
- Invoke
- ListAliases
- ListFunctionEventInvokeConfigurazioni
- ListProvisionedConcurrencyConfigs
- ListTags
- ListVersionsByFunction
- PublishVersion
- PutFunctionConcorrenza
- PutFunctionEventInvokeConfig
- PutProvisionedConcurrencyConfig
- TagResource
- UntagResource
- UpdateAlias
- UpdateFunctionCodice
- UpdateFunctionEventInvokeConfig

Concessione dell'accesso alle funzioni ai servizi AWS

Quando si utilizza un AWS servizio per richiamare la propria funzione, si concede l'autorizzazione in una dichiarazione su una politica basata sulle risorse. È possibile applicare l'istruzione all'intera funzione da richiamare o gestire 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.

Aggiungere una dichiarazione con il comando `add-permission`. La dichiarazione più semplice della policy basata sulla risorsa consente a un servizio di invocare una funzione. Il comando seguente concede ad Amazon SNS l'autorizzazione di invocare 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"}
```

In questo modo Amazon SNS richiama l'API `lambda:Invoke` per la funzione, ma non limita l'argomento Amazon SNS che attiva la chiamata. Per garantire che la tua funzione venga richiamata solo da una risorsa specifica, specificare l'Amazon Resource Name (ARN) della risorsa con l'opzione `source-arn`. Il comando seguente consente solo a Amazon SNS di richiamare la funzione per gli abbonamenti a un argomento denominato `my-topic`.

```
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
```

Alcuni servizi possono invocare le funzioni in altri account. Se si specifica un ARN origine che ha il proprio ID account, non è un problema. Per Amazon S3, tuttavia, l'origine è un bucket il cui ARN non dispone di un ID account. È possibile eliminare il bucket e un altro account può creare un bucket con lo stesso nome. Utilizza l'opzione `source-account` con il tuo ID account per avere la certezza che solo le risorse nel proprio account possano richiamare la funzione.

```
aws lambda add-permission --function-name my-function --action lambda:InvokeFunction --
statement-id s3-account \
    --principal s3.amazonaws.com --source-arn arn:aws:s3:::DOC-EXAMPLE-BUCKET --source-
account 123456789012
```

Concedere l'accesso alle funzioni a un'organizzazione

Per concedere le autorizzazioni a un'organizzazione in AWS Organizations, specifica l'ID dell'organizzazione come. `principal-org-id` Il AddPermission AWS CLI comando seguente concede l'accesso alle chiamate a tutti gli utenti dell'organizzazione. `o-a1b2c3d4e5f`

```
aws lambda add-permission --function-name example \
```

```
    --statement-id PrincipalOrgIDExample --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 AWS account o un ruolo come`Principal`, solo tale preside ottiene i permessi di invocazione delle funzioni, ma solo se fa anche parte dell'organizzazione. `o-a1b2c3d4e5f`

Questo comando crea una policy basata sulle risorse simile alla seguente:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PrincipalOrgIDExample",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:example",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-a1b2c3d4e5f"
                }
            }
        }
    ]
}
```

Per ulteriori informazioni, consulta aws: PrincipalOrg ID nella guida per l' AWS Identity and Access Management utente.

Concessione dell'accesso alle funzioni ad altri account

Per concedere le autorizzazioni a un altro AWS account, specifica l'ID dell'account come. `principal` L'esempio seguente concede l'autorizzazione `111122223333` a un account per invocare `my-function` con l'alias `prod`.

```
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:l
east-2:123456789012:function:my-function"}
```

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.

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`.

L'alias limita quale versione l'altro account può invocare. Richiede che l'altro account includa l'alias nell'ARN della funzione.

```
aws lambda invoke --function-name arn:aws:lambda:us-west-2:123456789012:function:my-
function:prod out
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "1"
}
```

Il proprietario della funzione può quindi aggiornare l'alias per puntare a una nuova versione senza che il chiamante debba cambiare il modo in cui invoca la funzione. In questo modo l'altro account non deve modificare il codice per utilizzare la nuova versione e dispone solo dell'autorizzazione per richiamare la versione della funzione associata all'alias.

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.

Aggiungere ogni autorizzazione separatamente oppure utilizzare `lambda:*` per concedere l'accesso a tutte le azioni per la funzione specificata.

Per concedere ad altri account l'autorizzazione per più funzioni o per operazioni che non operano su una funzione, utilizzare ruoli IAM.

Concessione dell'accesso ai livelli ad altri account

Per concedere l'autorizzazione per l'utilizzo del livello a un altro account, aggiungi un'istruzione alla policy di autorizzazione della versione del livello con il comando add-layer-version-permission. In ogni istruzione, puoi concedere l'autorizzazione a un singolo account, a tutti gli account o a un'organizzazione.

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 --statement-id
 xaccount \
--action lambda:GetLayerVersion --principal 111122223333 --version-number 2 --output
 text
```

Verrà visualizzato un output simile al seguente:

```
e210ffdc-e901-43b0-824b-5fcd0dd26d16    {"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.

Per concedere le autorizzazioni a tutti gli account in un'organizzazione, è possibile utilizzare l'opzione `organization-id`. L'esempio seguente concede a tutti gli account dell'organizzazione l'autorizzazione all'utilizzo della versione 3 di un livello.

```
aws lambda add-layer-version-permission --layer-name my-layer \
    --statement-id engineering-org --version-number 3 --principal '*' \
    --action lambda:GetLayerVersion --organization-id o-t194hfs8cz --output text
```

Verrà visualizzato l'output seguente:

```
b0cd9796-d4eb-4564-939f-de7fe0b42236    {"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 tutti AWS gli account, utilizza * for the principal e ometti l'ID dell'organizzazione. Per più account o organizzazioni, devi aggiungere più istruzioni.

Pulizia delle policy basate sulle risorse

Per visualizzare una policy basata sulle risorse di una funzione, utilizzare il comando `get-policy`.

```
aws lambda get-policy --function-name my-function --output text
```

Verrà visualizzato l'output seguente:

```
{"Version":"2012-10-17","Id":"default","Statement":
[{"Sid":"sns","Effect":"Allow","Principal":
{"Service":"s3.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:lambda*"}}}]}      7c681fc9-
b791-4e91-acdf-eb847fdaa0f0
```

Per le versioni e gli alias, aggiungere il numero di versione o un alias al nome della funzione.

```
aws lambda get-policy --function-name my-function:PROD
```

Per rimuovere le autorizzazioni dalla funzione, utilizzare `remove-permission`.

```
aws lambda remove-permission --function-name example --statement-id sns
```

Utilizzare il comando `get-layer-version-policy` per visualizzare le autorizzazioni su un layer.

```
aws lambda get-layer-version-policy --layer-name my-layer --version-number 3 --output
 text
```

Verrà visualizzato l'output seguente:

```
b0cd9796-d4eb-4564-939f-de7fe0b42236    {"Sid":"engineering-
org","Effect":"Allow","Principal":"*","Action":"lambda:GetLayerVersion","Resource":"arn:aws:lam
```

```
west-2:123456789012:layer:my-layer:3","Condition":{"StringEquals":
{"aws:PrincipalOrgID":"o-t194hfs8cz"}}}"
```

Utilizzare `remove-layer-version-permission` per rimuovere le istruzioni dalla policy.

```
aws lambda remove-layer-version-permission --layer-name my-layer --version-number 3 --
statement-id engineering-org
```

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 funzioni Lambda. Puoi allegare tag a una funzione Lambda, passarli in determinate richieste API o collegarli 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 Controlling access to AWS resources using tags 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. Il dimensionamento è più facile con ABAC perché non è necessario aggiornare le policy IAM quando si creano nuove funzioni. Invece, per controllare l'accesso aggiungi tag alle nuove funzioni.

In Lambda, i tag funzionano a livello di funzione. I tag non sono supportati per livelli, configurazioni di firma del codice o mappature di origini eventi. Quando aggiungi un tag a una funzione, tali tag si applicano a tutte le versioni e agli alias associati alla funzione. Per ulteriori informazioni su come aggiungere un tag alle funzioni, consulta Uso dei tag sulle funzioni Lambda.

Puoi utilizzare le seguenti chiavi di condizione per controllare le operazioni delle funzioni:

- aws: ResourceTag /tag-key: controlla l'accesso in base ai tag allegati alle funzioni 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 associati al suo utente o ruolo IAM.
- aws:TagKeys: Controlla se è possibile utilizzare chiavi di tag specifiche in una richiesta.

Per l'elenco completo delle operazioni Lambda a supporto di ABAC, consulta Azioni funzionali supportate e controlla la colonna Condition (Condizione) nella tabella.

La seguente procedura mostra un modo per impostare le autorizzazioni 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

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 aws: RequestTag /tag-key, aws: ResourceTag /tag-key e aws: TagKeys condition per richiedere che le nuove funzioni e il principale IAM che crea le funzioni abbiano entrambi il tag. `project` 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

```
{
```

```
  "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 condizione aws: ResourceTag /tag-key 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

```
{
  "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

```
{
  "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

```
{
  "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

```
{
  "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.

Questa strategia di autorizzazione consente di controllare l'accesso senza creare nuove policy 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.

Passaggio 8: Pulisci le tue risorse

Per eliminare il ruolo IAM

1. Aprire la pagina Ruoli della console IAM.
2. Seleziona il ruolo che hai creato nel passaggio 5.
3. Scegli Elimina.
4. Per confermare l'eliminazione, inserisci il nome del ruolo nel campo di immissione del testo.
5. Scegli Elimina.

Per eliminare l'utente IAM

1. Apri la pagina Utenti della console IAM.
2. Seleziona l'utente IAM che hai creato nel passaggio 6.
3. Scegli 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 **`delete`** nel campo di immissione testo e scegli Delete (Elimina).

Ottimizzazione delle 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 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`
- 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`

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

```
{
    "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

Comprensione della sezione Condizione nelle politiche

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 del Servizio di notifica semplice Amazon (Amazon SNS) per richiamare una funzione denominata `test`.

Example Gestione delle autorizzazioni di una policy della funzione

```
{
    "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 sulle risorse e le 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 politiche

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 API Lambda, gli utenti possono specificare una versione o un alias passando una versione ARN o un alias ARN nel parametro o impostando un valore GetFunction`FunctionName`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

```
{
    "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

```
{
    "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

```
{
    "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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction*"
        }
    ]
}
```

Azioni funzionali supportate

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. Le operazioni che non supportano limitazioni di risorse sono concesse a tutte le risorse (*).

Azioni funzione

Azione	Risorsa	Condizione
AddPermission RemovePermission	Funzione Versione funzione Alias funzione	lambda:Principal aws:ResourceTag/${TagKey} lambda:FunctionUrlAuthType
Invoke Autorizzazione: lambda:InvokeFunction	Funzione Versione funzione Alias funzione	aws:ResourceTag/${TagKey} lambda:EventSourceToken
CreateFunction	Funzione	lambda:CodeSigningConfigArn lambda:Layer

Azione	Risorsa	Condizione

		`lambda:VpcIds`
		`lambda:SubnetIds`
		`lambda:SecurityGroupIds`
		`aws:ResourceTag/${TagKey}`
		`aws:RequestTag/${TagKey}`
		`aws:TagKeys`

UpdateFunctionConfigurazione	Funzione	`lambda:CodeSigning ConfigArn`
		`lambda:Layer`
		`lambda:VpcIds`
		`lambda:SubnetIds`
		`lambda:SecurityGroupIds`
		`aws:ResourceTag/${TagKey}`

Azione	Risorsa	Condizione
CreateAlias	Funzione	`aws:ResourceTag/${TagKey}`
DeleteAlias		
DeleteFunction		
DeleteFunctionCodeSigningConfig		
DeleteFunctionConcorrenza		
GetAlias		
GetFunction		
GetFunctionCodeSigningConfig		
GetFunctionConcorrenza		
GetFunctionConfigurazione		
GetPolicy		
ListProvisionedConcurrencyConfigs		
ListAliases		
ListTags		
ListVersionsByFunction		
PublishVersion		
PutFunctionCodeSigningConfig		
PutFunctionConcorrenza		
UpdateAlias		
UpdateFunctionCodice		

Azione	Risorsa	Condizione
CreateFunctionUrlConfig DeleteFunctionUrlConfig GetFunctionUrlConfig UpdateFunctionUrlConfig	Funzione Alias funzione	lambda:FunctionUrlAuthType lambda:FunctionArn aws:ResourceTag/${TagKey}
ListFunctionUrlConfigs	Funzione	lambda:FunctionUrlAuthType
DeleteFunctionEventInvokeConfig GetFunctionEventInvokeConfig ListFunctionEventInvokeConfigurazioni PutFunctionEventInvokeConfig UpdateFunctionEventInvokeConfig	Funzione	aws:ResourceTag/${TagKey}
DeleteProvisionedConcurrencyConfig GetProvisionedConcurrencyConfig PutProvisionedConcurrencyConfig	Alias funzione Versione funzione	aws:ResourceTag/${TagKey}
GetAccountImpostazioni ListFunctions	*	Nessuno
TagResource	Funzione	aws:ResourceTag/${TagKey} aws:RequestTag/${TagKey} aws:TagKeys

Azione	Risorsa	Condizione
UntagResource	Funzione	`aws:ResourceTag/${TagKey}` `aws:TagKeys`

Azioni di mappatura delle sorgenti degli eventi supportate

Per le mappature delle origini eventi, puoi limitare le autorizzazioni di eliminazione e aggiornamento a un'origine eventi specifica. La condizione `lambda:FunctionArn` consente di limitare le funzioni che un utente può configurare per richiamare un origine eventi.

Per tali operazioni, la risorsa è la mappatura dell'origine evento, quindi Lambda fornisce una condizione che consente di limitare l'autorizzazione in base alla funzione che la mappatura dell'origine evento richiama.

Operazioni della mappatura dell'origine eventi

Azione	Risorsa	Condizione
DeleteEventSourceMapping UpdateEventSourceMapping	Mappatura delle origini eventi	`lambda:FunctionArn`
CreateEventSourceMapping GetEventSourceMapping	*	`lambda:FunctionArn`
ListEventSourceMappings	*	Nessuno

Azioni di livello supportate

Le operazioni relative ai livelli consentono di limitare i livelli che un utente può gestire o utilizzare con una funzione. Le operazioni che riguardano l'utilizzo e le autorizzazioni relative ai livelli agiscono su una versione di un livello, mentre `PublishLayerVersion` agisce su un nome di livello. Entrambe possono essere utilizzate con i caratteri jolly per limitare i livelli che un utente può adottare in base al nome.

> **Note**
>
> L'azione GetLayerVersione copre anche GetLayerVersionByArn. Lambda non supporta `GetLayerVersionByArn` come azione IAM.

Azioni livello

Azione	Risorsa	Condizione
AddLayerVersionPermission RemoveLayerVersionPermission GetLayerVersione GetLayerVersionPolicy DeleteLayerversione	Versione livello	Nessuno
ListLayerVersioni PublishLayerversione	Livello	Nessuno
ListLayers	*	Nessuno

Sicurezza in AWS Lambda

Per AWS, la sicurezza del cloud ha la massima priorità. In quanto cliente AWS, è possibile trarre vantaggio da un'architettura di data center e di rete progettata per soddisfare i requisiti delle organizzazioni più esigenti a livello di sicurezza.

La sicurezza è una responsabilità condivisa tra te e AWS. Il modello di responsabilità condivisa descrive questo come sicurezza del cloud e sicurezza nel cloud:

- La sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce i servizi AWS nel cloud AWS. AWS fornisce inoltre servizi che puoi utilizzare in sicurezza. I revisori di terze parti testano e verificano regolarmente l'efficacia della sicurezza come parte dei programmi di conformità AWS. Per ulteriori informazioni sui programmi di conformità che si applicano a AWS Lambda, consulta Servizi coperti dal programma di conformitàAWS.
- Sicurezza nel cloud: la tua responsabilità è determinata dal servizio AWS che utilizzi. Sei anche responsabile di altri fattori, tra cui la riservatezza dei dati, i requisiti della tua azienda e le leggi e normative vigenti.

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. Scoprirai anche come utilizzare altri servizi di AWS per monitorare e proteggere le risorse Lambda.

Per maggiori informazioni sull'applicazione dei principi di sicurezza alle applicazioni Lambda, consulta Sicurezza in Serverless Land.

Argomenti

Protezione dei dati in AWS Lambda

Il modello di responsabilità AWS condivisa modello di 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. Inoltre, sei responsabile della configurazione della protezione e delle attività di gestione per i Servizi AWS che utilizzi. Per ulteriori informazioni sulla privacy dei dati, vedi Domande frequenti sulla privacy dei dati. Per informazioni sulla protezione dei dati in Europa, consulta il post del blog AWS Shared Responsibility Model and GDPR nel Blog sulla sicurezza AWS .

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Ti suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- Usa SSL/TLS per comunicare con le risorse. AWS È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti con. AWS CloudTrail
- 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 140-2 per l'accesso AWS tramite un'interfaccia a riga di comando o un'API, utilizza un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il Federal Information Processing Standard (FIPS) 140-2.

Ti consigliamo vivamente 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 i la fatturazione o i log di diagnostica. Quando fornisci un URL a un server esterno, ti suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la tua richiesta al server.

Sections

- Crittografia a riposo

Crittografia in transito

Gli endpoint API di Lambda supportano solo connessioni protette tramite HTTPS. Quando gestisci le risorse Lambda con AWS Management Console,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 a riposo

Lambda crittografa sempre le variabili di ambiente inattive. Per impostazione predefinita, Lambda utilizza un file creato AWS KMS key da Lambda nell'account per crittografare le variabili di ambiente. Questo è denominato Chiave gestita da AWS . `aws/lambda`

Per ogni singola funzione, è possibile configurare Lambda affinché utilizzi una chiave gestita dal cliente anziché la Chiave gestita da AWS predefinita per crittografare le variabili d'ambiente. Per ulteriori informazioni, consulta Protezione delle variabili di ambiente Lambda.

Lambda crittografa sempre i file caricati su Lambda, inclusi i pacchetti di implementazione e gli archivi dei livelli.

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 Crittografa i dati di registro in CloudWatch Logs e Data protection in. AWS X-Ray

Identity and Access Management per AWS Lambda

AWS Identity and Access Management (IAM) è un Servizio AWSche consente agli amministratori di controllare in modo sicuro l'accesso alle risorse AWS. Gli amministratori IAM controllano chi può

essere autenticato (connesso) e autorizzato (avere le autorizzazioni) a utilizzare le risorse Lambda. IAM è un Servizio AWS il cui uso non comporta costi aggiuntivi.

Argomenti

Destinatari

Le modalità di utilizzo di AWS Identity and Access Management (IAM) cambiano in base alle operazioni eseguite in Lambda.

Utente del servizio: se utilizzi il servizio Lambda per svolgere il proprio lavoro, l'amministratore ti fornisce le credenziali e le autorizzazioni necessarie. Man mano che si utilizzano più funzionalità Lambda per svolgere il proprio lavoro, potresti necessitare di autorizzazioni aggiuntive. La comprensione della gestione dell'accesso consente di richiedere le autorizzazioni corrette all'amministratore. Se non riesci ad accedere a una funzionalità di Lambda, consultare Risoluzione dei problemi di identità e accesso in AWS Lambda.

Amministratore del servizio: se si è responsabile delle risorse Lambda presso la tua azienda, probabilmente hai pieno accesso a Lambda. Il tuo compito è determinare le funzioni Lambda e le risorse a cui gli utenti del servizio devono accedere. Devi inviare le richieste all'amministratore IAM per cambiare le autorizzazioni degli utenti del servizio. Esamina le informazioni contenute in questa pagina per comprendere i concetti di base relativi a IAM. Per ulteriori informazioni su come la tua azienda può utilizzare IAM con Lambda, consultare Funzionamento di AWS Lambdacon IAM.

Amministratore IAM: se sei un amministratore IAM, potresti voler conoscere i dettagli su come scrivere le policy di gestione dell'accesso a Lambda. Per visualizzare le policy Lambda di esempio basate sull'identità che è possibile utilizzare in IAM, consultare Esempi di policy basate su identità per AWS Lambda.

Autenticazione con identità

L'autenticazione è la procedura di accesso ad AWS con le credenziali di identità. Devi essere autenticato (connesso a AWS) come utente root Utente root dell'account AWS, come utente IAM o assumere un ruolo IAM.

Puoi accedere ad AWS come identità federata utilizzando le credenziali fornite attraverso un'origine di identità. Gli utenti AWS IAM Identity Center (Centro identità IAM), l'autenticazione Single Sign-On (SSO) dell'azienda e le credenziali di Google o Facebook sono esempi di identità federate. Se accedi come identità federata, l'amministratore ha configurato in precedenza la federazione delle identità utilizzando i ruoli IAM. Se accedi ad AWS tramite la federazione, assumi indirettamente un ruolo.

A seconda del tipo di utente, puoi accedere alla AWS Management Console o al portale di accesso AWS. Per ulteriori informazioni sull'accesso ad AWS, consulta la sezione Come accedere al tuo Account AWS nella Guida per l'utente di Accedi ad AWS.

Se accedi ad AWS in modo programmatico, AWS fornisce un Software Development Kit (SDK) e un'interfaccia a riga di comando (CLI) per firmare crittograficamente le richieste utilizzando le tue credenziali. Se non utilizzi gli strumenti AWS, devi firmare le richieste personalmente. Per ulteriori informazioni sulla firma delle richieste, consulta Firma delle richieste AWS nella Guida per l'utente IAM.

A prescindere dal metodo di autenticazione utilizzato, potrebbe essere necessario specificare ulteriori informazioni sulla sicurezza. AWS consiglia ad esempio di utilizzare l'autenticazione a più fattori (MFA) per aumentare la sicurezza dell'account. Per ulteriori informazioni, consulta Autenticazione a più fattori nella Guida per l'utente di AWS IAM Identity Center e Utilizzo dell'autenticazione a più fattori (MFA) in AWS nella Guida per l'utente di IAM.

Utente root di un Account AWS

Quando crei un Account AWS, inizi con una singola identità di accesso che ha accesso completo a tutti i Servizi AWS e le risorse nell'account. Tale identità è detta utente root Account AWS ed è possibile accedervi con l'indirizzo e-mail e la password utilizzati per creare l'account. Si consiglia vivamente di non utilizzare l'utente root per le attività quotidiane. Conserva le credenziali dell'utente root e utilizzarle per eseguire le operazioni che solo l'utente root può eseguire. Per un elenco completo delle attività che richiedono l'accesso come utente root, consulta la sezione Attività che richiedono le credenziali dell'utente root nella Guida per l'utente di IAM.

Identità federata

Come best practice, richiedi agli utenti umani, compresi quelli che richiedono l'accesso di amministratore, di utilizzare la federazione con un provider di identità per accedere a Servizi AWSutilizzando credenziali temporanee.

Un'identità federata è un utente della directory degli utenti aziendali, un provider di identità Web, AWS Directory Service, la directory Identity Center o qualsiasi utente che accede ai Servizi AWS utilizzando le credenziali fornite tramite un'origine di identità. Quando le identità federate accedono agli Account AWS, assumono ruoli e i ruoli forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, consigliamo di utilizzare AWS IAM Identity Center. È possibile creare utenti e gruppi in IAM Identity Center oppure connettersi e sincronizzarsi con un gruppo di utenti e gruppi nell'origine di identità per utilizzarli in tutte le applicazioni e gli Account AWS. Per ulteriori informazioni sul Centro identità IAM, consulta Cos'è Centro identità IAM? nella Guida per l'utente di AWS IAM Identity Center.

Utenti e gruppi IAM

Un utente IAM è una identità all'interno del tuo Account AWS che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ove possibile, consigliamo di fare affidamento a credenziali temporanee invece di creare utenti IAM con credenziali a lungo termine come le password e le chiavi di accesso. Tuttavia, per casi d'uso specifici che richiedono credenziali a lungo termine con utenti IAM, si consiglia di ruotare le chiavi di accesso. Per ulteriori informazioni, consulta la pagina Rotazione periodica delle chiavi di accesso per casi d'uso che richiedono credenziali a lungo termine nella Guida per l'utente di IAM.

Un gruppo IAM è un'identità che specifica un insieme di utenti IAM. Non è possibile eseguire l'accesso come gruppo. È possibile utilizzare gruppi per specificare le autorizzazioni per più utenti alla volta. I gruppi semplificano la gestione delle autorizzazioni per set di utenti di grandi dimensioni. Ad esempio, è possibile avere un gruppo denominato Amministratori IAM e concedere a tale gruppo le autorizzazioni per amministrare le risorse IAM.

Gli utenti sono diversi dai ruoli. Un utente è associato in modo univoco a una persona o un'applicazione, mentre un ruolo è destinato a essere assunto da chiunque ne abbia bisogno. Gli utenti dispongono di credenziali a lungo termine permanenti, mentre i ruoli forniscono credenziali temporanee. Per ulteriori informazioni, consulta Quando creare un utente IAM (invece di un ruolo) nella Guida per l'utente di IAM.

Ruoli IAM

Un ruolo IAM è un'identità all'interno di un Account AWS che dispone di autorizzazioni specifiche. È simile a un utente IAM, ma non è associato a una persona specifica. È possibile assumere temporaneamente un ruolo IAM nella AWS Management Console mediante lo scambio di ruoli. È possibile assumere un ruolo chiamando un'azione AWS CLI o API AWS oppure utilizzando un URL personalizzato. Per ulteriori informazioni sui metodi per l'utilizzo dei ruoli, consulta Utilizzo di ruoli IAM nella Guida per l'utente di IAM.

I ruoli IAM con credenziali temporanee sono utili nelle seguenti situazioni:

- Accesso utente federato: per assegnare le autorizzazioni a una identità federata, è possibile creare un ruolo e definire le autorizzazioni per il ruolo. Quando un'identità federata viene autenticata, l'identità viene associata al ruolo e ottiene le autorizzazioni da esso definite. Per ulteriori informazioni sulla federazione dei ruoli, consulta Creazione di un ruolo per un provider di identità di terza parte nella Guida per l'utente di IAM. Se utilizzi IAM Identity Center, configura un set di autorizzazioni. IAM Identity Center mette in correlazione il set di autorizzazioni con un ruolo in IAM per controllare a cosa possono accedere le identità dopo l'autenticazione. Per ulteriori informazioni sui set di autorizzazioni, consulta Set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center.
- Autorizzazioni utente IAM temporanee: un utente IAM o un ruolo può assumere un ruolo IAM per ottenere temporaneamente autorizzazioni diverse per un'attività specifica.
- Accesso multi-account: è possibile utilizzare un ruolo IAM per permettere a un utente (un principale affidabile) con un account diverso di accedere alle risorse nell'account. I ruoli sono lo strumento principale per concedere l'accesso multi-account. Tuttavia, per alcuni dei Servizi AWS, è possibile collegare una policy direttamente a una risorsa (anziché utilizzare un ruolo come proxy). Per informazioni sulle differenze tra ruoli e policy basate su risorse per l'accesso multi-account, consulta Differenza tra i ruoli IAM e le policy basate su risorse nella Guida per l'utente di IAM.
- Accesso multi-servizio: alcuni Servizi AWS utilizzano funzionalità in altri Servizi AWS. Ad esempio, quando effettui una chiamata in un servizio, è comune che tale servizio esegua applicazioni in Amazon EC2 o archivi oggetti in Amazon S3. Un servizio può eseguire questa operazione utilizzando le autorizzazioni dell'entità chiamante, utilizzando un ruolo di servizio o utilizzando un ruolo collegato al servizio.
  - Inoltro delle sessioni di accesso (FAS): quando si utilizza un utente o un ruolo IAM per eseguire operazioni in AWS, tale utente o ruolo viene considerato un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra azione in un servizio diverso. FAS utilizza le autorizzazioni del principale che effettua la chiamata a un Servizio

AWS, combinate con il Servizio AWS richiedente, per effettuare richieste a servizi a valle. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che necessita di interazioni con altri Servizi AWS o risorse per essere portata a termine. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le operazioni. Per i dettagli delle policy relative alle richieste FAS, consulta la pagina Forward access sessions.

- Ruolo di servizio: un ruolo di servizio è un ruolo IAM assunto da un servizio per eseguire operazioni per conto dell'utente. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione Creazione di un ruolo per delegare le autorizzazioni a un Servizio AWS nella Guida per l'utente di IAM.
- Ruolo collegato al servizio: 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'azione per tuo conto. I ruoli collegati ai servizi sono visualizzati nell'account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati ai servizi, ma non modificarle.

- Applicazioni in esecuzione su Amazon EC2: è possibile utilizzare un ruolo IAM per gestire credenziali temporanee per le applicazioni in esecuzione su un'istanza EC2 che eseguono richieste di AWS CLIo dell'API AWS. Ciò è preferibile all'archiviazione delle chiavi di accesso nell'istanza EC2. Per assegnare un ruolo AWS a un'istanza EC2, affinché sia disponibile per tutte le relative applicazioni, puoi creare un profilo dell'istanza collegato all'istanza. Un profilo dell'istanza contiene il ruolo e consente ai programmi in esecuzione sull'istanza EC2 di ottenere le credenziali temporanee. Per ulteriori informazioni, consulta Utilizzo di un ruolo IAM per concedere autorizzazioni ad applicazioni in esecuzione su istanze di Amazon EC2 nella Guida per l'utente di IAM.

Per informazioni sull'utilizzo dei ruoli IAM, consulta Quando creare un ruolo IAM (invece di un utente) nella Guida per l'utente di IAM.

Gestione dell'accesso con policy

Per controllare l'accesso a AWS è possibile creare policy e collegarle a identità o risorse AWS. Una policy è un oggetto in AWS che, quando associato a un'identità o a una risorsa, ne definisce le autorizzazioni. AWS valuta queste policy quando un principale IAM (utente, utente root o sessione ruolo) effettua una richiesta. Le autorizzazioni nelle policy determinano l'approvazione o il rifiuto della richiesta. La maggior parte delle policy viene archiviata in AWSsotto forma di documenti JSON. Per ulteriori informazioni sulla struttura e sui contenuti dei documenti delle policy JSON, consulta Panoramica delle policy JSON nella Guida per l'utente di IAM.

Gli amministratori possono utilizzare le policy AWSJSON per specificare l'accesso ai diversi elementi. In altre parole, quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Per concedere agli utenti l'autorizzazione a eseguire azioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. Successivamente l'amministratore può aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

Le policy IAM definiscono le autorizzazioni relative a un'operazione, a prescindere dal metodo utilizzato per eseguirla. Ad esempio, supponiamo di disporre di una policy che consente l'azione `iam:GetRole`. Un utente con tale policy può ottenere informazioni sul ruolo dalla AWS Management Console, la AWS CLI o l'API AWS.

Policy basate su identità

Le policy basate su 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 azioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta Creazione di policy IAM nella Guida per l'utente di IAM.

Le policy basate su identità possono essere ulteriormente classificate come policy inline o policy gestite. Le policy inline sono incorporate direttamente in un singolo utente, gruppo o ruolo. Le policy gestite sono policy autonome che possono essere collegate a più utenti, gruppi e ruoli in Account AWS. Le policy gestite includono le policy gestite da AWS e le policy gestite dal cliente. Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta Scelta fra policy gestite e policy inline nella Guida per l'utente di IAM.

Policy basate su risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile allegare a una risorsa. Gli esempi più comuni di policy basate su risorse sono le policy di attendibilità dei ruoli IAM e le policy dei bucket Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarle per controllare l'accesso a una risorsa specifica. Quando è allegata a una risorsa, una policy definisce le azioni che un principale può eseguire su tale risorsa e a quali condizioni. È necessario specificare un principale in una policy basata sulle risorse. I principali possono includere account, utenti, ruoli, utenti federati o Servizi AWS.

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non è possibile utilizzare le policy gestite da AWS da IAM in una policy basata su risorse.

Liste di controllo degli accessi (ACL)

Le liste di controllo degli accessi (ACL) controllano quali principali (membri, utenti o ruoli dell'account) hanno le autorizzazioni per accedere a una risorsa. Le ACL sono simili alle policy basate sulle risorse, sebbene non utilizzino il formato del documento di policy JSON.

Amazon S3, AWS WAF e Amazon VPC sono esempi di servizi che supportano le ACL. Per maggiori informazioni sulle ACL, consulta Panoramica delle liste di controllo degli accessi (ACL) nella Guida per gli sviluppatori di Amazon Simple Storage Service.

Altri tipi di policy

AWS supporta altri tipi di policy meno comuni. Questi tipi di policy possono impostare il numero massimo di autorizzazioni concesse dai tipi di policy più comuni.

- Limiti delle autorizzazioni: un limite delle autorizzazioni è una funzione avanzata nella quale si imposta il numero massimo di autorizzazioni che una policy basata su identità può concedere a un'entità IAM (utente o ruolo IAM). È possibile impostare un limite delle autorizzazioni per un'entità. Le autorizzazioni risultanti sono l'intersezione delle policy basate su identità dell'entità e i relativi limiti delle autorizzazioni. Le policy basate su risorse che specificano l'utente o il ruolo nel campo `Principal` sono condizionate dal limite delle autorizzazioni. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni sui limiti delle autorizzazioni, consulta Limiti delle autorizzazioni per le entità IAM nella Guida per l'utente di IAM.
- Policy di controllo dei servizi (SCP): le SCP sono policy JSON che specificano il numero massimo di autorizzazioni per un'organizzazione o unità organizzativa (OU) in AWS Organizations. AWS Organizationsè un servizio per il raggruppamento e la gestione centralizzata degli Account AWSmultipli di proprietà dell'azienda. Se abiliti tutte le funzionalità in un'organizzazione, puoi applicare le policy di controllo dei servizi (SCP) a uno o tutti i tuoi account. La SCP limita le autorizzazioni per le entità negli account membri, compreso ogni Utente root dell'account AWS. Per ulteriori informazioni su organizzazioni e policy SCP, consulta la pagina sulle Policy di controllo dei servizi nella Guida per l'utente di AWS Organizations.
- Policy di sessione: le policy di sessione sono policy avanzate che vengono trasmesse come parametro quando si crea in modo programmatico una sessione temporanea per un ruolo o un utente federato. Le autorizzazioni della sessione risultante sono l'intersezione delle policy basate su identità del ruolo o dell'utente e le policy di sessione. Le autorizzazioni possono anche provenire da una policy basata su risorse. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni, consulta Policy di sessione nella Guida per l'utente di IAM.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, le autorizzazioni risultanti sono più complicate da comprendere. Per informazioni su come AWSdetermina se consentire una richiesta quando sono coinvolti più tipi di policy, consultare Logica di valutazione delle policy nella Guida per l'utente di IAM.

Funzionamento di AWS Lambdacon IAM

Prima di utilizzare IAM per gestire l'accesso a Lambda, scopri quali funzionalità IAM sono disponibili per l'uso con Lambda.

Funzionalità IAM che è possibile utilizzare con AWS Lambda

Funzionalità IAM	Supporto Lambda
Policy basate su identità	Sì
Policy basate su risorse	Sì
Azioni di policy	Sì
Risorse relative alle policy	Sì
Chiavi di condizione della policy (specifica del servizio)	Sì
Liste di controllo degli accessi (ACL)	No
ABAC (tag nelle policy)	Parziale
Credenziali temporanee	Sì
Inoltro delle sessioni di accesso (FAS)	No
Ruoli di servizio	Sì
Ruoli collegati al servizio	Parziale

Per avere una visione di alto livello di come Lambda e AWS altri servizi funzionano con la maggior parte delle funzionalità IAM, AWSconsulta i servizi che funzionano con IAM nella IAM User Guide.

Politiche basate sull'identità per Lambda

Supporta le policy basate su identità	Sì

Le policy basate su 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 azioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta Creazione di policy IAM nella Guida per l'utente di IAM.

Con le policy basate su 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. Non è possibile specificare l'entità principale in una policy basata sull'identità perché si applica all'utente o al ruolo a cui è associato. 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 di IAM.

Esempi di policy basate sull'identità per Lambda

Per visualizzare esempi di policy basate sull'identità Lambda, vedere. Esempi di policy basate su identità per AWS Lambda

Politiche basate sulle risorse all'interno di Lambda

Supporta le policy basate su risorse	Sì

Le policy basate su risorse sono documenti di policy JSON che è possibile allegare a una risorsa. Gli esempi più comuni di policy basate su risorse sono le policy di attendibilità dei ruoli IAM e le policy dei bucket Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarle per controllare l'accesso a una risorsa specifica. Quando è allegata a una risorsa, una policy definisce le azioni che un principale può eseguire su tale risorsa e a quali condizioni. È necessario specificare un principale in una policy basata sulle risorse. I principali possono includere account, utenti, ruoli, utenti federati o Servizi AWS.

Per consentire l'accesso multi-account, puoi specificare un intero account o entità IAM in un altro account come principale in una policy basata sulle risorse. L'aggiunta di un principale multi-account a una policy basata sulle risorse rappresenta solo una parte della relazione di trust. Quando l'entità

principale e la risorsa si trovano in diversi Account AWS, un amministratore IAM nell'account attendibile deve concedere all'entità principale (utente o ruolo) anche l'autorizzazione per accedere alla risorsa. L'autorizzazione viene concessa collegando all'entità una policy basata sull'identità. Tuttavia, se una policy basata su risorse concede l'accesso a un principale nello stesso account, non sono richieste ulteriori policy basate su identità. Per ulteriori informazioni, consulta Differenza tra i ruoli IAM e le policy basate su risorse nella Guida per l'utente di IAM.

È possibile allegare una policy basata sulle risorse a una funzione o a un layer Lambda. Questa policy definisce quali principali possono eseguire azioni sulla funzione o sul livello.

Per informazioni su come collegare una policy basata sulle risorse a una funzione o a un livello, consulta. Utilizzo di politiche basate sulle risorse in Lambda

Azioni politiche per Lambda

Supporta le azioni di policy	Sì

Gli amministratori possono utilizzare le policy JSON AWS per specificare gli accessi ai diversi elementi. Cioè, quale principale può eseguire azioni 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 a un criterio. Le azioni di policy hanno spesso lo stesso nome dell'operazione API AWS. Ci sono alcune eccezioni, ad esempio le azioni di sola autorizzazione che non hanno un'operazione API corrispondente. Esistono anche alcune operazioni che richiedono più operazioni in una policy. Queste operazioni aggiuntive sono denominate operazioni dipendenti.

Includi le operazioni in una policy per concedere le autorizzazioni a eseguire l'operazione associata.

Per visualizzare un elenco di azioni Lambda, consulta Actions defined by AWS Lambda nel Service Authorization Reference.

Le azioni politiche in Lambda utilizzano il seguente prefisso prima dell'azione:

```
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 sull'identità Lambda, vedere. Esempi di policy basate su identità per AWS Lambda

Risorse politiche per Lambda

Supporta le risorse di policy	Sì

Gli amministratori possono utilizzare le policy JSON AWS per specificare gli accessi ai diversi elementi. Cioè, quale 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'azione. Le istruzioni devono includere un elemento `Resource` o un elemento `NotResource`. Come best practice, specifica una risorsa utilizzando il suo nome della risorsa Amazon (ARN). Puoi eseguire questa operazione per azioni che supportano un tipo di risorsa specifico, note come autorizzazioni a livello di risorsa.

Per le azioni che non supportano le autorizzazioni a livello di risorsa, ad esempio le operazioni di elenco, 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 dei relativi ARN, 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 sull'identità Lambda, vedere. Esempi di policy basate su identità per AWS Lambda

Chiavi relative alle condizioni delle politiche per Lambda

Supporta le chiavi di condizione delle policy specifiche del servizio	Sì

Gli amministratori possono utilizzare le policy JSON AWS per specificare gli accessi ai diversi elementi. Cioè, quale principale può eseguire azioni su quali risorse, e in quali condizioni.

L'elemento `Condition` (o blocco `Condition`) consente di specificare le condizioni in cui un'istruzione è in vigore. L'elemento `Condition` è facoltativo. Puoi 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.

Se specifichi più elementi `Condition` in un'istruzione o più chiavi in un singolo elemento `Condition`, questi vengono valutati da AWS utilizzando un'operazione `AND` logica. Se specifichi più valori per una singola chiave di condizione, AWS valuta la condizione utilizzando un'operazione `OR` logica. Tutte le condizioni devono essere soddisfatte prima che le autorizzazioni dell'istruzione vengano concesse.

Puoi anche utilizzare variabili segnaposto quando specifichi le condizioni. Ad esempio, puoi autorizzare un utente IAM ad accedere a una risorsa solo se è stata taggata con il relativo nome utente IAM. Per ulteriori informazioni, consulta Elementi delle policy IAM: variabili e tag nella Guida per l'utente di IAM.

AWS supporta chiavi di condizione globali e chiavi di condizione specifiche per il servizio. Per visualizzare tutte le chiavi di condizione globali di AWS, consulta Chiavi di contesto delle condizioni globali di AWS nella Guida per l'utente di IAM.

Per visualizzare un elenco di chiavi di condizione Lambda, consulta Condition keys for AWS Lambda nel Service Authorization Reference. Per informazioni su operazioni e risorse con cui è possibile utilizzare una chiave di condizione, consulta la sezione Operazioni definite da AWS Lambda.

Per visualizzare esempi di policy basate sull'identità Lambda, vedere. Esempi di policy basate su identità per AWS Lambda

ACL in Lambda

Supporta le ACL	No

Le liste di controllo degli accessi (ACL) controllano quali principali (membri, utenti o ruoli dell'account) hanno le autorizzazioni ad accedere a una risorsa. Le ACL sono simili alle policy basate su risorse, sebbene non utilizzino il formato del documento di policy JSON.

ABAC con Lambda

Supporta ABAC (tag nelle policy)	Parziale

Il controllo dell'accesso basato su attributi (ABAC) è una strategia di autorizzazione che definisce le autorizzazioni in base agli attributi. In AWS, tali attributi sono denominati tag. È possibile collegare dei tag alle entità IAM (utenti o ruoli) e a numerose risorse AWS. L'assegnazione di tag alle entità e alle risorse è il primo passaggio di ABAC. In seguito, vengono progettate policy ABAC per consentire operazioni quando il tag dell'entità principale corrisponde al tag sulla risorsa a cui si sta provando ad accedere.

La strategia ABAC è utile in ambienti soggetti a una rapida crescita e aiuta in situazioni in cui la gestione delle policy diventa impegnativa.

Per controllare l'accesso basato su tag, fornisci 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 Yes (Sì). Se un servizio supporta tutte e tre le chiavi di condizione solo per alcuni tipi di risorsa, allora il valore sarà Parziale.

Per ulteriori informazioni su ABAC, consulta Che cos'è 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 sull'etichettatura 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

Alcuni Servizi AWS non funzionano quando si accede utilizzando credenziali temporanee. Per ulteriori informazioni, inclusi i Servizi AWS che funzionano con le credenziali temporanee, consulta Servizi AWS supportati da IAM nella Guida per l'utente IAM.

Le credenziali temporanee sono utilizzate se si accede alla AWS Management Console utilizzando qualsiasi metodo che non sia la combinazione di nome utente e password. Ad esempio, quando accedi ad AWS utilizzando il collegamento Single Sign-On (SSO) della tua azienda, tale processo crea in automatico credenziali temporanee. Le credenziali temporanee vengono create in automatico anche quando accedi alla console come utente e poi cambi ruolo. Per ulteriori informazioni sullo scambio dei ruoli, consulta Cambio di un ruolo (console) nella Guida per l'utente di IAM.

È possibile creare manualmente credenziali temporanee utilizzando la AWS CLIo l'API AWS. È quindi possibile utilizzare tali credenziali temporanee per accedere ad AWS. AWSconsiglia di generare le credenziali temporanee dinamicamente anziché utilizzare chiavi di accesso a lungo termine. Per ulteriori informazioni, consulta Credenziali di sicurezza provvisorie in IAM.

Sessioni di accesso diretto per Lambda

Supports forward access sessions (FAS)	No

Quando si utilizza un utente o un ruolo IAM per eseguire operazioni in AWS, si viene considerati un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'azione che attiva un'altra azione in un servizio diverso. FAS utilizza le autorizzazioni del principale che effettua la chiamata a un Servizio AWS, combinate con il Servizio AWS richiedente, per effettuare richieste a servizi a valle. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che necessita di interazioni con altri Servizi AWS o risorse per essere portata a termine. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le operazioni. Per i dettagli delle policy relative alle richieste FAS, consulta la pagina 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 la sezione Creazione di un ruolo per delegare le autorizzazioni a un Servizio AWS nella Guida per l'utente di IAM.

In Lambda, un ruolo di servizio è noto come ruolo di esecuzione.

> **Warning**
>
> La modifica delle autorizzazioni per un ruolo di esecuzione potrebbe interrompere la funzionalità Lambda.

Ruoli collegati ai servizi per Lambda

Supporta i ruoli collegati ai servizi	Parziale

Un ruolo collegato ai servizi è un tipo di ruolo di servizio che è collegato a un Servizio AWS. Il servizio può assumere il ruolo per eseguire un'operazione per tuo conto. I ruoli collegati ai servizi sono visualizzati nell'account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati ai servizi, 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 su identità per AWS Lambda

Per impostazione predefinita, gli utenti e i ruoli non sono autorizzati a creare o modificare le risorse Lambda. Inoltre, non sono in grado di eseguire attività utilizzando la AWS Management Console, l'AWS Command Line Interface (AWS CLI) o l'API AWS. Per concedere agli utenti l'autorizzazione per eseguire operazioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. L'amministratore può quindi aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

Per informazioni su come creare una policy basata su identità IAM utilizzando questi documenti di policy JSON di esempio, consulta Creazione di policy IAM nella Guida per l'utente di IAM.

Per informazioni dettagliate sulle azioni e sui tipi di risorse definiti da Lambda, incluso il formato degli ARN 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 operazioni possono comportare costi aggiuntivi per l'Account AWS. Quando crei o modifichi policy basate su identità, segui queste linee guida e raccomandazioni:

- Nozioni di base sulle policy gestite da AWSe passaggio alle autorizzazioni con privilegio minimo: per le informazioni di base su come concedere autorizzazioni a utenti e carichi di lavoro, utilizza le policy gestite da AWSche concedono le autorizzazioni per molti casi d'uso comuni. Sono disponibili nel tuo Account AWS. Ti consigliamo pertanto di ridurre ulteriormente le autorizzazioni definendo policy gestite dal cliente di AWSspecifiche per i tuoi casi d'uso. Per ulteriori informazioni, consulta Policy gestite da AWS o Policy gestite da AWS per le funzioni dei processi nella Guida per l'utente IAM.
- Applica le autorizzazioni con privilegi minimi: quando imposti le autorizzazioni con le policy IAM, concedi solo le autorizzazioni richieste per eseguire un'attività. Puoi farlo definendo le azioni che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come autorizzazioni con privilegi minimi. Per ulteriori 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 a operazioni e risorse puoi aggiungere una condizione alle tue policy. Ad esempio, è possibile scrivere una condizione di policy per specificare che tutte le richieste devono essere inviate utilizzando SSL. Puoi inoltre utilizzare le condizioni per concedere l'accesso alle operazioni di servizio, ma solo se vengono utilizzate tramite uno specifico Servizio AWS, ad esempio AWS CloudFormation. Per ulteriori informazioni, consulta la sezione Elementi delle policy JSON di IAM: condizione nella Guida per l'utente di IAM.
- Utilizzo di IAM Access Analyzer per convalidare le policy IAM e garantire autorizzazioni sicure e funzionali: IAM Access Analyzer convalida le policy nuove ed esistenti in modo che aderiscano alla

sintassi della policy IAM (JSON) e alle best practice di IAM. IAM Access Analyzer offre oltre 100 controlli delle policy e consigli utili per creare policy sicure e funzionali. Per ulteriori informazioni, consulta Convalida delle policy per IAM Access Analyzer nella Guida per l'utente di IAM.

- Richiesta dell'autenticazione a più fattori (MFA): se hai uno scenario che richiede utenti IAM o utenti root nel tuo Account AWS, attiva MFA per una maggiore sicurezza. Per richiedere la MFA quando vengono chiamate le operazioni API, aggiungi le condizioni MFA alle policy. Per ulteriori informazioni, consulta Configurazione dell'accesso alle API protetto 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 console AWS Lambda è necessario disporre di un insieme di autorizzazioni minimo. 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 concedere le autorizzazioni minime della console agli utenti che effettuano chiamate solo alla AWS CLI o all'API AWS. Al contrario, concedi l'accesso solo alle operazioni 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 Scrivere un esempio di policy che conceda le autorizzazioni utente a una funzione. Oltre alle API Lambda, la console Lambda utilizza altri servizi per visualizzare la configurazione di attivazione e permettere l'aggiunta di 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 di altri servizi AWS.

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 allegate alla relativa identità utente. Questa policy include le autorizzazioni per completare questa operazione sulla console o a livello di codice utilizzando AWS CLI o l'API 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": "*"
        }
    ]
}
```

Policy gestite da AWS per AWS Lambda

Una policy gestita da AWSè una policy autonoma creata e amministrata da AWS. Le policy gestite da AWSsono progettate per fornire autorizzazioni per molti casi d'uso comuni in modo da poter iniziare ad assegnare autorizzazioni a utenti, gruppi e ruoli.

Ricorda che le policy gestite da AWSpotrebbero non concedere autorizzazioni con privilegi minimi per i tuoi casi d'uso specifici perché possono essere utilizzate da tutti i clienti AWS. Consigliamo pertanto

di ridurre ulteriormente le autorizzazioni definendo policy gestite dal cliente specifiche per i tuoi casi d'uso.

Non è possibile modificare le autorizzazioni definite nelle policy gestite da AWS. Se AWSaggiorna le autorizzazioni definite in una policy gestita da AWS, l'aggiornamento riguarda tutte le identità principali (utenti, gruppi e ruoli) a cui è collegata la policy. È molto probabile che AWSaggiorni una policy gestita da AWSquando viene lanciato un nuovo Servizio AWSo nuove operazioni API diventano disponibili per i servizi esistenti.

Per ulteriori informazioni, consultare Policy gestite da AWSnella Guida per l'utente di IAM.

Argomenti

- AWSpolitica gestita: AWSLambda_FullAccess
- AWSpolitica gestita: AWSLambda_ReadOnlyAccess
- AWSpolitica gestita: AWSLambdaBasicExecutionRole
- AWSpolitica gestita: AWSLambdaDynamoDBExecutionRole
- AWSpolitica gestita: AWSLambdaENIManagementAccess
- AWSpolitica gestita: AWSLambdaExecute
- AWSpolicy gestita: AWSLambdaInvocation -DynamoDB
- AWSpolitica gestita: AWSLambdaKinesisExecutionRole
- AWSpolitica gestita: AWSLambdaMSKExecutionRole
- AWSpolitica gestita: AWSLambdaRole
- AWSpolitica gestita: AWSLambdaSQSQueueExecutionRole
- AWSpolitica gestita: AWSLambdaVPCAccessExecutionRole
- Aggiornamenti di Lambda alle policy gestite da AWS

AWSpolitica gestita: AWSLambda_FullAccess

Questa policy concede l'accesso completo a tutte le operazioni di Lambda. Concede inoltre autorizzazioni ad altri AWSservizi utilizzati per sviluppare e gestire le risorse Lambda.

Puoi associare la policy `AWSLambda_FullAccess` ai tuoi utenti, gruppi e ruoli.

Dettagli dell'autorizzazione

Questa policy include le seguenti autorizzazioni:

- `lambda`: consente ai principali l'accesso completo a Lambda.
- `cloudformation`: consente ai principali di descrivere gli stack AWS CloudFormation ed elenca le risorse in tali stack.
- `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.
- `kms`: consente ai principali di elencare gli alias.
- `logs`— Consente ai responsabili di descrivere i gruppi di CloudWatch log di Amazon. Per i gruppi di log associati a una funzione Lambda, questa policy consente al principale di descrivere i flussi di log, ottenere log eventi e filtrare i log eventi.
- `states`: consente ai principali di descrivere ed elencare le macchine a stati di AWS Step Functions.
- `tag`: consente ai principali di ottenere risorse in base ai loro tag.
- `xray`: consente ai principali di ottenere riepiloghi delle tracce AWS X-Ray e recuperare un elenco di tracce specificato dall'ID.

Per ulteriori informazioni su questa policy, inclusi il documento sulla policy JSON e le versioni della policy, consulta AWSLambda_FullAccessla AWSManaged Policy Reference Guide.

AWSpolitica gestita: AWSLambda_ReadOnlyAccess

Questa policy garantisce l'accesso in sola lettura alle risorse Lambda e ad altri servizi AWS utilizzati per sviluppare e gestire le risorse Lambda.

Puoi associare la policy `AWSLambda_ReadOnlyAccess` ai tuoi utenti, gruppi e ruoli.

Dettagli dell'autorizzazione

Questa policy include le seguenti autorizzazioni:

- `lambda`: consente ai principali di ottenere ed elencare tutte le risorse.

- `cloudformation`: consente ai principali di descrivere ed elencare gli stack AWS CloudFormation ed elencare le risorse in tali stack.
- `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 gruppi di CloudWatch log di Amazon. Per i gruppi di log associati a una funzione Lambda, questa policy consente al principale di descrivere i flussi di log, ottenere log eventi e filtrare i log eventi.
- `states`: consente ai principali di descrivere ed elencare le macchine a stati di AWS Step Functions.
- `tag`: consente ai principali di ottenere risorse in base ai loro tag.
- `xray`: consente ai principali di ottenere riepiloghi delle tracce AWS X-Ray e recuperare un elenco di tracce specificato dall'ID.

Per ulteriori informazioni su questa policy, inclusi il documento sulla policy JSON e le versioni della policy, consulta AWSLambda_ReadOnlyAccessla AWSManaged Policy Reference Guide.

AWSpolitica 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 AWSLambdaBasicExecutionRoleGuide. AWS

AWSpolitica 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

AWSpolitica gestita: AWSLambdaENIManagementAccess

Questa policy concede le autorizzazioni per creare, descrivere ed eliminare interfacce di rete elastiche utilizzate da una funzione Lambda abilitata per VPC.

Puoi associare la policy `AWSLambdaENIManagementAccess` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento sulla policy JSON e le versioni delle policy, consulta AWSLambdaENIManagementAccessla AWSManaged Policy Reference Guide.

AWSpolitica gestita: AWSLambdaExecute

Questa politica garantisce `PUT` `GET` l'accesso ad Amazon Simple Storage Service e l'accesso completo ai CloudWatch log.

Puoi associare la policy `AWSLambdaExecute` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento e le versioni della policy JSON, consulta la AWSManaged Policy AWSLambdaExecuteReference Guide.

AWSpolicy 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 policy, incluso il documento sulla policy JSON e le versioni delle policy, consulta AWSLambdaInvocation-DynamoDB nella AWSManaged Policy Reference Guide.

AWSpolitica 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 AWSLambdaKinesisExecutionRoleGuide. AWS

AWSpolitica 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. AWSLambdaMSKExecutionRoleAWS

AWSpolitica 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 AWSLambdaRolela AWSManaged Policy Reference Guide.

AWSpolitica 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. AWSLambdaSQSQueueExecutionRoleAWS

AWSpolitica gestita: AWSLambdaVPCAccessExecutionRole

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 `AWSLambdaVPCAccessExecutionRole` 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 AWSLambdaVPCAccessExecutionRoleGuide. AWS

Aggiornamenti di Lambda alle policy gestite da AWS

Modifica	Descrizione	Data
AWSLambdaVPCAccess ExecutionRole— Modifica	Lambda ha aggiornato la `AWSLambdaVPCAccess`	5 gennaio 2024

Modifica	Descrizione	Data
	`ExecutionRole` policy per consentire l'azione. `ec2:DescribeSubnets`	
AWSLambda_ReadOnly Access— Cambia	Lambda ha aggiornato la policy `AWSLambda_ReadOnlyAccess` per consentire ai principali di elencare gli stack AWS CloudFormation.	27 luglio 2023
AWS Lambda ha iniziato il rilevamento delle modifiche	AWS Lambda ha iniziato il rilevamento delle modifiche per le relative policy gestite da AWS.	27 luglio 2023

Risoluzione dei problemi di identità e accesso in AWS Lambda

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`.

Per ulteriore assistenza con l'accesso, contatta l'amministratore AWS. 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 AWSconsentono di trasmettere un ruolo esistente a tale servizio, invece di creare un nuovo ruolo di servizio o un ruolo collegato ai servizi. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere 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 passare 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`.

Per ulteriore assistenza con l'accesso, contatta l'amministratore AWS. 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 servizi che supportano policy basate su risorse o liste di controllo accessi (ACL), utilizza tali policy per concedere alle persone l'accesso alle tue risorse.

Per ulteriori informazioni, consultare gli argomenti seguenti:

- Per sapere se Lambda supporta queste funzionalità, consultare Funzionamento di AWS Lambdacon IAM.
- Per informazioni su come garantire l'accesso alle risorse negli Account AWS che possiedi, consultare Fornire l'accesso a un utente IAM in un altro Account AWS che si possiede nella Guida per l'utente di IAM.
- Per informazioni su come fornire l'accesso alle risorse ad Account AWS di terze parti, consulta Fornire l'accesso agli 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 di IAM.
- Per informazioni sulle differenze tra l'utilizzo di ruoli e policy basate su risorse per l'accesso multi-account, consultare Differenza tra i ruoli IAM e le policy basate su risorse nella Guida per l'utente di IAM.

Crea una strategia di governance per le funzioni e i livelli Lambda

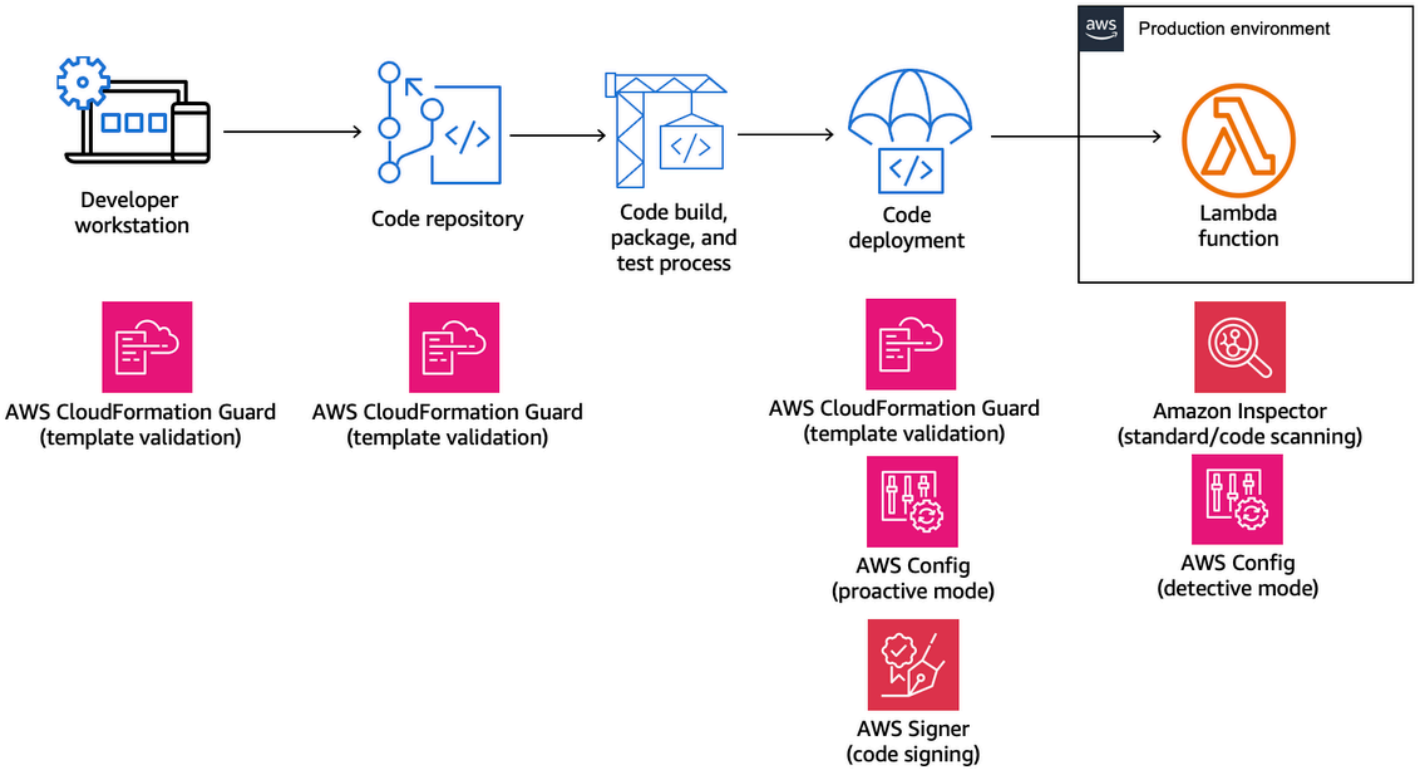
Per creare e implementare applicazioni serverless native del cloud, è necessario garantire agilità e velocità di immissione sul mercato con una governance e guardrail adeguati. Devi stabilire le priorità a livello aziendale, magari enfatizzando l'agilità come priorità assoluta o, in alternativa, sottolineando l'avversione al rischio attraverso governance, guardrail e controlli. Realisticamente, non adotterai una strategia "o/o", ma una strategia "e" che bilanci agilità e guardrail nel ciclo di vita dello sviluppo 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.

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

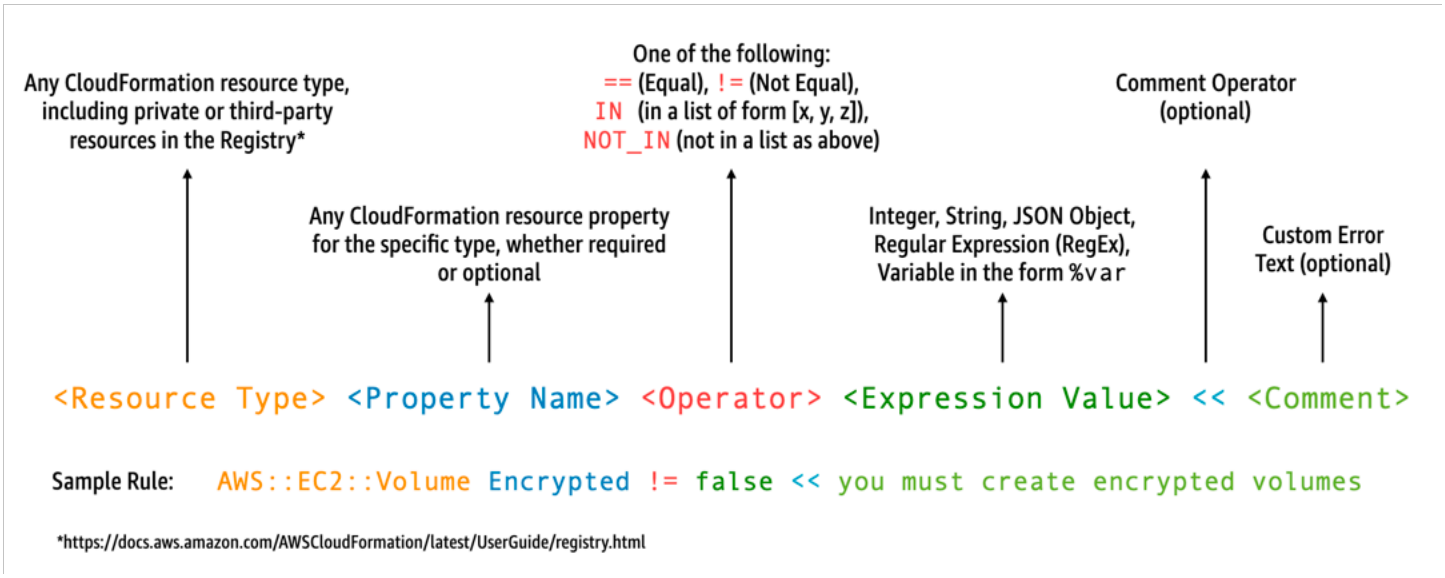
- Automatizza le valutazioni di sicurezza per Lambda con Amazon Inspector
- Implementazione dell'osservabilità per la sicurezza e la conformità Lambda

Controlli proattivi per Lambda con AWS CloudFormation Guard

AWS CloudFormation Guardè uno strumento di valutazione open source, policy-as-code 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 nelle pipeline CI/CD, 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 nelle postazioni di lavoro locali. Gli sviluppatori creano codice, convalidano i modelli con strumenti a riga di comando e quindi eseguono il commit del codice nei repository, che vengono poi convalidati in automatico attraverso le pipeline CI/CD prima dell'implementazione in un ambiente AWS.

Guard ti consente di scrivere le regole e di implementare i controlli con un linguaggio specifico per il dominio come riportato di seguito.



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

      }
    }
```

```
  }
}
```

Guard consente agli sviluppatori di vedere dalle loro postazioni di lavoro locali che devono aggiornare il modello per utilizzare un runtime consentito dall'organizzazione. Ciò avviene prima di effettuare il commit in un repository di codice e quindi 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 alla postazione di lavoro locale degli sviluppatori, a un webhook di convalida pre-commit e/o alla pipeline CI/CD prima dell'implementazione.

Avvertenze

Se utilizzi 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 `AWS::Serverless::Function` come riportato di seguito.

```
let lambda_functions = Resources.*[
    Type == "AWS::Serverless::Function"
]
```

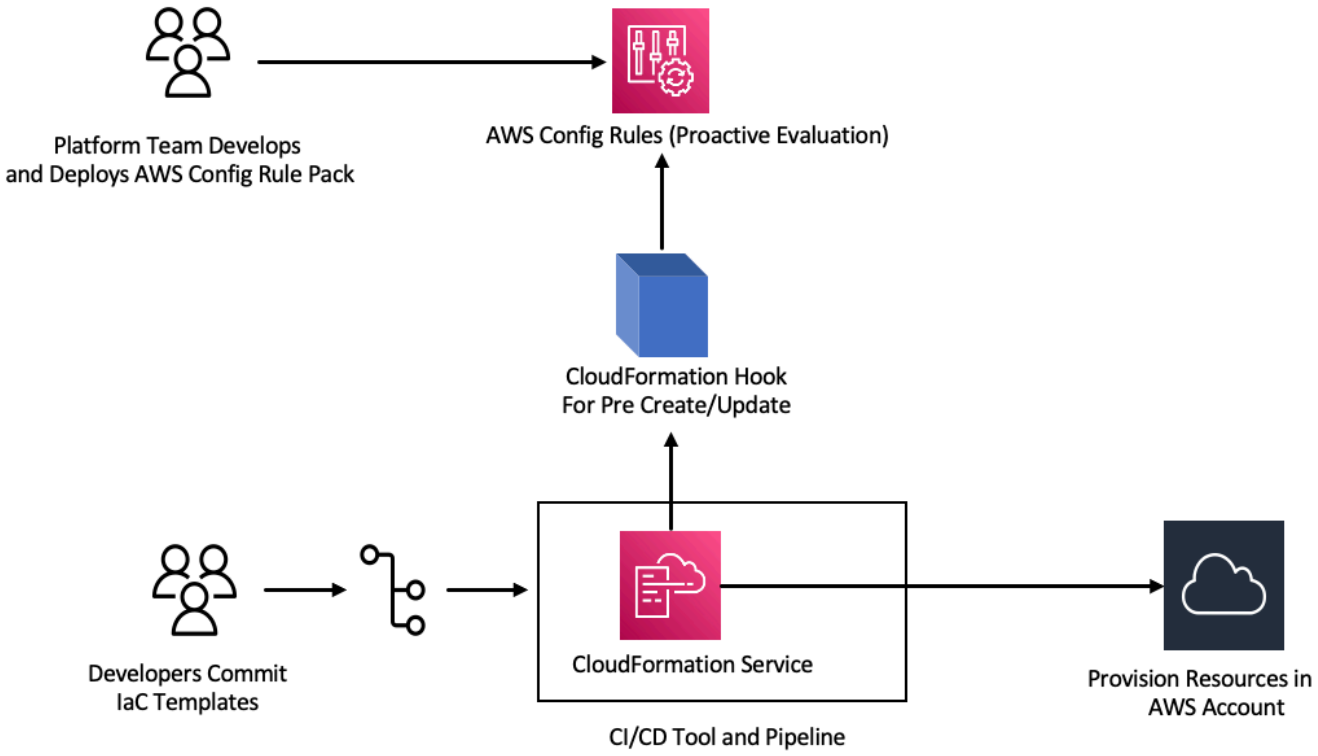
Guard si aspetta inoltre che le proprietà vengano incluse nella definizione della risorsa. Nel frattempo, i modelli AWS SAM consentono di specificare le proprietà in una sezione Globali separata. Le proprietà definite nella sezione Globali non vengono convalidate con le regole Guard.

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. Ciò ti consente di implementare i controlli di conformità nelle fasi iniziali del processo di sviluppo e di implementare i medesimi controlli nelle pipeline CI/CD. Ciò standardizza anche i controlli in un archivio di regole gestito centralmente in modo da poter applicare i controlli in modo uniforme 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 funzioni Lambda prima dell'implementazione, 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 di una AWS Config regola specifica da applicare in modo proattivo su tutti gli account. Questa regola contrassegna come risorsa non conforme ogni funzione Lambda priva di una configurazione di tracciamento X-Ray configurata. Il team sviluppa una regola, la racchiude in un pacchetto di conformità e distribuisce il pacchetto di conformità su tutti gli AWS account per garantire che tutti gli 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 implementazione richiami un hook di pre-creazione/aggiornamento. AWS CloudFormation 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, consulta Developing AWS CloudFormation Hooks nella documentazione dell'interfaccia a riga di CloudFormation comando (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
rpdk.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 di creare 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 `StartResourceEvaluation`API in modalità proattiva utilizzando il. AWS SDK for 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 delle AWS Lambda funzioni.

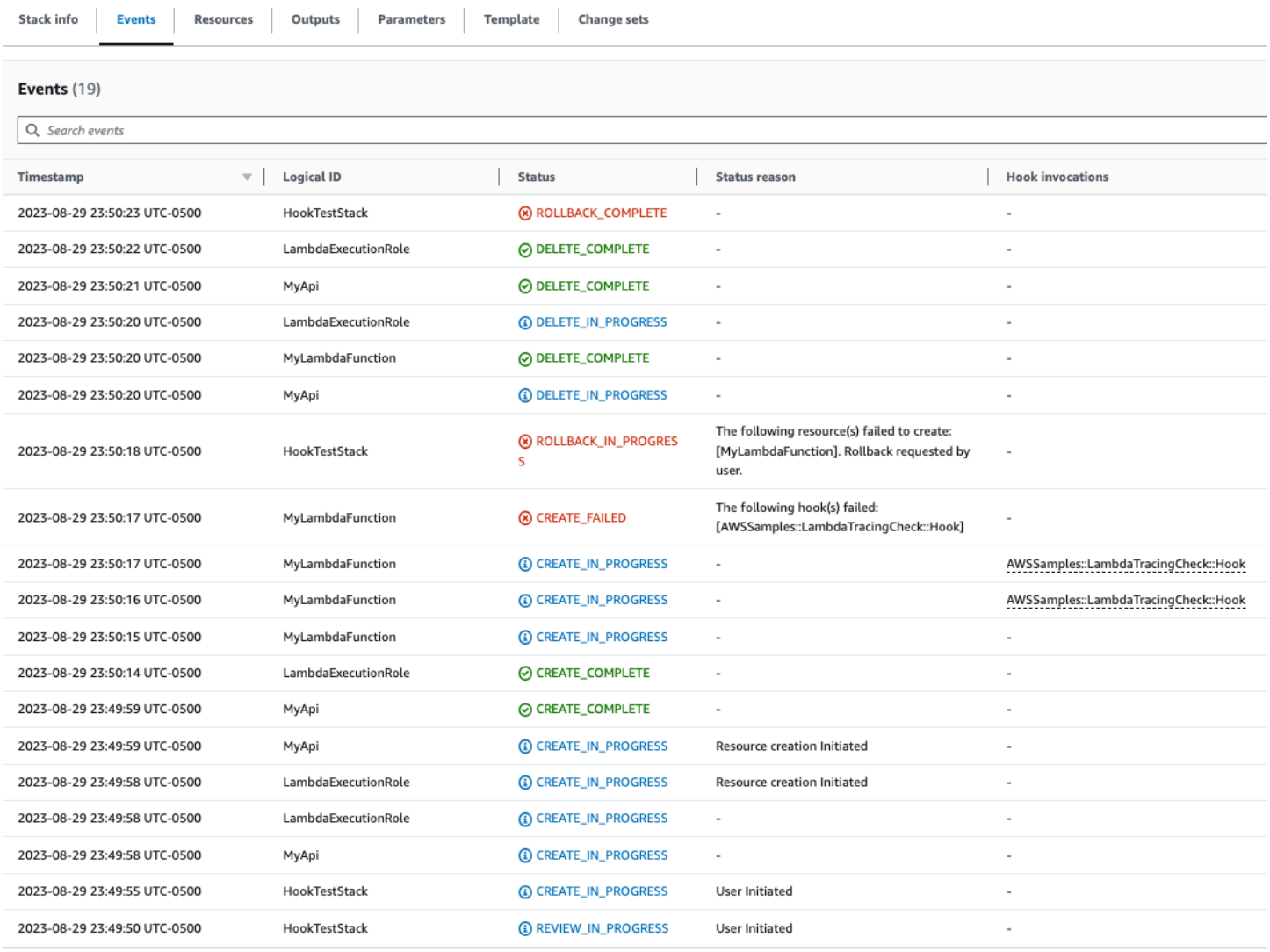
Uno sviluppatore prepara il modello di infrastructure as code (IaC) per un microservizio serverless utilizzando Lambda. La preparazione prevede il rispetto degli standard interni, a cui seguono il test e il commit locale del modello nel repository. Ecco un esempio di modello IaC:

```
  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: python3.8
    TracingConfig:
        Mode: PassThrough
    MemorySize: 256
```

```
    Timeout: 10
```

Come parte del processo CI/CD, quando il CloudFormation modello viene distribuito, il CloudFormation servizio richiama l'hook di pre-creazione/aggiornamento subito prima del provisioning del 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 procede alla fornitura delle risorse. CloudFormation In caso contrario, l'implementazione CloudFormation dello stack fallisce, la pipeline si arresta immediatamente e il sistema registra i dettagli per la successiva revisione. Le notifiche di conformità vengono inviate ai soggetti interessati.

Puoi trovare le informazioni relative al successo/fallimento dell'hook nella console: CloudFormation



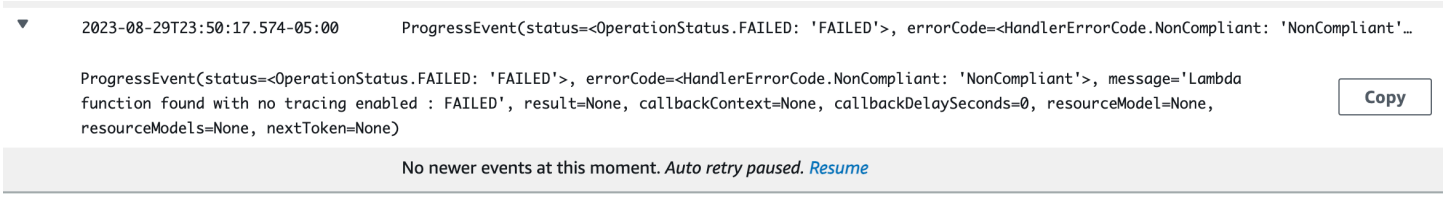
Stack info | Events | Resources | Outputs | Parameters | Template | Change sets

Events (19)

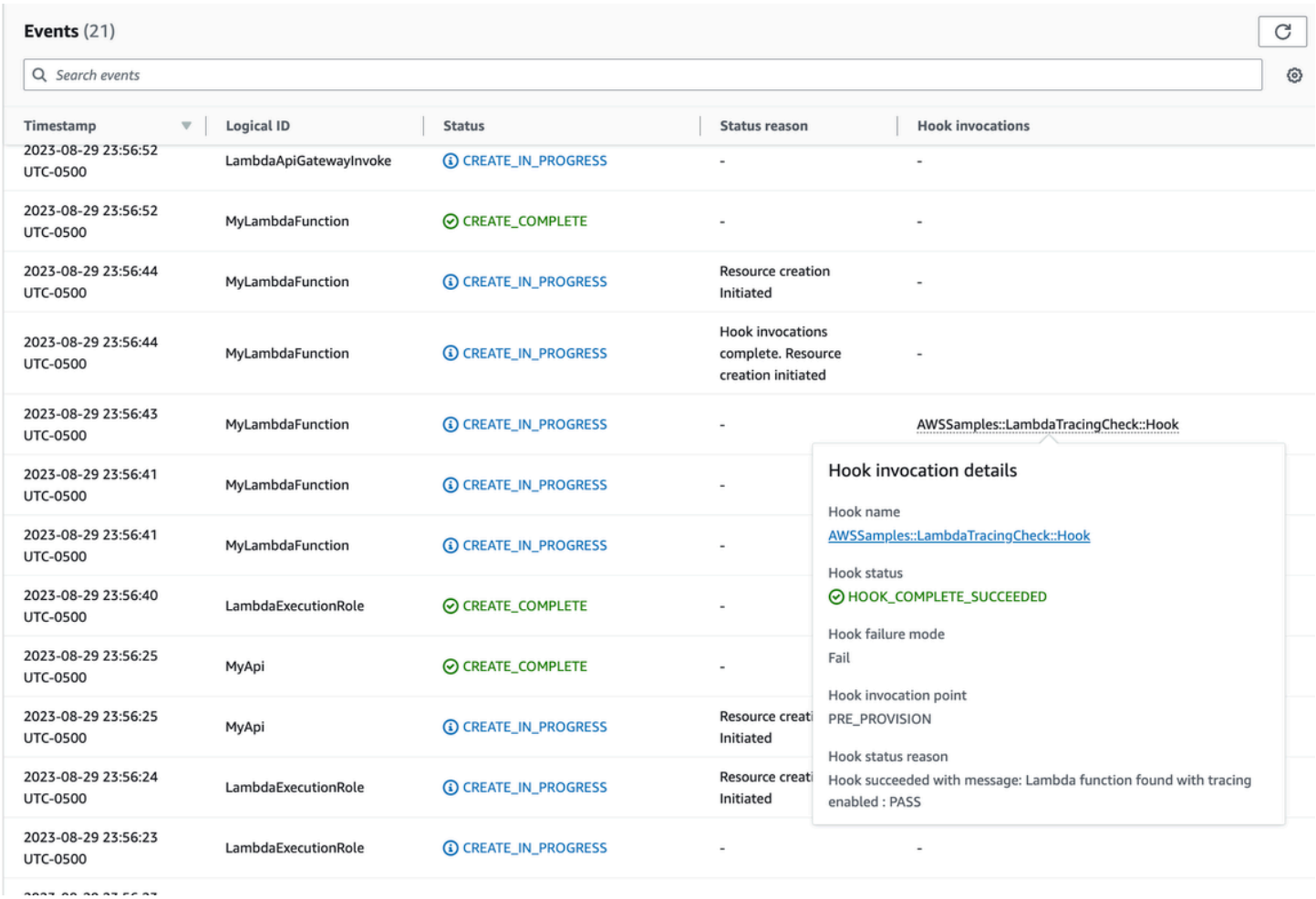
Search events

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: [AWSSamples::LambdaTracingCheck::Hook]	-
2023-08-29 23:50:17 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	-	AWSSamples::LambdaTracingCheck::Hook
2023-08-29 23:50:16 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	-	AWSSamples::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. Di seguito è riportato un log di esempio relativo a un hook con stato non riuscito, che indica il fatto che nella funzione Lambda non è abilitato X-Ray:



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.



In questo modo, puoi implementare controlli preventivi AWS Config in modalità proattiva durante lo sviluppo e la distribuzione di risorse serverless nei tuoi account. AWS Integrando regole AWS Config nella pipeline CI/CD, puoi identificare e, facoltativamente, bloccare le implementazioni di

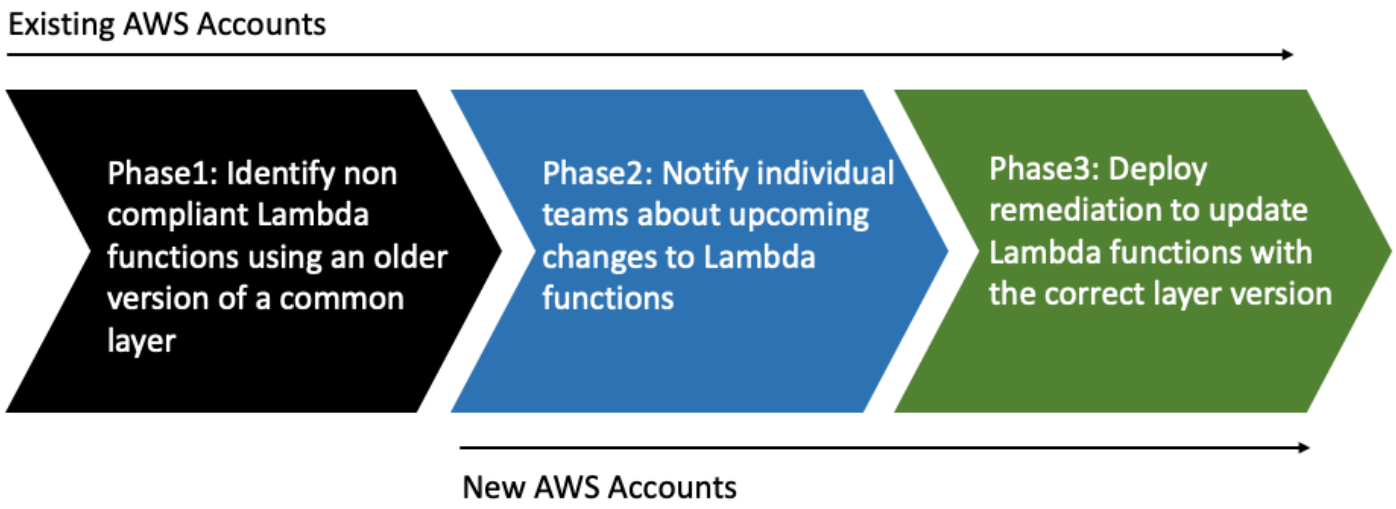
risorse non conformi, come funzioni Lambda prive di una configurazione del tracciamento attiva. Ciò garantisce che negli ambienti vengano implementate solo le risorse conformi alle più recenti politiche di governance. AWS

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 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 riparazione 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 AWS SDK for 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:



Fase 1: identificazione delle risorse di accesso

Inizia eseguendo l'attivazione AWS Config su tutti i tuoi account e configurandola per registrare le funzioni Lambda AWS . Ciò consente di AWS Config 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 data AWS Config using Athena and Amazon QuickSight sul blog AWS Cloud Operations and Management.

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

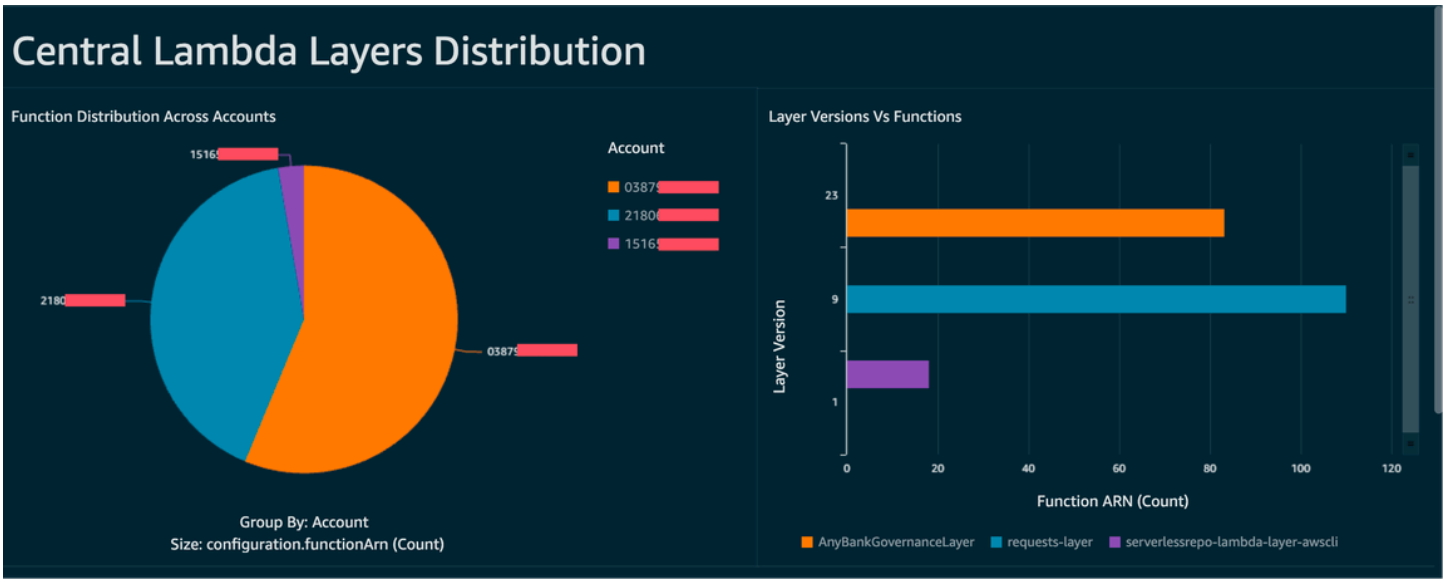
Completed — Time in queue: 127 ms — Run time: 1.803 sec — Data scanned: 239.40 KB

Results (27) — Copy — Download results

Search rows — 1

#	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 QuickSight. Importando i risultati di Athena in QuickSight Amazon, puoi visualizzare in che misura le tue funzioni Lambda aderiscono alla regola della versione del livello. 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, come opzione, puoi abbinare la regola della versione di livello che hai creato nella fase 1 con un'azione di correzione attraverso un documento di automazione Systems Manager, creato come script Python scritto con AWS SDK for 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 consente di convalidare il codice rispetto a una firma digitale per confermare che il codice sia inalterato e provenga da un editore affidabile. AWS Signer può essere utilizzato insieme a AWS Lambda per verificare che le funzioni e i livelli siano inalterati prima dell'implementazione 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. Dopodiché, 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 sarà 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 ha configurato AWS Signer con Lambda per bloccare le implementazioni 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
```

Otterrai un output come il seguente, dove `jobId` è l'oggetto creato nel bucket e nel prefisso di destinazione e `jobOwner` è l'ID dell'Account AWS a 12 cifre in cui il processo è stato eseguito.

```
{
    "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 stai creando e implementando le funzioni con AWS Serverless Application Model (AWS SAM), il comando del pacchetto gestisce il caricamento dell'artefatto zip in S3, 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 implementati negli account siano affidabili per l'implementazione. Puoi includere il processo di cui sopra nelle pipeline CI/CD e richiedere che tutte le funzioni abbiano una configurazione di firma del codice associata utilizzando le tecniche descritte negli argomenti precedenti. Utilizzando la firma del codice con le implementazioni delle funzioni Lambda, impedisci a malintenzionati che potrebbero aver ottenuto le credenziali per creare o aggiornare le funzioni di iniettare codice dannoso nelle funzioni.

Automatizza 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`. La console Amazon Inspector ti consente di creare questo ruolo utilizzando un'opzione con un solo clic.

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 di possibili rilevamenti, consulta 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.

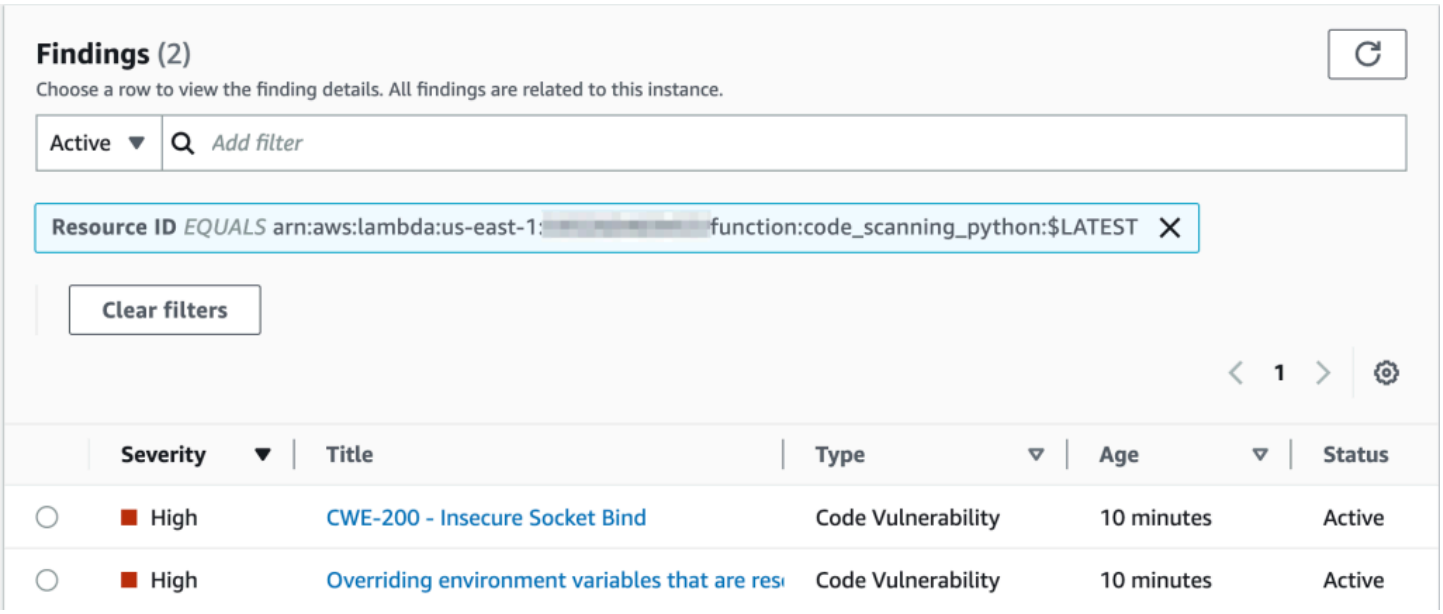
Per escludere una funzione Lambda dalla scansione standard, etichetta la funzione con la seguente coppia chiave-valore:

- `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.



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	Override of reserved variable names in a Lambda function
Relevant CWE	--
Rule ID	Rule-434311
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 Usa le variabili di ambiente Lambda per configurare i valori nel codice.

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 AWS non conformi. Ogni modifica apportata alle risorse serverless viene registrata in AWS Config. Inoltre, AWS Config ti consente di archiviare i dati degli snapshot di configurazione in S3. Puoi usare Amazon Athena e Amazon QuickSight 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 ottenere configurazioni importanti come l'ID dell'Account AWS, Regione, la configurazione del tracciamento AWS X-Ray, la configurazione del VPC, la dimensione della memoria, il runtime e i 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 QuickSight dashboard Amazon e visualizzare i dati. Per aggregare i dati di configurazione AWS delle risorse, creare tabelle in Athena e creare dashboard QuickSight Amazon sui dati di Athena, consulta Visualizing AWS Config data using Athena and Amazon on the Cloud Operations and QuickSight Management blog. 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.

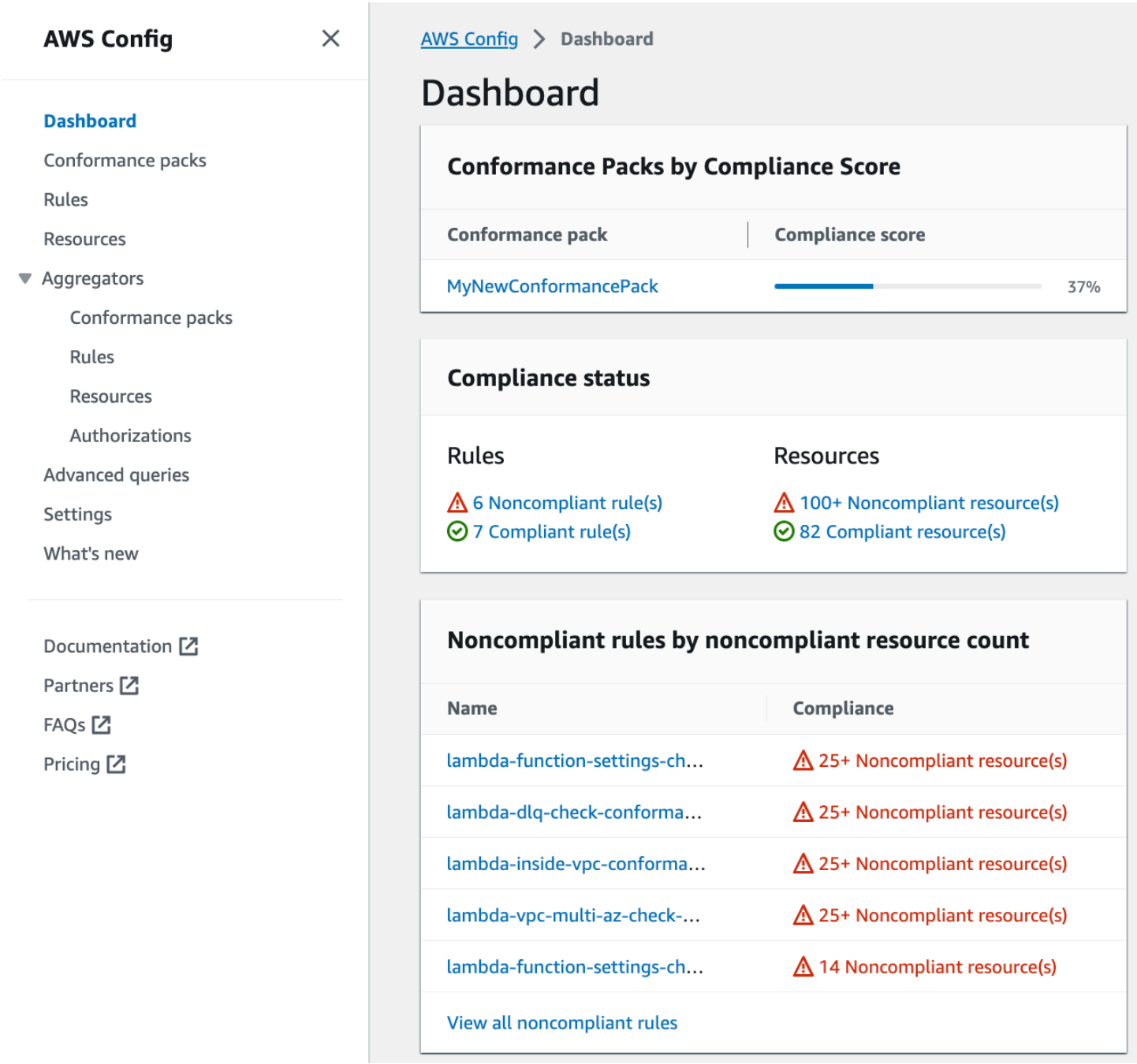
Per ulteriori informazioni sulle azioni che puoi intraprendere, consulta la sezione Risoluzione degli esiti di osservabilità più avanti in questo argomento.

Visibilità sulla conformità Lambda

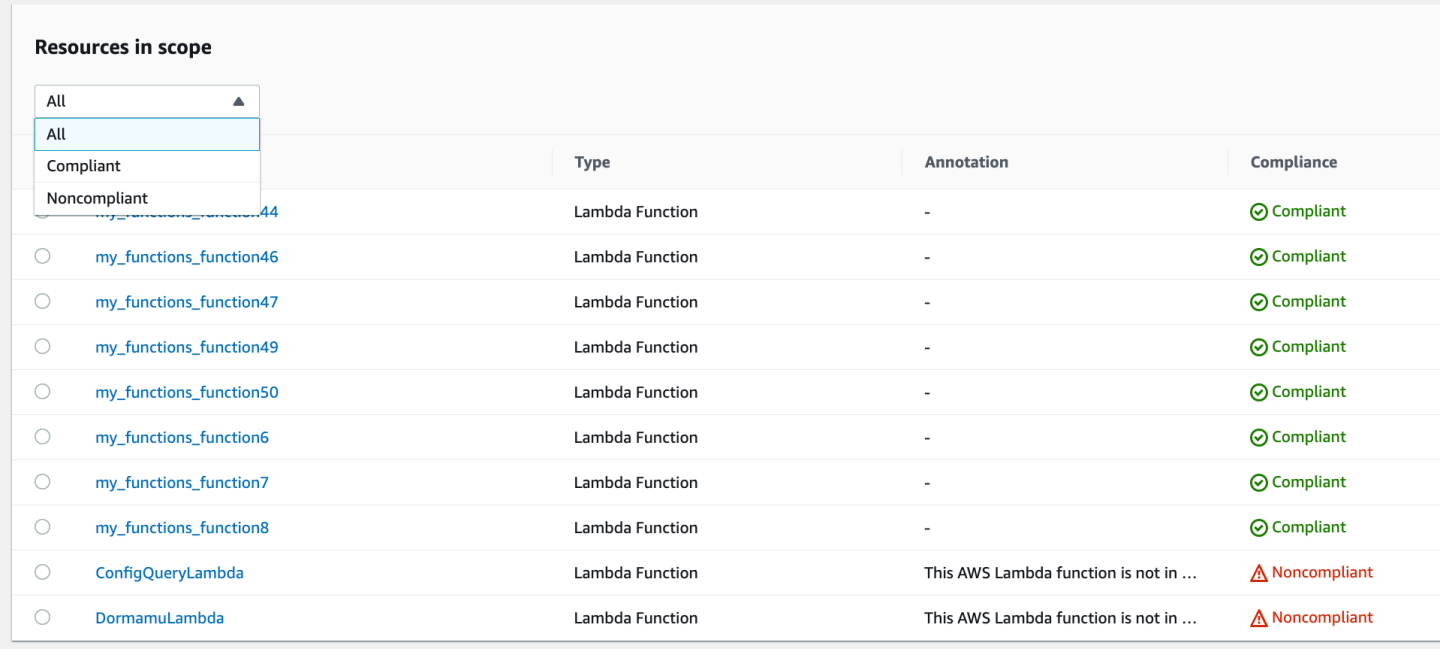
Con i dati generati da AWS Config, puoi creare dashboard a livello di organizzazione per monitorare la conformità. Ciò consente il tracciamento e il monitoraggio coerenti di:

- Pacchetti di conformità per punteggio di conformità

- Regole per risorse non conformi
- Compliance status (Stato di conformità)

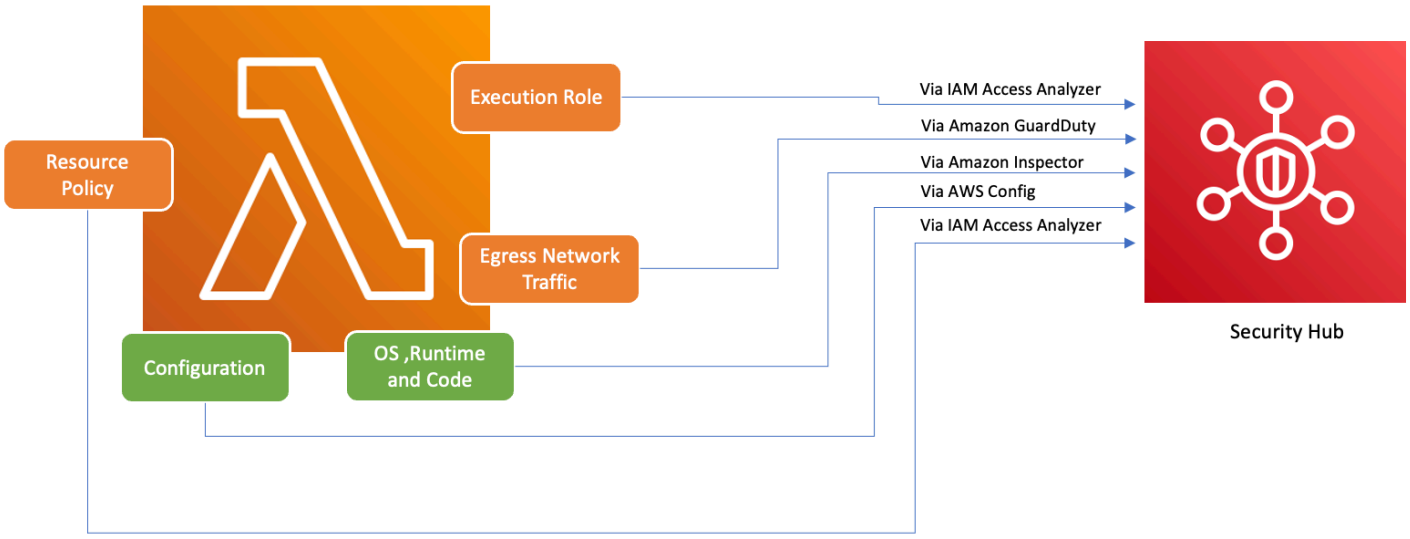


Controlla ogni regola per identificare le risorse non conformi alla stessa. Ad esempio, se l'organizzazione impone che tutte le funzioni Lambda siano associate a un VPC e se hai implementato una regola di AWS Config per identificare la conformità, puoi selezionare la regola `lambda-inside-vpc` nell'elenco precedente.



Per ulteriori informazioni sulle azioni che puoi intraprendere, consulta la sezione Risoluzione degli esiti di osservabilità di seguito.

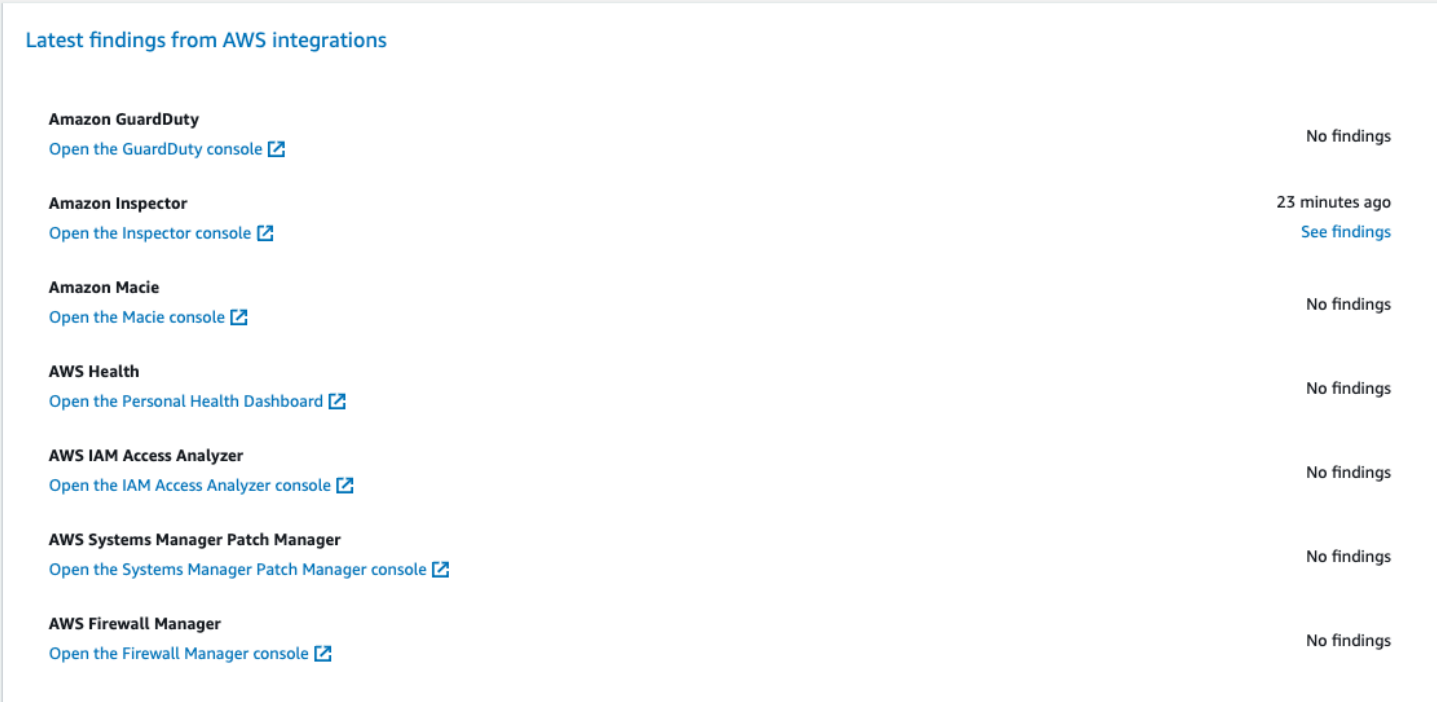
Visibilità sui limiti delle funzioni Lambda con Centrale di sicurezza



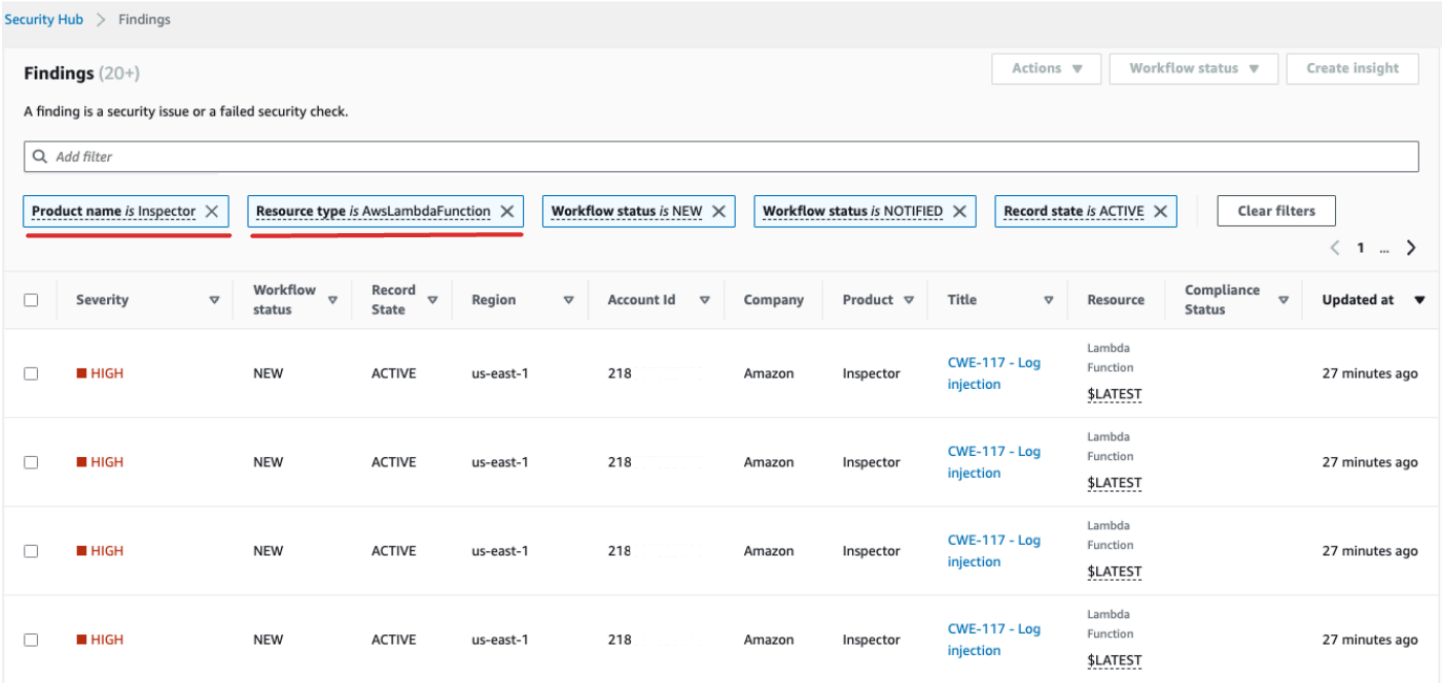
Per garantire che i servizi AWS, tra cui Lambda, vengano utilizzati in modo sicuro, AWS ha introdotto le Foundational Security Best Practices v1.0.0. Questo insieme di best practice fornisce linee guida chiare per proteggere risorse e dati nell'ambiente AWS, sottolineando l'importanza di mantenere una posizione di sicurezza solida. AWS Security Hub integra quanto descritto con l'offerta di un centro unificato per la sicurezza e la conformità. Aggrega, organizza e dà priorità ai risultati di sicurezza

provenienti da più servizi come AWS Amazon Inspector e Amazon. AWS Identity and Access Management Access Analyzer GuardDuty

Se hai Security Hub, Amazon Inspector, IAM Access Analyzer e sono GuardDuty abilitati all'interno della tua AWS organizzazione, Security Hub aggrega automaticamente i risultati di questi servizi. Prendiamo ad esempio Amazon Inspector. Utilizzando Centrale di sicurezza, puoi identificare in modo efficiente le vulnerabilità di codice e pacchetti nelle funzioni Lambda. Nella console di Centrale di sicurezza, accedi alla sezione inferiore, denominata Risultati più recenti delle integrazioni AWS. Qui puoi visualizzare e analizzare gli esiti provenienti da vari servizi AWS integrati.



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.



Infatti GuardDuty, puoi 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'AddPermissionoperazione 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, Centrale di sicurezza 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

Revisori di terze parti valutano la sicurezza e la conformità di AWS Lambda come parte di più programmi di conformità di AWS. Questi includono SOC, PCI, FedRAMP, HIPAA e altri.

Per un elenco di servizi AWS che rientrano nell'ambito di programmi di conformità specifici, consultare Servizi AWS coperti dal programma di compliance. Per informazioni generali, consultare Programmi per la conformità di AWS.

Puoi 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 Crea una strategia di governance per le funzioni e i livelli Lambda.

Resilienza nell'AWS Lambda

L'infrastruttura globale di AWSè basata su Regioni e zone di disponibilità AWS. AWS forniscono più zone di disponibilità fisicamente separate e isolate che sono connesse tramite reti altamente ridondanti, a bassa latenza e velocità effettiva elevata. 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 sulle Regioni e le Zone di disponibilità AWS, consulta Infrastruttura globale di AWS.

Oltre all'infrastruttura globale di AWS, Lambda offre numerose funzionalità per 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 all'utilizzo delgi alias, è possibile utilizzare la gestione delle versioni per eseguire distribuzioni blue/green e progressive. Per informazioni dettagliate, vedi 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 Comprendere il ridimensionamento delle funzioni 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 Offrire 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 concorrenza 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 Comprensione del comportamento dei tentativi in Lambda.
- Coda DLQ – In caso di invocazioni asincrone, è possibile configurare Lambda affinché invii le richieste a una dead-letter queue se tutti i tentativi ripetuti non vanno a buon fine. 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 Code DLQ.

Sicurezza dell'infrastruttura nell'AWS Lambda

Come servizio gestito, AWS Lambda è protetto dalla sicurezza di rete globale AWS. Per informazioni sui servizi di sicurezza AWSe su come AWSprotegge l'infrastruttura, consulta la pagina Sicurezza del cloud AWS. Per progettare l'ambiente AWSutilizzando le best practice per la sicurezza dell'infrastruttura, consulta la pagina Protezione dell'infrastruttura nel Pilastro della sicurezza di AWSWell-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.

Inoltre, le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associata a un principale IAM. In alternativa, è possibile utilizzare AWS Security Token Service (AWS STS) per generare le credenziali di sicurezza temporanee per sottoscrivere le richieste.

Monitoraggio e risoluzione dei problemi delle funzioni Lambda

AWS Lambda si integra con altri AWS servizi per aiutarti a monitorare e risolvere i problemi delle funzioni Lambda. Lambda monitora automaticamente le funzioni Lambda per tuo conto e riporta i parametri 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 utilizzare altri AWS servizi per risolvere i problemi delle 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.

Per ulteriori informazioni sul monitoraggio delle applicazioni Lambda, consulta Monitoraggio e osservabilità in Serverless Land.

Sections

Monitoraggio delle funzioni sulla console Lambda

Il servizio Lambda monitora le funzioni per tuo conto e invia i parametri ad Amazon. CloudWatch La console Lambda permette di creare dei grafici di tali parametri e di mostrarli nella pagina Monitoraggio di ciascuna funzione Lambda.

La console Lambda offre una visualizzazione di parametri, log e tracce in un unico riquadro. La console fornisce filtri per l'intervallo di tempo, il fuso orario e le opzioni di aggiornamento che si applicano universalmente a tutti i riquadri. Puoi correlare facilmente parametri, log e tracce, riducendo il tempo medio di ripristino (MTTR) durante la risoluzione degli errori nelle funzioni Lambda.

Prezzi

CloudWatch ha un piano gratuito perpetuo. Oltre la soglia del livello gratuito, CloudWatch addebiti per metriche, dashboard, allarmi, registri e approfondimenti. Per ulteriori informazioni, consulta i CloudWatch prezzi di Amazon.

Uso della console Lambda

Puoi monitorare le funzioni e le applicazioni Lambda dalla console Lambda.

Per monitorare una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Selezionare la scheda Monitor (Monitora).

Tipi di grafici di monitoraggio

Nella sezione seguente vengono descritti i grafici di monitoraggio nella console Lambda.

Grafici di monitoraggio di Lambda

- Invocations (Chiamate) – Il numero di volte in cui la funzione è stata richiamata.
- Durata: la quantità di tempo minima, media e massima che il codice della funzione impiega per l'elaborazione di un evento.
- Conteggio degli errori e percentuale di successo (%): il numero di errori e la percentuale di invocazioni completate senza errori.

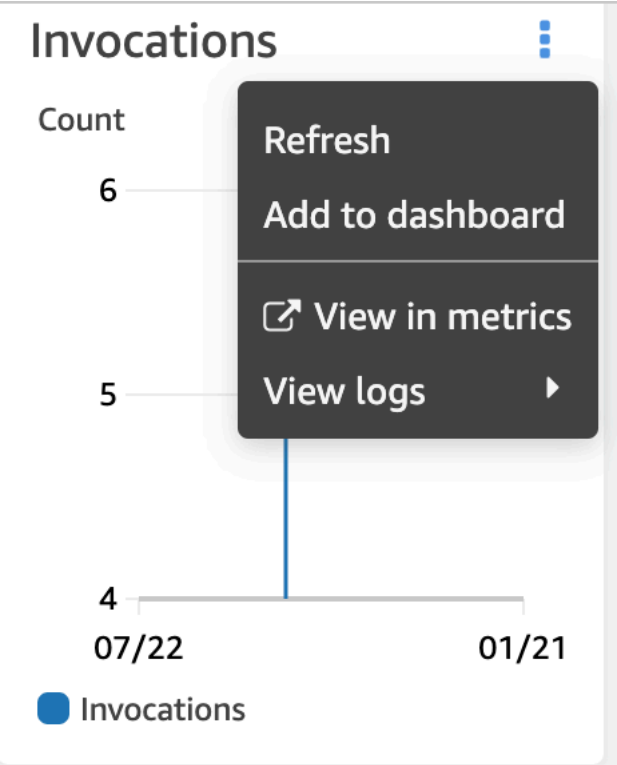
- Limitazioni: il numero di volte in cui l'invocazione non si è conclusa correttamente a causa di limiti di convergenza.
- IteratorAge— Per le sorgenti di eventi di flusso, l'età dell'ultimo elemento del batch quando Lambda lo ha ricevuto e ha richiamato la funzione.
- Async delivery failures (Errori di recapito asincrono) – Numero di errori che si sono verificati nel tentativo di scrivere in una coda di destinazione o di lettere DLQ da parte di Lambda.
- Concurrent executions (Esecuzioni simultanee) – Numero di istanze di funzione che stanno elaborando eventi.

Visualizzazione di grafici nella console Lambda

La sezione seguente descrive come visualizzare i grafici di CloudWatch monitoraggio sulla console Lambda e aprire CloudWatch il dashboard delle metriche.

Per visualizzare i grafici di monitoraggio per una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Selezionare la scheda Monitor (Monitora).
4. Scegli tra gli intervalli di tempo predefiniti oppure scegli un intervallo di tempo personalizzato.
5. Per visualizzare la definizione di un grafico CloudWatch, scegli i tre punti verticali (azioni Widget), quindi scegli Visualizza nelle metriche per aprire la dashboard delle metriche sulla console. CloudWatch

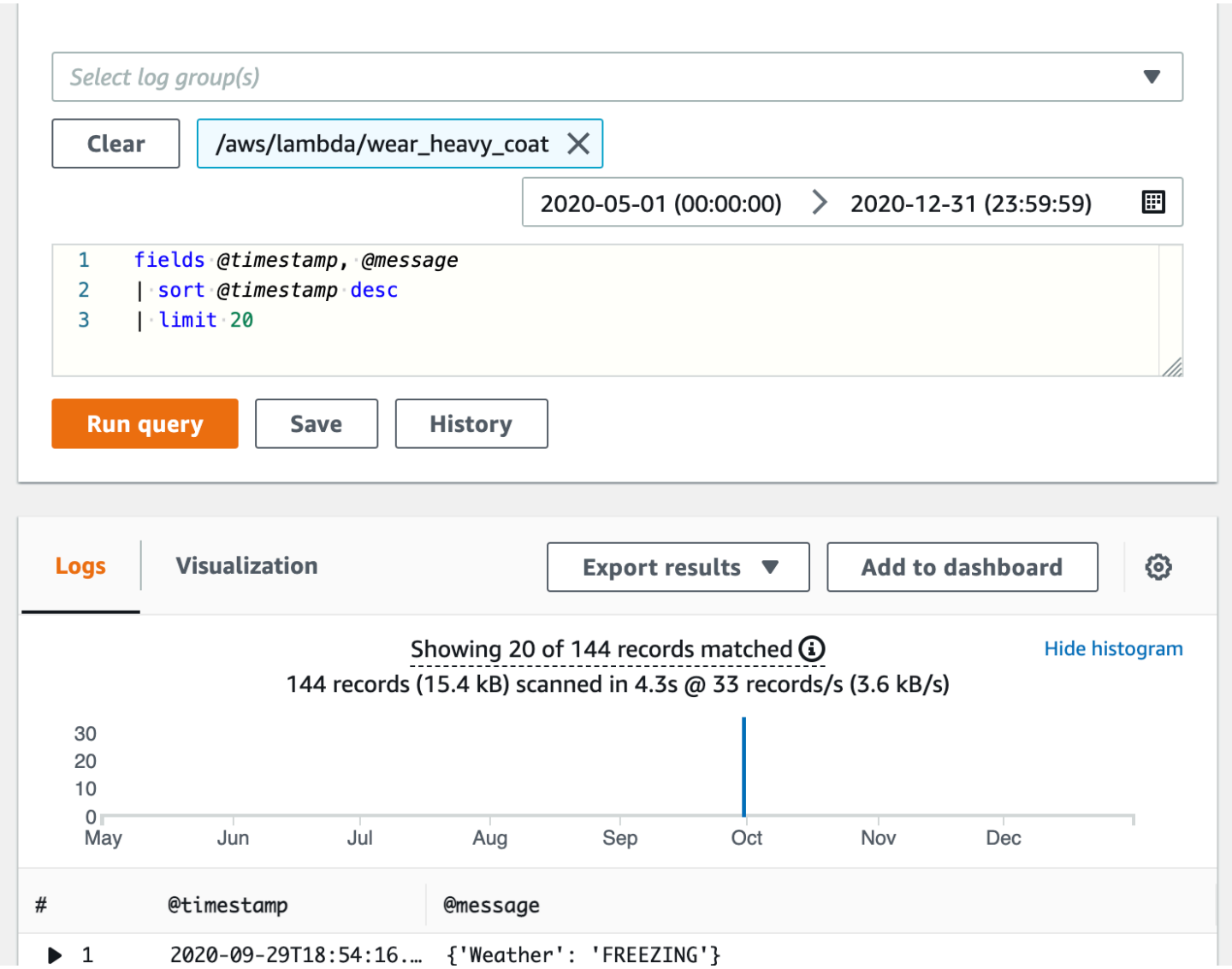


Visualizzazione delle interrogazioni sulla console Logs CloudWatch

La sezione seguente descrive come visualizzare e aggiungere report da CloudWatch Logs Insights a una dashboard personalizzata nella CloudWatch console Logs.

Per visualizzare i report per una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Selezionare la scheda Monitor (Monitora).
4. Scegli Visualizza i log in. CloudWatch
5. Seleziona Visualizza in Log Insights.
6. Scegli tra gli intervalli di tempo predefiniti oppure scegli un intervallo di tempo personalizzato.
7. Scegli Esegui query.
8. (Facoltativo) Seleziona Salva.



Fasi successive

- Scopri le metriche registrate e CloudWatch inviate da Lambda. Utilizzo dei parametri delle funzioni Lambda
- Scopri come utilizzare CloudWatch Lambda Insights per raccogliere e aggregare le metriche delle prestazioni di runtime della funzione Lambda e i log in. Monitora le prestazioni delle funzioni con Amazon CloudWatch Lambda Insights

Utilizzo dei parametri delle funzioni Lambda

Quando la AWS Lambda funzione termina l'elaborazione di un evento, Lambda invia i parametri relativi alla chiamata ad Amazon. CloudWatch Questi parametri non prevedono alcun costo.

Sulla CloudWatch console, puoi creare grafici e dashboard con queste metriche. È possibile impostare allarmi per rispondere alle modifiche apportate alle percentuali di utilizzo, prestazioni o errore. Lambda invia i dati metrici a CloudWatch intervalli di 1 minuto. Per informazioni più approfondite sulla funzione Lambda, è possibile creare parametri personalizzati ad alta risoluzione come descritto in Serverless Land. Per metriche e allarmi personalizzati si applicano costi. CloudWatch Per ulteriori informazioni, consulta Prezzi di Amazon CloudWatch.

Questa pagina descrive le metriche di invocazione, prestazioni e concorrenza della funzione Lambda disponibili sulla console. CloudWatch

Sections

Visualizzazione delle metriche sulla console CloudWatch

È possibile utilizzare la CloudWatch console per filtrare e ordinare le metriche delle funzioni in base al nome della funzione, all'alias o alla versione.

Per visualizzare le metriche sulla console CloudWatch

1. Apri la pagina Metriche (`AWS/Lambda`namespace) 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.
   - In tutte le funzioni (nessuna): visualizza le metriche aggregate per tutte le funzioni della versione corrente. Regione AWS

3. Scegli un parametro, quindi scegli Aggiungi a grafico o un'altra opzione grafica.

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

La sezione seguente descrive i tipi di metriche Lambda disponibili sulla console. CloudWatch

Parametri di invocazione

I parametri di invocazione sono indicatori binari del risultato di una chiamata alla funzione Lambda. 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.

> **Note**
>
> Visualizza i seguenti parametri di invocazione con la statistica `Sum`.

- `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 la chiamata asincrona, il numero di tentativi di invio non riusciti da parte di Lambda di un evento a una coda DLQ. Gli errori DLQ possono verificarsi a causa di risorse configurate erroneamente o limiti di dimensione.
- `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`.
- `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 con provisioning.
- `ProvisionedConcurrencySpilloverInvocations`: il numero di volte in cui il codice di funzione viene chiamato tramite la simultaneità standard quando è in uso tutta la simultaneità con provisioning.
- `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. Usa il rilevamento ricorsivo del loop Lambda per prevenire loop infinitimonitora quante volte una funzione viene richiamata come parte di una catena di richieste tracciando i metadati aggiunti dagli SDK supportati. AWS Se la funzione viene richiamata come parte di una catena di richieste più di 16 volte, Lambda interrompe l'invocazione successiva.

Parametri prestazionali

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 sorgenti di eventi DynamoDB, Kinesis e Amazon DocumentDB, l'età 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 con provisioning. Per ogni chiamata di un alias o versione con la simultaneità fornita, Lambda emette il conteggio corrente.
- `ProvisionedConcurrencyUtilization`— Per una versione o un alias, il valore `ProvisionedConcurrentExecutions` diviso per la quantità totale di valuta simultanea fornita configurata. Ad esempio, se si configura una concorrenza predisposta di 10 per la funzione e il valore è 7, il valore `ProvisionedConcurrentExecutions` è 0,7. `ProvisionedConcurrencyUtilization`
- `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.

Utilizzo dei CloudWatch log di Amazon con AWS Lambda

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto per aiutarti a risolvere i guasti nelle tue funzioni. Se il ruolo di esecuzione della funzione dispone delle autorizzazioni necessarie, Lambda acquisisce i log per tutte le richieste gestite dalla funzione e li invia ad Amazon Logs. CloudWatch

È possibile inserire istruzioni di registrazione nel codice per verificare che il codice funzioni nel modo previsto. Lambda si integra automaticamente con CloudWatch Logs e invia tutti i log del codice a un CloudWatch gruppo di log associato a una funzione Lambda.

Per impostazione predefinita, Lambda invia i log a un gruppo di log denominato `/aws/lambda/<function name>`. Se desideri che la tua funzione invii i log a un altro gruppo, puoi configurarla utilizzando la console Lambda, la AWS CLI() o AWS Command Line Interface l'API Lambda. Per ulteriori informazioni, consulta the section called “Configurazione dei gruppi di log CloudWatch ”.

Puoi visualizzare i log delle funzioni Lambda utilizzando la console Lambda, la console, CloudWatch il AWS CLI() o AWS Command Line Interface l'API. CloudWatch

> **Note**
>
> Potrebbero essere necessari da 5 a 10 minuti prima che i log vengano visualizzati dopo una chiamata di funzione.

Sezione

Prerequisiti

Il ruolo di esecuzione richiede l'autorizzazione per caricare i log su Logs. CloudWatch
Puoi aggiungere le autorizzazioni di CloudWatch Logs 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 dei CloudWatch log. Per ulteriori informazioni, consulta la pagina dei prezzi. CloudWatch

Configurazione dei controlli di registrazione avanzati per la funzione Lambda

Per avere un maggiore controllo sul modo in cui i log delle tue funzioni vengono acquisiti, elaborati e utilizzati, Lambda offre le seguenti opzioni di configurazione della registrazione:

- Formato di log: scegli tra i formati di testo normale e JSON strutturato per i log della funzione
- Livello di log: per i log strutturati 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 log a cui la CloudWatch funzione invia i 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.

Per tutti i runtime gestiti da Lambda, puoi scegliere se i log di sistema della tua funzione vengono inviati 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.

Runtime	Versioni supportate
Java	Tutti i runtime Java eccetto Java 8 su Amazon Linux 1
Node.js	Node.js 16 e versioni successive
Python	Python 3.7 e versioni successive

Affinché Lambda invii 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 logger `LambdaLogger` o Log4j2.

- Node.js: i metodi della console `console.trace`, `console.debug`, `console.log`, `console.info`, `console.error` e `console.warn`
- Python: la libreria standard `logging` di Python

Per ulteriori informazioni sull'utilizzo dei controlli di registrazione avanzati con runtime supportati, consulta le pagine the section called “Registrazione”, the section called “Registrazione” e the section called “Registrazione”.

Per altri runtime Lambda gestiti, Lambda attualmente supporta nativamente solo l'acquisizione dei log di sistema in formato JSON strutturato. Tuttavia, puoi comunque acquisire i log delle applicazioni in formato JSON strutturato in qualsiasi runtime utilizzando strumenti di registrazione come Powertools per generare output di log in formato JSON. AWS Lambda

Formati di log predefiniti

Attualmente, il formato di log predefinito per tutti i runtime Lambda è il testo normale.

Se state già utilizzando librerie di registrazione come Powertools per AWS Lambda generare i log delle funzioni in formato strutturato JSON, non è necessario modificare il codice se selezionate la formattazione dei log JSON. Lambda non codifica due volte i log che sono già codificati in JSON, quindi i log delle applicazioni della funzione continueranno 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:

```
2023-03-13 18:56:24.046000 fbe8c1   INIT_START  Runtime Version:
 python:3.9.v18  Runtime Version ARN: arn:aws:lambda:eu-
west-1::runtime:edb5a058bfa782cb9cedc6d534ac8b8c193bc28e9a9879d9f5ebaaf619cd0fc0
```

Example JSON strutturato:

```
{
  "time": "2023-03-13T18:56:24.046Z",
  "type": "platform.initStart",
  "record": {
    "initializationType": "on-demand",
    "phase": "init",
    "runtimeVersion": "python:3.9.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 e Node.js) o `"AWSrequestId"` (Java): l'ID di richiesta univoco per l'invocazione della funzione

A seconda del runtime e del metodo di registrazione di log utilizzato dalla funzione, questo oggetto JSON può anche contenere coppie di chiavi aggiuntive. Ad esempio, in Node.js, 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`. Per ulteriori informazioni sui log in formato JSON in diversi runtime Lambda, consulta the section called “Registrazione”, the section called “Registrazione” e the section called “Registrazione”.

> **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:

```
2023-10-27T19:17:45.586Z 79b4f56e-95b1-4643-9700-2807f4e68189 INFO some log message
```

Example JSON strutturato:

```
{
    "timestamp":"2023-10-27T19:17:45.586Z",
    "level":"INFO",
    "message":"some log message",
    "requestId":"79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Impostazione del formato di log della funzione

Per configurare il formato di registro per la tua funzione, puoi usare la console Lambda o il AWS Command Line Interface ()AWS CLI. Puoi anche configurare il formato di registro di una

funzione utilizzando i comandi CreateFunctione l'API UpdateFunctionConfiguration Lambda, la risorsa AWS Serverless Application Model (AWS SAM) e la AWS::Serverless::Function AWS CloudFormationAWS::Lambda::Functionrisorsa.

La modifica del formato di registro della funzione non influisce sui log esistenti archiviati in CloudWatch Logs. Solo i nuovi log utilizzeranno 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 e inferiore a Logs. CloudWatch Per ulteriori informazioni sul filtraggio a livello di log di applicazioni e sistemi, consulta the section called "Filtraggio a livello di log"

> **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".

Per le funzioni Node.js che emettono log in formato EMF (Embedded Metric Format), la modifica del formato di registro della funzione in JSON potrebbe comportare l'impossibilità di riconoscere le metriche. CloudWatch

> **Important**
>
> Se la vostra funzione utilizza Powertools for AWS Lambda (TypeScript) o le librerie client EMF open source per emettere i log EMF, aggiornate le librerie Powertools ed EMF alle versioni più recenti per assicurarvi che possa continuare ad analizzare i log correttamente. 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".

Configurazione del formato 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. In Contenuto del log, per Formato del log seleziona Testo o JSON.
6. Selezionare Salva.

Modifica del formato di log 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
```

Configurazione del formato di log durante la creazione di 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 comando di esempio seguente crea una funzione utilizzando il runtime Node.js 18 che genera i log in formato JSON strutturato.

  Se non specifichi un formato di log quando crei una funzione, Lambda utilizzerà il formato di log 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 nodejs18.x \
--handler index.handler --zip-file fileb://function.zip \
--role arn:aws:iam::123456789012:role/LambdaRole --logging-config LogFormat=JSON
```

Filtraggio a livello di log

Lambda può filtrare i log della funzione in modo che solo i log con un certo livello di dettaglio o inferiore vengano inviati a Logs. CloudWatch 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":
 "2023-11-02T16:51:31.587199Z"}')
```

Se il campo del valore `"level"` non è valido o è assente, Lambda assegnerà all'output 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 assegnerà al log il livello INFO e aggiungerà un timestamp per tuo conto.

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. Ad esempio, nelle funzioni che utilizzano il runtime .NET, Lambda riconosce la chiave `"Timestamp"`.

> **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 funzionam ento della funzione
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'appl icazione

Quando si seleziona un livello di registro, Lambda invia i log a quel livello e successivamente a Logs. CloudWatch Ad esempio, se imposti il livello di log dell'applicazione di una funzione su WARN, Lambda non invia output 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 senza livello verrà 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 relativi a potenziali errori che possono 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 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 filtraggio a livello di registro dell'applicazione e del sistema per la tua funzione, puoi utilizzare la console Lambda o il (). AWS Command Line Interface AWS CLI Puoi anche configurare il livello di registro di una funzione utilizzando i comandi CreateFunctione l'API UpdateFunctionConfiguration Lambda, la risorsa AWS Serverless Application Model (AWS SAM) e la AWS::Serverless::Function AWS CloudFormationAWS::Lambda::Functionrisorsa.

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.

Configurazione del livello di log dell'applicazione o di sistema di una funzione esistente (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 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. Selezionare Salva.

Configurazione del livello di log dell'applicazione o 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`. Imposta `--system-log-level` su `DEBUG`, `INFO` o `WARN`. Imposta `--application-log-level` su `DEBUG`, `INFO`, `WARN`, `ERROR` o `FATAL`.

```
aws lambda update-function-configuration \
--function-name myFunction --system-log-level WARN \
--application-log-level ERROR
```

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 le opzioni `--system-log-level` e `--application-log-level` nel comando create-function. Imposta `--system-log-level` su `DEBUG`, `INFO` o `WARN`. Imposta `--application-log-level` su `DEBUG`, `INFO`, `WARN`, `WARN` o `FATAL`.

```
aws lambda create-function --function-name myFunction --runtime nodejs18.x \
--handler index.handler --zip-file fileb://function.zip \
--role arn:aws:iam::123456789012:role/LambdaRole --system-log-level WARN \
--application-log-level ERROR
```

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
iniziare RuntimeDone	status=success	WARN
initReport	initializationType=snapstart	INFO
initReport	initializationType!=snapstart	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 runtime personalizzato per controllare 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.

Configurazione dei gruppi di log CloudWatch

Per impostazione predefinita, crea CloudWatch automaticamente un gruppo di log denominato in base alla funzione quando viene richiamata `/aws/lambda/<function name>` 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 CreateFunctione

UpdateFunctionConfiguration Lambda API e la risorsa AWS Serverless Application Model (AWS SAM) AWS: :Serverless: :Function.

È possibile configurare più funzioni Lambda per inviare i log allo stesso CloudWatch gruppo di log. 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 dei 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 si configura un gruppo di log personalizzato, il nome selezionato per il 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 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 Logging group, 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.

Modifica del gruppo di log di una funzione (AWS CLI)

- Per modificare il gruppo di log di una funzione esistente, utilizza il comando `update-function-configuration`. Se specifichi il nome di un gruppo di log esistente, la funzione

utilizzerà quel gruppo. Se non esiste alcun gruppo di log con il nome specificato, Lambda creerà un nuovo gruppo di log per la funzione con tale nome.

```
aws lambda update-function-configuration \
--function-name myFunction --log-group myLogGroup
```

Definizione di un gruppo di log personalizzato durante la creazione di una funzione (AWS CLI)

- Per specificare un gruppo di log personalizzato quando si crea una nuova funzione Lambda utilizzando AWS CLI, utilizzare l'`--log-group`opzione. Se specifichi il nome di un gruppo di log esistente, la funzione utilizzerà quel gruppo. Se non esiste alcun gruppo di log con il nome specificato, Lambda creerà un nuovo gruppo di log per la funzione con tale nome.

  Il comando di esempio seguente crea una funzione Lambda Node.js che invia i log a un gruppo di log denominato `myLogGroup`.

  ```
  aws lambda create-function --function-name myFunction --runtime nodejs18.x \
  --handler index.handler --zip-file fileb://function.zip \
  --role arn:aws:iam::123456789012:role/LambdaRole --log-group myLogGroup
  ```

Autorizzazioni del ruolo di esecuzione

Affinché la funzione invii log a CloudWatch Logs, deve disporre dell'autorizzazione. `logs:PutLogEvents` Quando configuri il gruppo di log della funzione utilizzando la console Lambda, se la funzione non dispone di questa autorizzazione, Lambda la aggiunge al ruolo di esecuzione della funzione per impostazione predefinita. Quando Lambda aggiunge questo permesso, concede alla funzione il permesso di inviare i log a qualsiasi gruppo di log CloudWatch 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 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 politica AWSLambdaBasicExecutionRolegestita.

Accesso ai log con la console Lambda

Per visualizzare i log tramite la console Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegli Monitor (Monitoraggio).
4. Scegli Visualizza accessi. CloudWatch

Accesso ai log con AWS CLI

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario:

- AWS Command Line Interface (AWS CLI) versione 2
- AWS CLI — Configurazione rapida con `aws configure`

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":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "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-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.

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 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.

```
#!/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 seguente comando 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"
}
```


Registrazione delle funzioni di runtime

Per eseguire il debug e verificare che il codice funzioni come previsto, è possibile generare i registri con la funzionalità di registrazione standard per il linguaggio di programmazione. Il runtime Lambda carica l'output di registro della funzione in Logs. CloudWatch Per istruzioni specifiche del linguaggio, consulta gli argomenti riportati di seguito.

- AWS Lambda registrazione delle funzioni in Node.js
- AWS Lambda registrazione delle funzioni in Python
- AWS Lambda registrazione delle funzioni in Ruby
- AWS Lambda registrazione delle funzioni in Java
- AWS Lambda funzione di registrazione in Go
- Registrazione della funzione Lambda in C#
- AWS Lambda funzione di accesso PowerShell

Fasi successive

- Scopri di più sui gruppi di log e sull'accesso ad essi tramite la CloudWatch console nella sezione Sistema di monitoraggio, applicazione e file di registro personalizzati nella Amazon CloudWatch User Guide.

Registrazione delle chiamate AWS Lambda API utilizzando AWS CloudTrail

AWS Lambda è integrato con AWS CloudTrail, un servizio che fornisce una registrazione delle azioni intraprese da un utente, 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 ulteriori dettagli.

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 di IAM Identity Center.
- 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 AWSPer ulteriori informazioni, consulta Lavorare con la cronologia degli CloudTrail eventi nella Guida per l'utente.AWS CloudTrail Non sono CloudTrail previsti costi per la visualizzazione della cronologia degli eventi.

Per una registrazione continua degli eventi degli Account AWS ultimi 90 giorni, crea un trail o un data store di eventi CloudTrailLake.

CloudTrail sentieri

> Un trail consente di CloudTrail inviare file di log a un bucket Amazon S3. Tutti i percorsi creati utilizzando il AWS Management Console sono multiregionali. È possibile creare un percorso a regione singola o multiregione utilizzando. AWS CLI La creazione di un percorso multiregionale è consigliata in quanto consente di registrare l'intera attività del proprio account Regioni AWS . Se crei un percorso a regione singola, puoi visualizzare solo gli eventi registrati nel percorso. Regione

AWS Per ulteriori informazioni sui sentieri, consulta Creazione di un percorso per te Account AWS e Creazione di un percorso per un'organizzazione nella Guida per l'AWS CloudTrail utente.

Puoi inviare gratuitamente una copia dei tuoi eventi di gestione in corso al tuo bucket Amazon S3 CloudTrail creando un percorso, tuttavia ci sono costi di storage di Amazon S3. Per ulteriori informazioni sui CloudTrail prezzi, consulta la pagina Prezzi.AWS CloudTrail Per informazioni sui prezzi di Amazon S3, consulta Prezzi di Amazon S3.

CloudTrail Archivi di dati sugli eventi di Lake

CloudTrail Lake ti consente di eseguire query basate su SQL sui tuoi eventi. CloudTrail Lake converte gli eventi esistenti in formato JSON basato su righe in formato Apache ORC. 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, consulta Working with AWS CloudTrail Lake nella Guida per l'utente.AWS CloudTrail

CloudTrail Gli archivi e le richieste di dati sugli 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, consulta Prezzi.AWS CloudTrail

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 CloudTrail relativo ai dati che viene registrato per impostazione predefinita per i servizi supportati è`LambdaESMDisabled`. Per ulteriori informazioni sull'utilizzo di questo evento per risolvere i problemi relativi alle mappature delle sorgenti degli eventi Lambda, consulta. the section called “Utilizzo CloudTrail per la risoluzione dei problemi relativi alle sorgenti di eventi Lambda disabilitate”

Per gli eventi di dati sono previsti costi aggiuntivi. Per ulteriori informazioni sui CloudTrail prezzi, consulta Prezzi.AWS CloudTrail

Puoi registrare gli eventi relativi ai dati per il tipo di `AWS::Lambda::Function` risorsa utilizzando la CloudTrail console o AWS CLI le operazioni CloudTrail dell'API. Per ulteriori informazioni su come registrare gli eventi relativi ai dati, vedere Registrazione degli eventi relativi ai dati con AWS Management Console e Registrazione degli eventi relativi ai dati con the AWS Command Line Interface nella Guida per l'AWS CloudTrail utente.

La tabella seguente elenca il tipo di risorsa Lambda per il quale è possibile registrare gli eventi relativi ai dati. La colonna Data event type (console) mostra il valore da scegliere dall'elenco dei tipi di evento Data 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 o. 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 registrate su CloudTrail
Lambda	`AWS::Lambda::Function`	Invoke

Puoi configurare selettori di eventi avanzati per filtrare in base a `eventName`, `readOnly`, e `resources.ARN` i campi per registrare solo gli eventi che ritieni importanti. L'esempio seguente è la visualizzazione JSON di una configurazione di eventi di dati che registra gli eventi solo per una funzione specifica. Per ulteriori informazioni su questi campi, consulta l'AWS CloudTrail API AdvancedFieldSelectorReference.

```
[
  {
    "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 di. 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` potrebbero includere informazioni sulla data e sulla versione, ma si riferisce comunque alla stessa azione pubblica dell'API. Ad esempio, l'`GetFunction`azione appare come`GetFunction20150331v2`. L'elenco seguente specifica quando il nome dell'evento è diverso dal nome dell'azione dell'API.

- AddLayerVersionPermission
- AddPermission(nome dell'evento:) `AddPermission20150331v2`
- CreateAlias(nome dell'evento:`CreateAlias20150331`)
- CreateEventSourceMapping(nome dell'evento:`CreateEventSourceMapping20150331`)
- CreateFunction(nome dell'evento:`CreateFunction20150331`)

(I `ZipFile` parametri `Environment` and vengono omessi dai CloudTrail registri per`CreateFunction`.)

- CreateFunctionUrlConfig
- DeleteAlias(nome dell'evento:) `DeleteAlias20150331`
- DeleteCodeSigningConfig
- DeleteEventSourceMapping(nome dell'evento:`DeleteEventSourceMapping20150331`)
- DeleteFunction(nome dell'evento:`DeleteFunction20150331`)
- DeleteFunctionConcorrenza (nome dell'evento:`DeleteFunctionConcurrency20171031`)
- DeleteFunctionUrlConfig
- DeleteProvisionedConcurrencyConfig
- GetAlias(nome dell'evento:`GetAlias20150331`)
- GetEventSourceMapping
- GetFunction
- GetFunctionUrlConfig
- GetFunctionConfigurazione
- GetLayerVersionPolicy
- GetPolicy
- ListEventSourceMappings
- ListFunctions
- ListFunctionUrlConfigs
- PublishLayerVersione (nome dell'evento:`PublishLayerVersion20181031`)

(Il `ZipFile` parametro viene omesso dai CloudTrail registri per`PublishLayerVersion`.)

- PublishVersion(nome dell'evento:) `PublishVersion20150331`
- PutFunctionConcorrenza (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`)
- UpdateFunctionCodice (nome dell'evento:`UpdateFunctionCode20150331v2`)

  (Il `ZipFile` parametro viene omesso dai CloudTrail registri per`UpdateFunctionCode`.)
- UpdateFunctionConfigurazione (nome dell'evento:) `UpdateFunctionConfiguration20150331v2`

  (Il `Environment` parametro viene omesso dai CloudTrail registri per`UpdateFunctionConfiguration`.)
- UpdateFunctionEventInvokeConfig
- UpdateFunctionUrlConfig

Utilizzo CloudTrail per la risoluzione dei problemi relativi alle sorgenti di eventi Lambda disabilitate

Quando si modifica lo stato di una mappatura delle sorgenti di eventi utilizzando l'azione UpdateEventSourceMappingAPI, la chiamata API viene registrata come evento di gestione. CloudTrail Le mappature delle sorgenti degli eventi possono anche passare direttamente `Disabled` allo stato a causa di errori.

Per i seguenti servizi, Lambda pubblica l'evento `LambdaESMDisabled` dei dati CloudTrail quando l'origine dell'evento passa allo stato Disabilitato:

- Amazon Simple Queue Service (Amazon SQS)
- Amazon DynamoDB
- Amazon Kinesis

Lambda non supporta questo evento per nessun altro tipo di mappatura delle sorgenti degli eventi.

Per ricevere avvisi quando le mappature delle sorgenti degli eventi per i servizi supportati passano allo `Disabled` stato, configura un allarme in Amazon CloudWatch utilizzando l'evento. `LambdaESMDisabled` CloudTrail Per ulteriori informazioni sulla configurazione di un CloudWatch allarme, consulta Creazione di CloudWatch allarmi per CloudTrail eventi: esempi.

L'`serviceEventDetails`entità nel messaggio dell'`LambdaESMDisabled`evento 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 proveniente 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` e.

> **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-httplib2/0.8 (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-httplib2/0.8 (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 sul contenuto dei CloudTrail record, consultate il contenuto dei CloudTrail record nella Guida AWS CloudTrail per l'utente.

Visualizza le chiamate alla funzione Lambda utilizzando AWS X-Ray

Puoi utilizzarli AWS X-Ray per visualizzare i componenti dell'applicazione, identificare i punti deboli nelle prestazioni e risolvere le richieste che hanno provocato un errore. Le funzioni Lambda inviano dati di traccia a X-Ray e X-Ray elabora i dati per generare una mappa di servizio e riepiloghi di traccia ricercabili.

Se è stata abilitata l'analisi X-Ray in un servizio che richiama la funzione, Lambda invia automaticamente le tracce a X-Ray. Il servizio upstream, ad esempio Amazon API Gateway, o un'applicazione ospitata su Amazon EC2 analizzata con l'SDK X-Ray, campiona le richieste in arrivo e aggiunge un'intestazione di traccia che indica a Lambda di inviare o meno le tracce. Le tracce dei produttori di messaggi upstream, come Amazon SQS, vengono automaticamente collegate alle tracce delle funzioni Lambda downstream, creando end-to-end una visualizzazione dell'intera applicazione. Per ulteriori informazioni, consulta Tracciamento delle applicazioni guidate dagli eventi nella Guida per gli sviluppatori di AWS X-Ray .

> **Note**
>
> Il tracciamento X-Ray non è attualmente supportato per le funzioni Lambda con Streaming gestito da Amazon per Apache Kafka (Amazon MSK), Apache Kafka gestito dal cliente, Amazon MQ con ActiveMQ e RabbitMQ attivi oppure mappature delle origini degli 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. Scegli Modifica.
5. In X-Ray, attivare Active tracing (Tracciamento attivo).
6. Selezionare Salva.

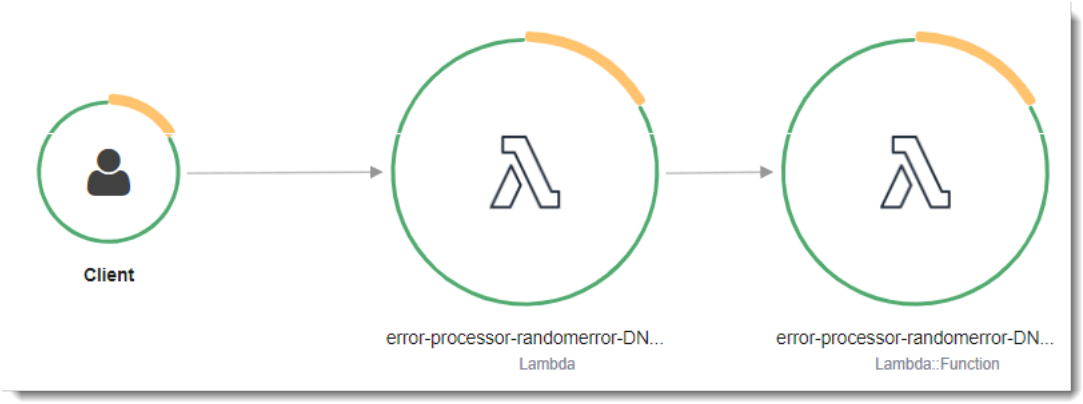
> **Prezzi**
>
> Puoi utilizzare il tracciamento X-Ray gratuitamente ogni mese fino a un determinato limite come parte del AWS piano gratuito. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consultare Prezzi di AWS X-Ray.

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 AWSXRayDaemonWriteAccesspolicy al ruolo di esecuzione.

X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste.

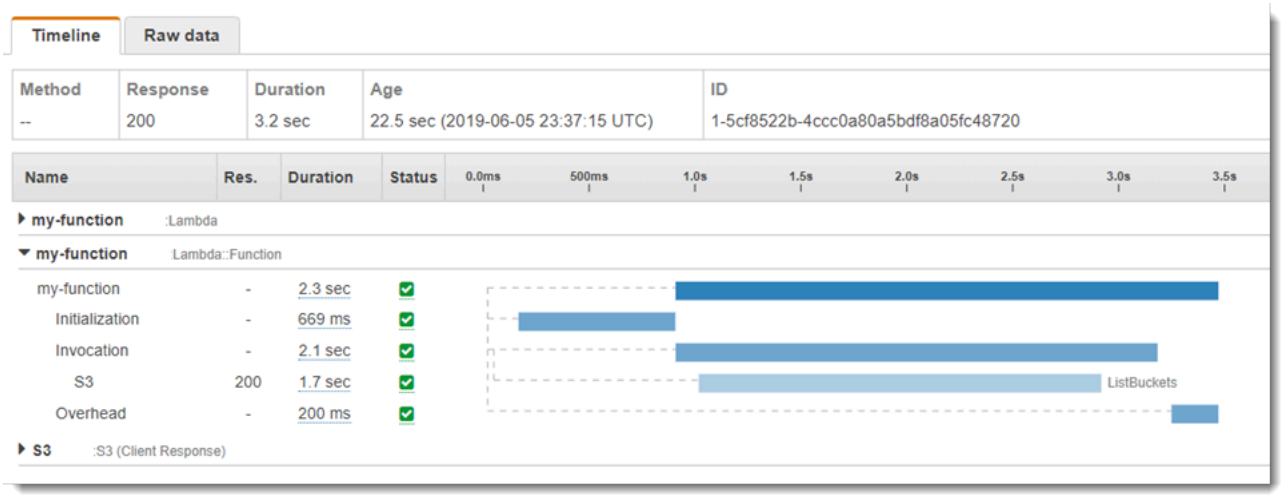
> **Note**
>
> La frequenza di campionamento di X-Ray non può essere configurata per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, il che crea 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 denominati my-function, ma uno ha un'origine di `AWS::Lambda` e l'altro ha

un'origine di. `AWS::Lambda::Function` Se il `AWS::Lambda` segmento mostra un errore, il servizio Lambda presentava un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione presentava un problema.



La funzione segment (`AWS::Lambda::Function`) include sottosegmenti per`Initialization`,`Invocation`, `Restore` (Lambda SnapStartsolo) e. `Overhead` Per ulteriori informazioni, consulta la sezione Ciclo di vita dell'ambiente di esecuzione Lambda.

> **Note**
>
> X-Ray considera le eccezioni non gestite nella funzione Lambda come stati `Error`. X-Ray registra gli stati `Fault` solo quando Lambda presenta errori interni al server. Per maggiori informazioni, consulta Errori, malfunzionamenti ed eccezioni nella Guida per gli sviluppatori di X-Ray.

Il sottosegmento `Initialization` rappresenta la fase iniziale del ciclo di vita dell'ambiente di esecuzione Lambda. 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 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.

(Solo Lambda SnapStart) Il sottosegmento `Restore` mostra il tempo impiegato da Lambda per ripristinare uno snapshot, caricare il runtime (JVM) ed eseguire qualsiasi hook di runtime

`afterRestore`. 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.

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.

> **Note**
>
> Occasionalmente, potresti notare un ampio divario tra le fasi di inizializzazione e invocazione della funzione 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 può inizializzare in modo proattivo un'istanza di funzione, anche in assenza di chiamata. Visivamente, entrambi questi casi si presentano come un intervallo di tempo tra le fasi di inizializzazione e invocazione.

> **Important**
>
> In Lambda, è possibile utilizzare l'SDK X-Ray per estendere il sottosegmento `Invocation` con sottosegmenti aggiuntivi per chiamate a valle, annotazioni e metadati. Non è possibile accedere direttamente al segmento di funzione o registrare la parola eseguita al di fuori dell'ambito di chiamata del gestore.

Vedere i seguenti argomenti per un'introduzione specifica della lingua all'analisi in Lambda:

- Strumentazione del codice Node.js in 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 in AWS Lambda
- Strumentazione del codice C# in AWS Lambda

Per un elenco completo dei servizi che supportano la strumentazione attiva, consulta AWS Servizi supportati nella Guida per gli sviluppatori. AWS X-Ray

Sections

Autorizzazioni del ruolo di esecuzione

Lambda richiede le seguenti autorizzazioni per inviare i dati di traccia a X-Ray. Aggiungerle al ruolo di esecuzione della funzione.

- xray: segmenti PutTrace
- xray: record PutTelemetry

Queste autorizzazioni sono incluse nella politica AWSXRayDaemonWriteAccessgestita.

Il demone AWS X-Ray

Invece di inviare dati di traccia direttamente all'API X-Ray, l'SDK X-Ray utilizza un processo daemon. Il daemon AWS X-Ray è un'applicazione che viene eseguita nell'ambiente Lambda e ascolta il traffico UDP che contiene segmenti e sottosegmenti. Memorizza i dati in entrata e li scrive su X-Ray in batch, riducendo l'elaborazione e il sovraccarico di memoria necessari per tracciare le chiamate.

Il runtime Lambda consente al daemon di raggiungere fino al 3% della memoria configurata della funzione o 16 MB, a seconda di quale sia maggiore. Se la tua funzione esaurisce la memoria durante l'invocazione, il runtime termina prima il processo daemon per liberare memoria.

Il processo daemon è completamente gestito da Lambda e non può essere configurato dall'utente. Tutti i segmenti generati dalle chiamate di funzione vengono registrati nello stesso account della funzione Lambda. Il daemon non può essere configurato per reindirizzarli a nessun altro account.

Per maggiori informazioni, consulta la sezione sul daemon X-Ray nella Guida per sviluppatori X-Ray.

Abilitazione del tracciamento attivo con l'API Lambda

Per gestire la configurazione di tracciamento con AWS CLI o AWS SDK, utilizza le seguenti operazioni API:

- UpdateFunctionConfigurazione
- GetFunctionConfigurazione
- CreateFunction

Il AWS CLI comando di esempio seguente abilita il tracciamento attivo 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 del tracciamento attivo con AWS CloudFormation

Per attivare il tracciamento su una `AWS::Lambda::Function` risorsa in un AWS 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), utilizzate la `Tracing` proprietà.

Example template.yml – Configurazione del 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 i parametri e i log delle prestazioni di runtime della funzione 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

CloudWatch Lambda Insights è una soluzione di monitoraggio e risoluzione dei problemi per 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 metrica incorporata per estrarre le metriche dagli eventi di 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 CloudWatch i prezzi di Amazon.

Runtime supportati

È possibile utilizzare Lambda Insights con qualsiasi tempo di esecuzione 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 si abilita Lambda Insights su una funzione nella console Lambda per un tempo di esecuzione supportato, Lambda aggiunge l'estensione Lambda Insights come livello alla funzione e verifica o prova ad allegare la policy `CloudWatchLambdaInsightsExecutionRolePolicy` al ruolo di esecuzione della funzione.

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. Nel menu a sinistra, scegli Strumenti operativi e di monitoraggio.
5. Nel riquadro Strumenti di monitoraggio aggiuntivi, scegli Modifica.
6. In CloudWatch Lambda Insights, attiva il monitoraggio avanzato.
7. Selezionare 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 la. AWS CloudFormation AWS Cloud Development

Kit (AWS CDK)Quando abiliti Lambda Insights a livello di codice su una funzione per un runtime supportato, associa la `CloudWatchLambdaInsightsExecutionRolePolicy`policy al CloudWatch ruolo di esecuzione della funzione.

Per ulteriori informazioni, consulta la sezione Guida introduttiva a Lambda Insights nella Amazon CloudWatch User Guide.

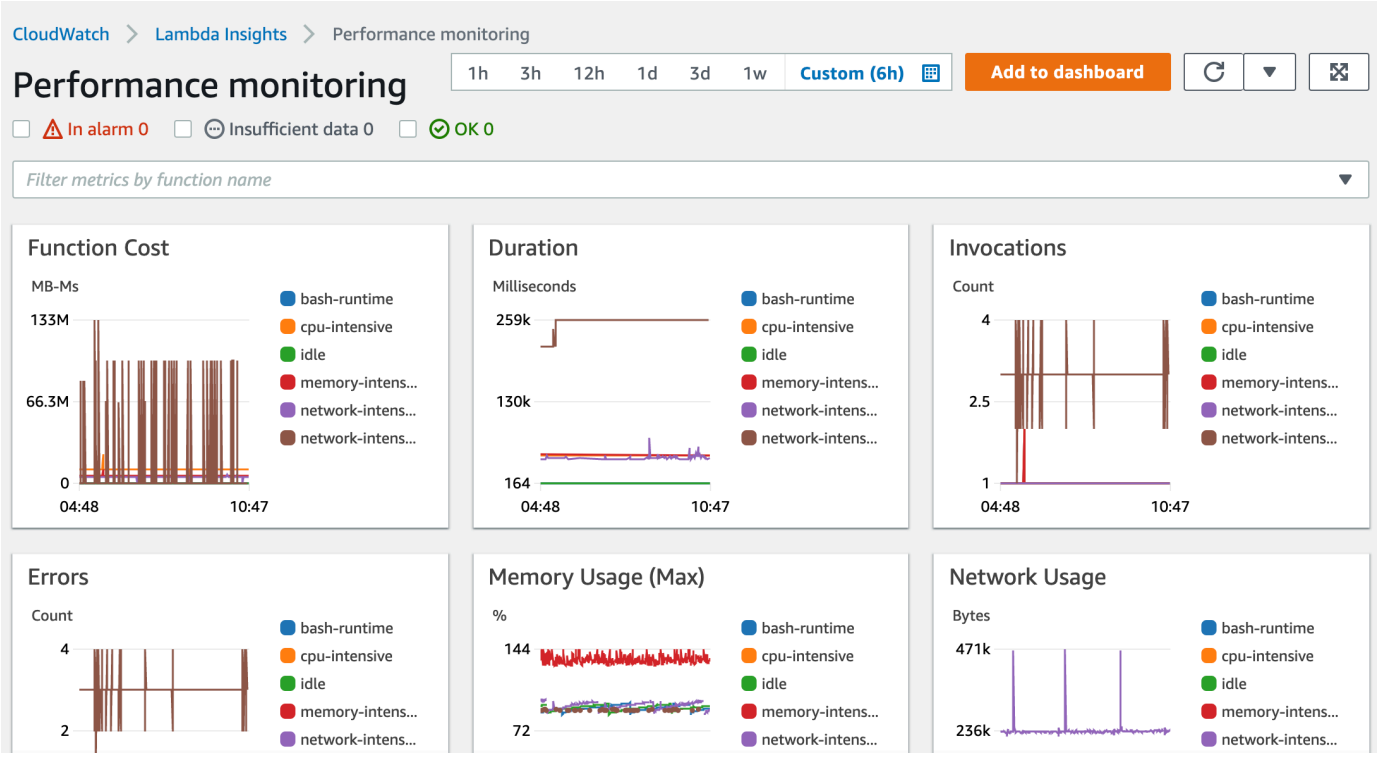
Uso del pannello di controllo di Lambda Insights

La dashboard Lambda Insights ha due visualizzazioni nella CloudWatch console: la panoramica multifunzione e la visualizzazione 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 tempo di esecuzione disponibili per una singola funzione Lambda.

Puoi utilizzare la panoramica multifunzione della dashboard di Lambda Insights nella CloudWatch console per identificare le funzioni Lambda sovrautilizzate e sottoutilizzate. Puoi utilizzare la visualizzazione a funzione singola del dashboard di Lambda Insights nella CloudWatch console per risolvere i problemi delle singole richieste.

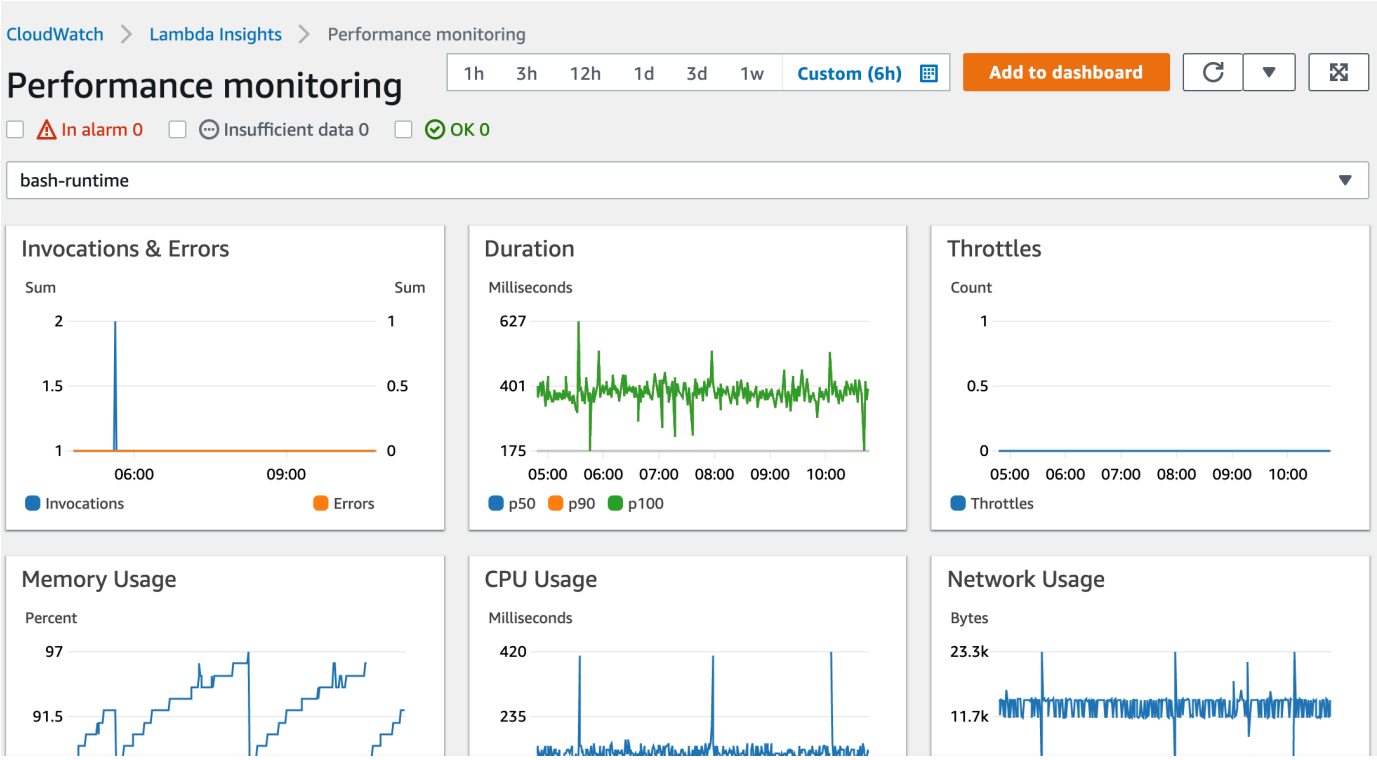
Per visualizzare i parametri di tempo di esecuzione per tutte le funzioni

1. Apri la pagina Multifunzione 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 dashboard CloudWatch .



Per visualizzare le metriche di tempo di esecuzione di una singola funzione

1. Apri la pagina a funzione singola 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 dashboard CloudWatch .



Per ulteriori informazioni, consulta Creazione e utilizzo dei widget nelle dashboard. CloudWatch

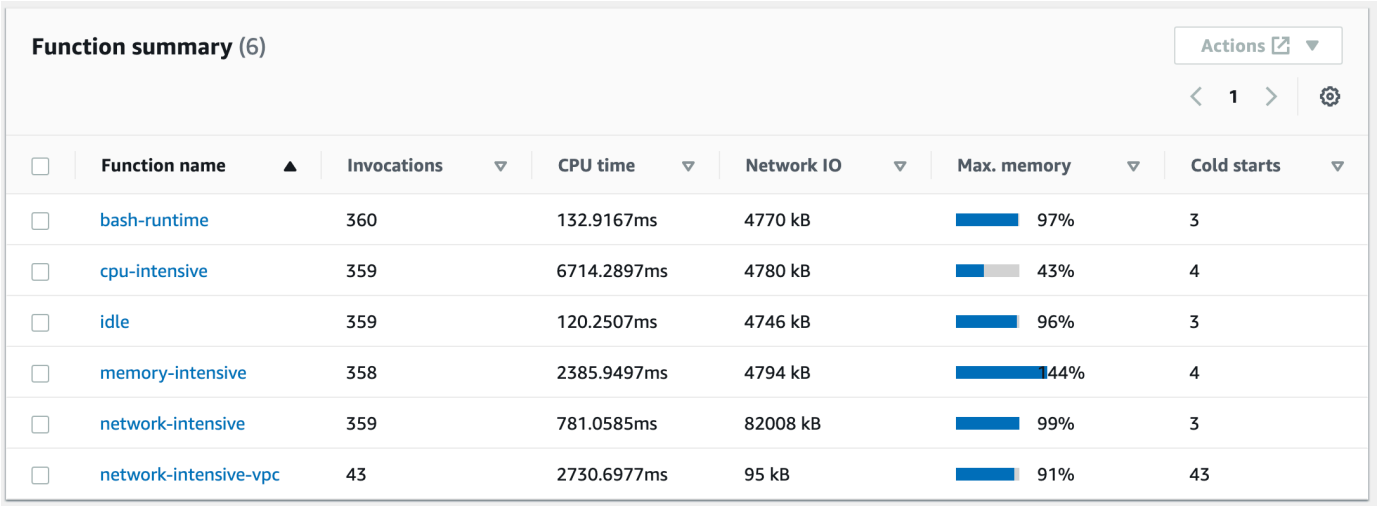
Esempio di flusso di lavoro per rilevare anomalie delle funzioni

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.

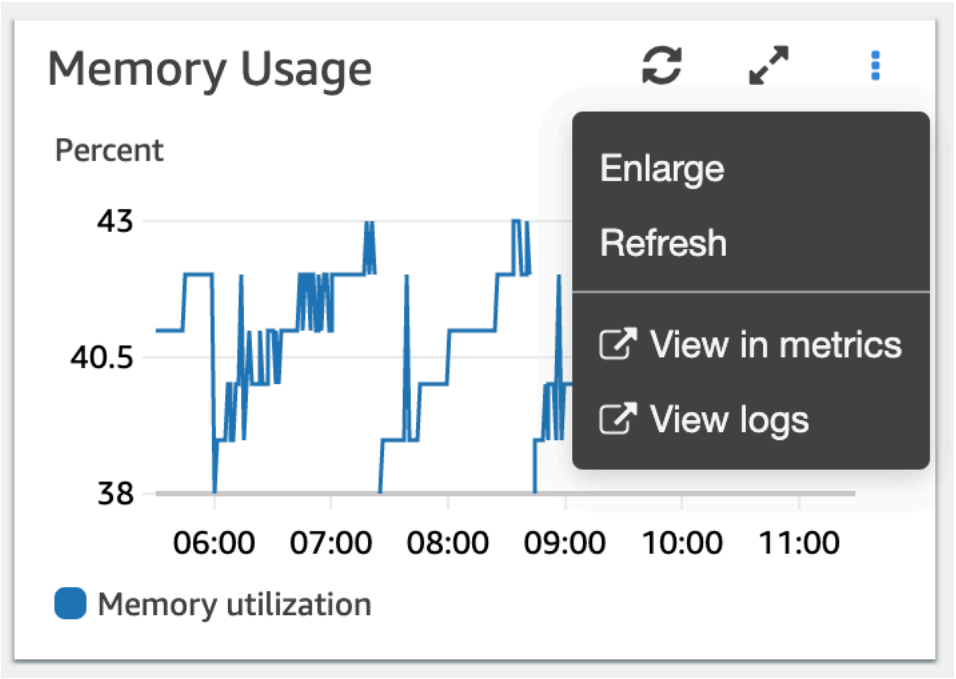
Per abilitare il rilevamento delle anomalie per una funzione

1. Apri la pagina Multifunzione nella console. CloudWatch
2. In Riepilogo funzione, scegliere il nome della funzione.

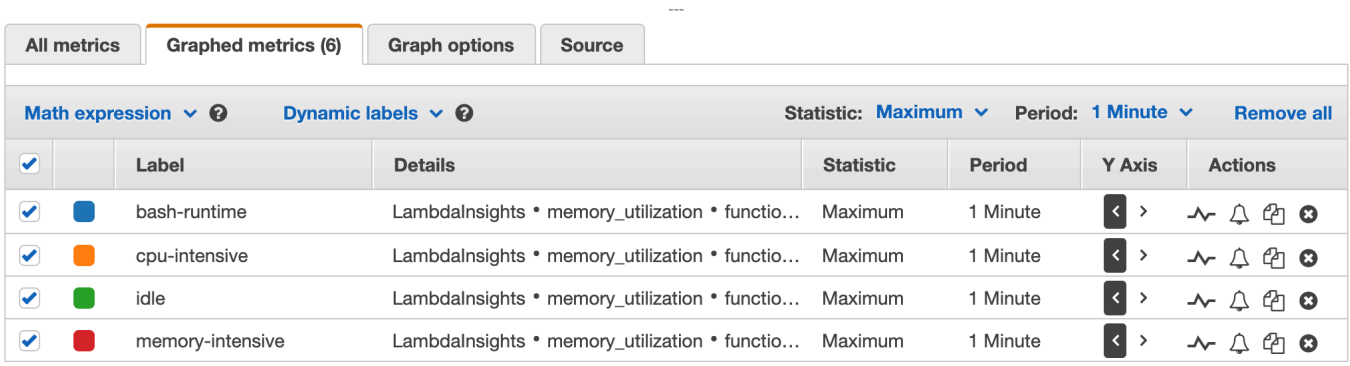
   La vista a funzione singola si apre con le metriche di tempo di esecuzione della funzione.



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.



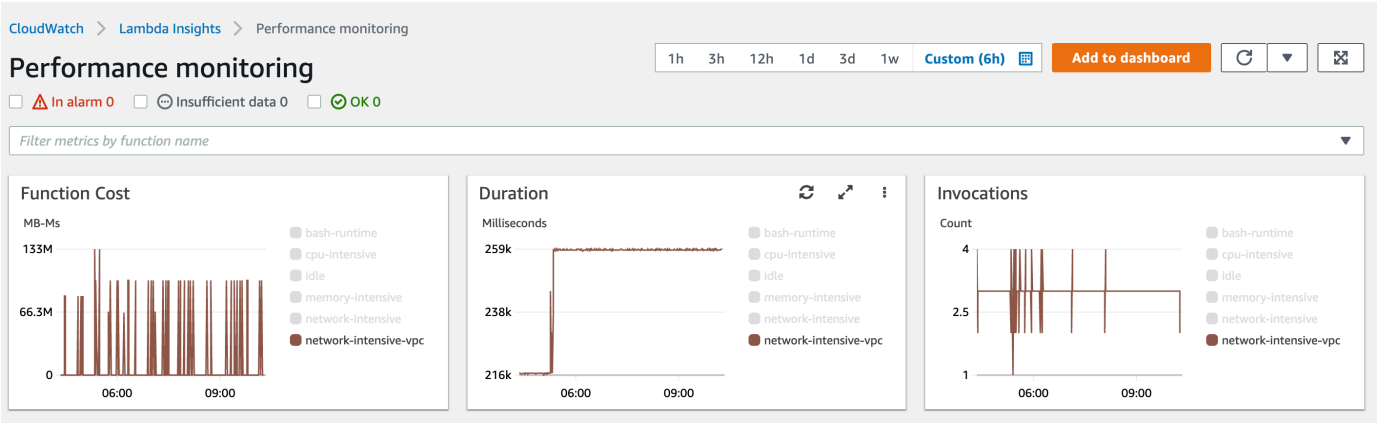
Per ulteriori informazioni, vedere Utilizzo del rilevamento CloudWatch delle anomalie.

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.

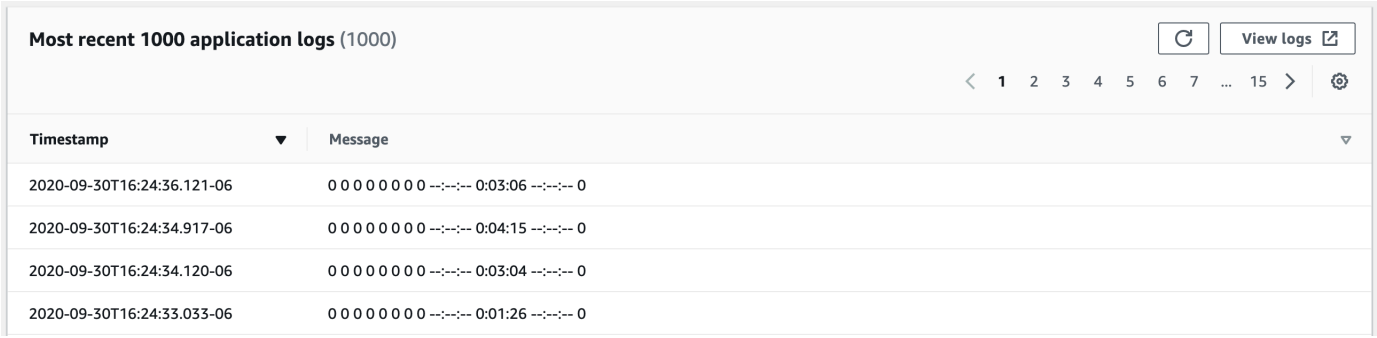
Per eseguire query su una funzione

1. Apri la pagina Multifunzione nella CloudWatch console.
2. Nel riquadro Durata scegliere la funzione per filtrare le metriche della durata.

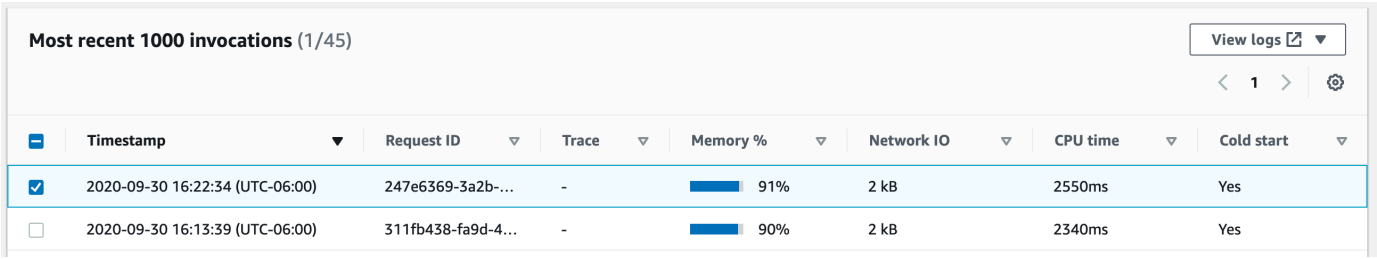


3. Aprire la pagina Funzione singola.
4. Scegli l'elenco a discesa Filtra metriche per nome funzione, quindi scegli la funzione.

5. Per visualizzare i 1000 registri applicazioni più recenti, scegliere la scheda Registri applicazioni.
6. Esaminare il timestamp e il messaggio per identificare la richiesta di chiamata per cui si desidera risolvere i problemi.

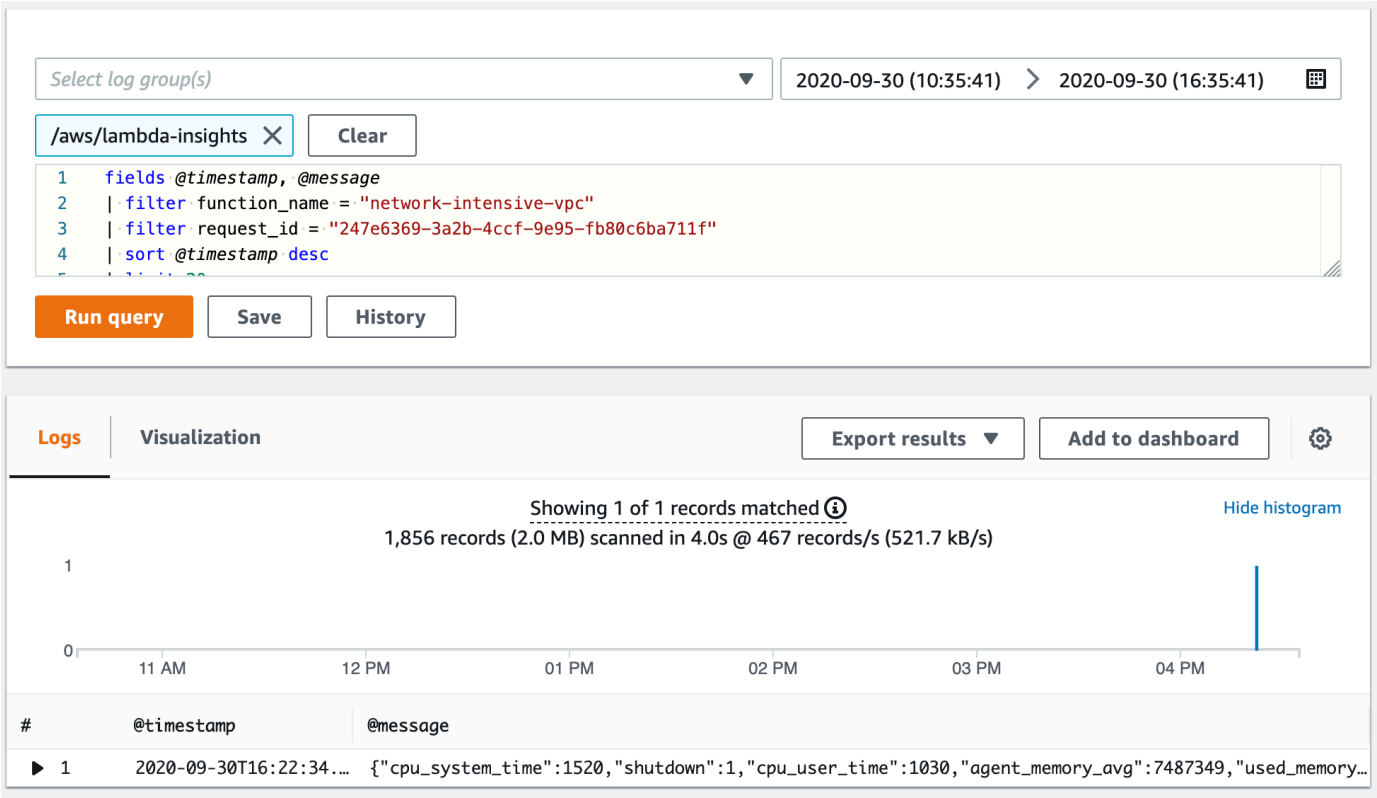


7. Per visualizzare le 1000 invocazioni più recenti, scegliere la scheda Invocazioni .
8. Selezionare il timestamp o il messaggio per la richiesta di invocazione per cui si desidera risolvere i problemi.



9. 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) .
10. Scegliere Esegui query per generare un messaggio Log per la richiesta di chiamata.



Fasi successive

- Scopri come creare un pannello di controllo CloudWatch dei log nella sezione Create a Dashboard della Amazon CloudWatch User Guide.
- Scopri come aggiungere query a una dashboard di CloudWatch Logs in Add Query to Dashboard o Export Query Results nella Amazon CloudWatch User Guide.

Usare CodeGuru Profiler con la funzione Lambda

Puoi usare Amazon CodeGuru Profiler per ottenere informazioni dettagliate sulle prestazioni di runtime delle tue funzioni Lambda. Questa pagina descrive come attivare CodeGuru Profiler dalla console Lambda.

Sections

Runtime supportati

Puoi attivare CodeGuru Profiler dalla console Lambda se il runtime della tua funzione è Python3.8, Python3.9, Java 8 con Amazon Linux 2, Java 11 o Java 17. Per versioni di runtime aggiuntive, puoi attivare Profiler manualmente. CodeGuru

- Per i runtime Java, consulta Profilazione delle applicazioni Java in esecuzione su AWS Lambda.
- Per i runtime Python, consulta Profilazione delle applicazioni Python in esecuzione su AWS Lambda.

> **Note**
>
> CodeGuru Attualmente Profiler supporta solo funzioni che utilizzano l'architettura x86_64.

Attivazione di CodeGuru Profiler dalla console Lambda

Questa sezione descrive come attivare CodeGuru Profiler dalla console Lambda.

Per attivare CodeGuru Profiler dalla console Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere la funzione.
3. Scegli la scheda Configurazione.

4. Nel riquadro Monitoring and operations tools (Strumenti di monitoraggio e operazioni) scegliere Modifica.
5. In Amazon CodeGuru Profiler, attiva la profilazione del codice.
6. Selezionare Salva.

Dopo l'attivazione, crea CodeGuru automaticamente un gruppo di profiler con il nome. `aws-lambda-<your-function-name>` È possibile modificare il nome dalla CodeGuru console.

Cosa succede quando attivi CodeGuru Profiler dalla console Lambda?

Quando attivi CodeGuru Profiler dalla console, Lambda esegue automaticamente le seguenti operazioni per tuo conto:

- Lambda aggiunge un livello CodeGuru Profiler alla tua funzione. Per ulteriori dettagli, consulta Use AWS Lambda layers nella Amazon CodeGuru Profiler User Guide.
- Lambda aggiunge anche variabili di ambiente alla tua funzione. Il valore esatto varia in base al runtime.

  Variabili di ambiente

Runtime	Chiave	Valore
java8.al2, java11	`JAVA_TOOL_OPTIONS`	`-javaagent:/opt/codeguru-profiler-java-agent-standalone.jar`
python3.8, python3.9	`AWS_LAMBDA_EXEC_WRAPPER`	`/opt/codeguru_profiler_lambda_exec`

- Lambda aggiunge la policy `AmazonCodeGuruProfilerAgentAccess` al ruolo di esecuzione della tua funzione.

> **Note**
>
> Quando disattivi CodeGuru Profiler dalla console, Lambda rimuove automaticamente il livello CodeGuru Profiler dalla tua funzione. Tuttavia, Lambda non rimuove le variabili di ambiente per la policy `AmazonCodeGuruProfilerAgentAccess` dal ruolo di esecuzione.

Fasi successive

- Scopri di più sui dati raccolti dal tuo gruppo di profiler nella sezione Lavorare con le visualizzazioni nella Amazon CodeGuru Profiler User Guide.

Flussi di lavoro di esempio che utilizzano altri servizi AWS

AWS Lambda si integra con altri servizi AWS per aiutarti a monitorare, tracciare, eseguire il debug e risolvere i problemi relativi alle tue funzioni Lambda. Questa pagina indica i flussi di lavoro che puoi utilizzare con AWS X-Ray e AWS Trusted Advisor per tracciare le funzioni Lambda e risolvere i relativi problemi.

Sections

Prerequisiti

Nella sezione seguente vengono descritti i passaggi per utilizzare AWS X-Ray e Trusted Advisor per risolvere i problemi relativi alle funzioni Lambda.

Uso di AWS X-Ray

AWS X-Ray deve essere abilitato nella console Lambda per completare i flussi di lavoro di AWS X-Ray in questa pagina. Se il tuo ruolo di esecuzione non dispone delle autorizzazioni richieste, la console Lambda proverà ad aggiungerli.

Per attivare AWS X-Ray nella console Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere la funzione.
3. Scegli la scheda Configurazione.
4. Nel riquadro Strumenti di monitoraggio scegliere Modifica.
5. In AWS X-Ray, attiva Tracciamento attivo.
6. Selezionare Salva.

Uso di AWS Trusted Advisor

AWS Trusted Advisor analizza l'ambiente AWS e fornisce suggerimenti su come risparmiare denaro, migliorare la disponibilità e le prestazioni del sistema e colmare le lacune legate alla sicurezza. Puoi utilizzare controlli Trusted Advisor per valutare le funzioni e applicazioni Lambda nel tuo account AWS. I controlli forniscono le misure consigliate da adottare e le risorse per ulteriori informazioni.

- Per ulteriori informazioni sui piani di supporto Trusted Advisor per AWS controlli, vedere Piani di supporto.
- Per ulteriori informazioni sui controlli per Lambda, vedere elenco di controllo delle best practice di AWS Trusted Advisor.
- Per ulteriori informazioni su come utilizzare la console Trusted Advisor, vedere Nozioni di base su AWS Trusted Advisor .
- Per istruzioni su come consentire e negare l'accesso alla console a Trusted Advisor, vedere Esempi di policy IAM.

Prezzi

- Con AWS X-Ray i prezzi sono calcolati solo in base al numero di tracce registrate, recuperate e scansionate. Per ulteriori informazioni, consulta la sezione Prezzi di AWS X-Ray.
- I controlli di ottimizzazione dei costi Trusted Advisor sono inclusi nelle iscrizioni al supporto AWS Business ed Enterprise. Per ulteriori informazioni, consulta la sezione Prezzi di AWS Trusted Advisor.

Esempio di flusso di lavoro AWS X-Ray per visualizzare una mappa del tracciamento

Se hai abilitatoAWS X-Ray, puoi visualizzare una mappa di tracciamento sulla CloudWatch console. Una mappa del tracciamento visualizza gli endpoint e le risorse del servizio come nodi ed evidenzia il traffico, la latenza e gli errori per ciascun nodo e le relative connessioni.

Puoi scegliere un nodo per visualizzare informazioni dettagliate sui parametri, i log e le tracce correlati associati a tale parte del servizio. Ciò consente di esaminare i problemi e il loro effetto sull'applicazione.

Per visualizzare la mappa e le tracce di traccia utilizzando la CloudWatch console

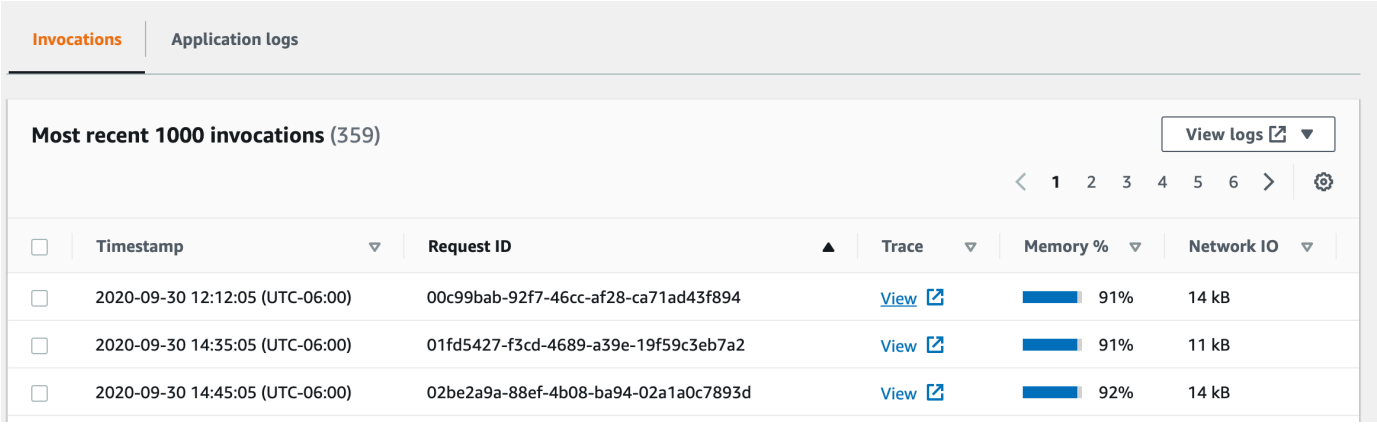
1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Selezionare Monitoring (Monitoraggio).
4. Seleziona Visualizza tracce X-Ray.
5. Scegli Mappa del tracciamento in Tracce di X-Ray dal riquadro di navigazione a sinistra.
6. Scegli tra gli intervalli di tempo predefiniti oppure scegli un intervallo di tempo personalizzato.
7. Per risolvere i problemi relativi alle richieste, scegli un filtro.

Flusso di lavoro di AWS X-Ray di esempio per visualizzare i dettagli di traccia

Se hai abilitatoAWS X-Ray, puoi utilizzare la visualizzazione a funzione singola sulla dashboard di CloudWatch Lambda Insights per mostrare i dati di traccia distribuiti di un errore di invocazione della funzione. Ad esempio, se il messaggio dei log dell'applicazione indica un errore, puoi aprire la mappa del tracciamento per visualizzare i dati di tracciamento distribuiti e gli altri servizi che gestiscono la transazione.

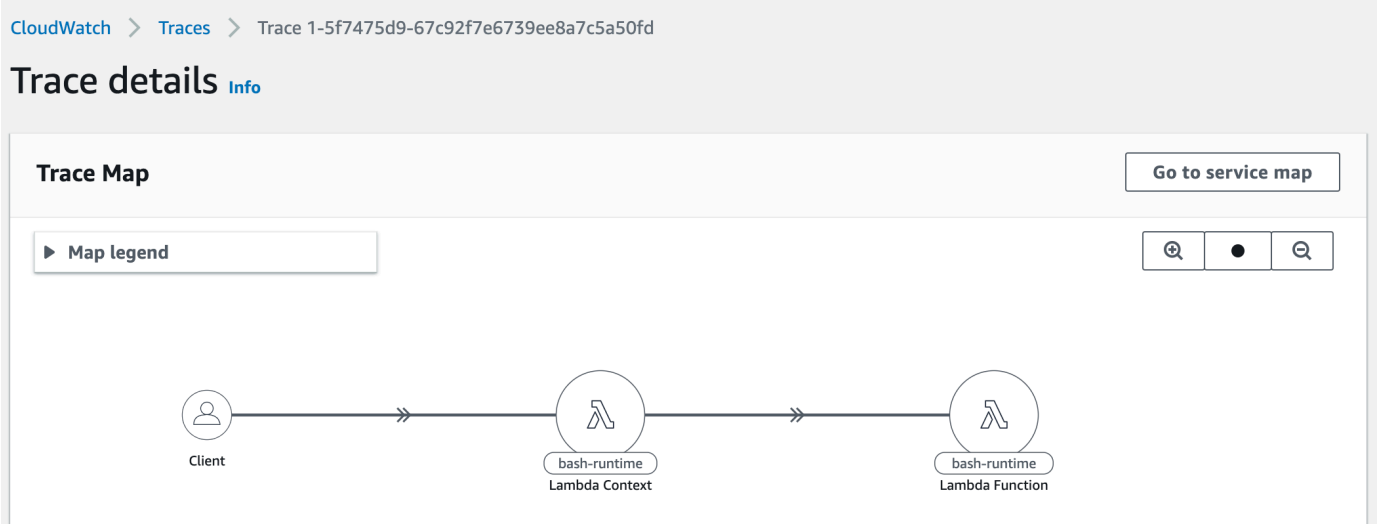
Per visualizzare i dettagli di traccia di una funzione

1. Apri la visualizzazione a funzione singola nella console. CloudWatch
2. Seleziona la scheda Log delle applicazioni.
3. Utilizza il timestamp o il messaggio per identificare la richiesta di chiamata per cui si desidera risolvere i problemi.
4. Per visualizzare le 1000 invocazioni più recenti, scegliere la scheda Invocazioni .



5. Scegli la colonna ID richiesta per ordinare le voci in ordine alfabetico crescente.
6. Nella colonna Traccia seleziona Visualizza.

   La pagina Dettagli del tracciamento si apre nella vista della mappa del tracciamento.



Flusso di lavoro di AWS Trusted Advisor di esempio per visualizzare le raccomandazioni

Trusted Advisor controlla le funzioni Lambda in tutte le regioni AWS per identificare le funzioni con il massimo potenziale di risparmio sui costi e fornire raccomandazioni utili per l'ottimizzazione. Analizza i tuoi dati di utilizzo Lambda quali tempo di esecuzione delle funzioni, durata fatturata, memoria utilizzata, memoria configurata, configurazione del timeout ed errori.

Ad esempio, il controllo Funzioni Lambda con tasso di errore elevato consiglia di utilizzare AWS X-Ray o di CloudWatch rilevare errori con le funzioni Lambda.

Per verificare la presenza di funzioni con elevati tassi di errore

1. Aprire la console Trusted Advisor.
2. Scegli la categoria Ottimizzazione dei costi.
3. Scorri verso il basso fino a Funzioni AWS Lambda con elevati tassi di errore . Espandi la sezione per visualizzare i risultati e le azioni consigliate.

Fasi successive

- Ulteriori informazioni su come integrare tracce, parametri, log e allarmi sono disponibili in Uso della mappa del tracciamento X-Ray.
- Scopri di più su come ottenere un elenco di Trusted Advisor controlli in Utilizzo di Trusted Advisor come servizio web .

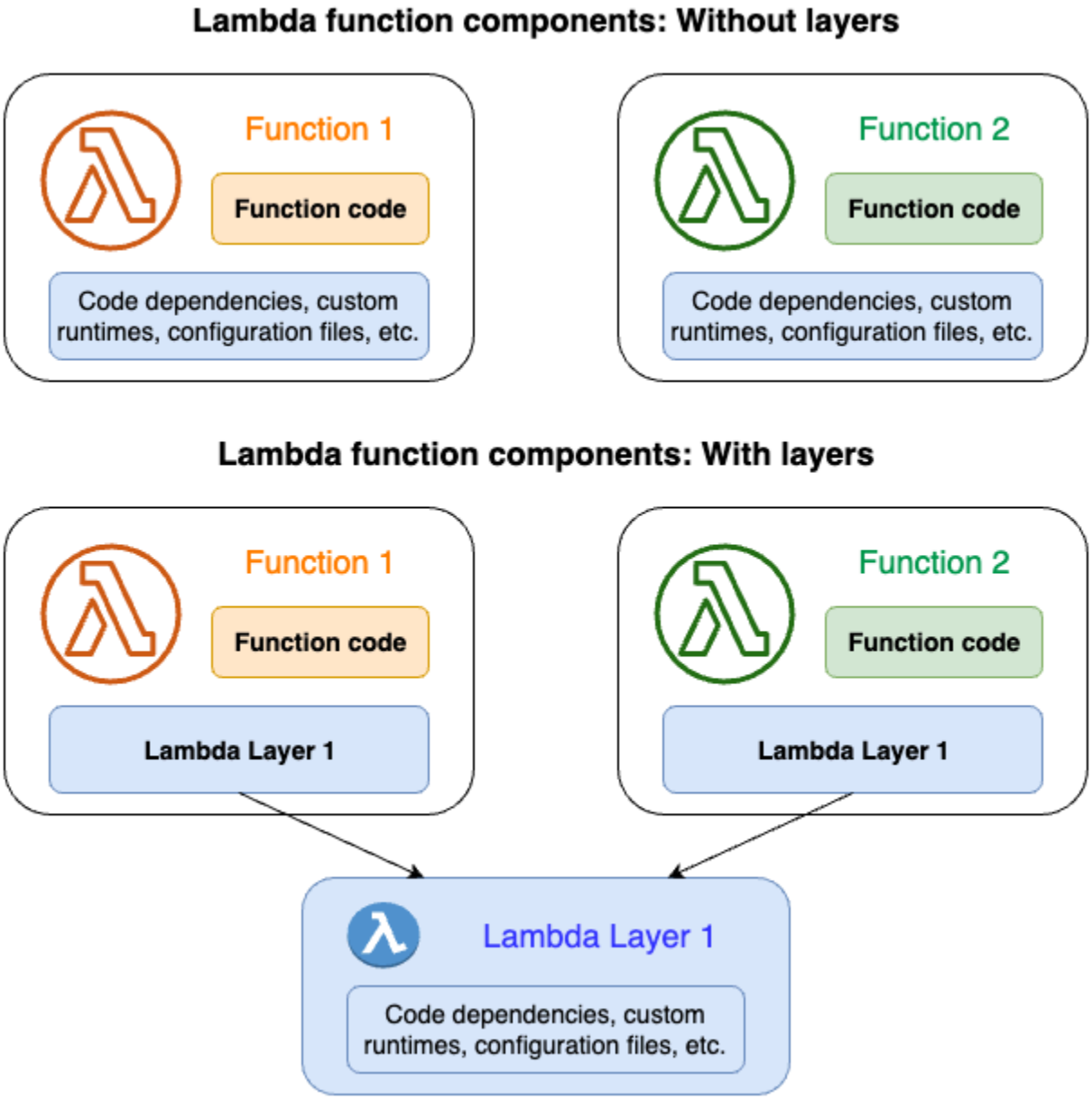
Gestione delle dipendenze Lambda con livelli

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 la dimensione del pacchetto di implementazione è troppo grande. L'uso dei livelli riduce le dimensioni del pacchetto e può sbloccare l'utilizzo dell'editor di codice.

Il diagramma seguente illustra le principali differenze di architettura tra due funzioni che condividono dipendenze. Una utilizza i livelli Lambda e l'altra no.



Quando si include un livello in una funzione Lambda, Lambda estrae il contenuto del livello nella directory `/opt` nell'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 AWS Compute.

Argomenti

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 la versione esatta del livello che si desidera utilizzare.

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 Livelli Lambda.

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 layer funzioni correttamente in un ambiente Linux, consigliamo di creare il contenuto del layer utilizzando uno strumento come Docker o AWS Cloud9. AWS Cloud9 è un ambiente di sviluppo integrato (IDE) basato su cloud che fornisce l'accesso integrato a un server Linux per l'esecuzione e il test del codice. Per ulteriori informazioni, consulta Utilizzo dei livelli Lambda per semplificare il processo di sviluppo nel Compute Blog AWS .

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 `PATH` variabile raccolga il contenuto del layer, il file layer.zip dovrebbe avere le sue dipendenze nei seguenti percorsi di cartella:

Percorsi dei livelli per ciascun runtime Lambda

Runtime	Path
Node.js	`nodejs/node_modules`
	`nodejs/node14/node_modules` (`NODE_PATH`)
	`nodejs/node16/node_modules` (`NODE_PATH`)
	`nodejs/node18/node_modules` (`NODE_PATH`)

Runtime	Path
Python	`python`
	`python/lib/` *`python3.x`* `/site-packages` (directory del sito)
Java	`java/lib` (`CLASSPATH`)
Ruby	`ruby/gems/3.2.0` (`GEM_PATH`)
	`ruby/lib` (`RUBYLIB`)
Tutti i runtime	`bin` (`PATH`)
	`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 per Node.js AWS X-Ray

```
xray-sdk.zip
# nodejs/node_modules/aws-xray-sdk
```

Python

Example struttura di file per la libreria Requests

```
layer_content.zip
# python
    # lib
        # python3.11
            # site-packages
                # requests
                # <other_dependencies> (i.e. dependencies of the requests package)
                # ...
```

Ruby

Example struttura dei file per gem JSON

```
json.zip
# ruby/gems/2.7.0/
               | 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 sulla lingua sulla creazione, la creazione e l'aggiunta di un layer, consultate le pagine seguenti:

- Python – the section called “Livelli”
- Java — the section called “Livelli”

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 Livelli Lambda.

Dopo aver creato i pacchetti del contenuto dei livelli, il passaggio successivo consiste nel creare il livello 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 AWS CloudFormation”. Per creare un livello utilizzando AWS Serverless Application Model (AWS SAM), consulta the section called “Strati con AWS SAM”.

Argomenti

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). 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. (Facoltativo) Per Architetture compatibili, scegli un valore o entrambi i valori. Per ulteriori informazioni, consulta the section called “Set di istruzioni (ARM/x86)”.
7. (Facoltativo) Per Runtime compatibili, scegli i runtime con cui il tuo livello è compatibile.
8. (Facoltativo) Per Licenza, inserisci tutte le informazioni necessarie sulla licenza.
9. Scegli Crea.

In alternativa, puoi anche utilizzare l'PublishLayerVersionAPI per creare un livello. Ad esempio, puoi usare il comando `publish-layer-version` di AWS Command Line Interface (CLI) con un nome, una descrizione e un archivio di file .zip specificati. Le informazioni sulla licenza, i runtime compatibili e i parametri dell'architettura compatibile sono opzionali.

```
aws lambda publish-layer-version --layer-name my-layer \
    --description "My layer" \
    --license-info "MIT" \
    --zip-file fileb://layer.zip \
    --compatible-runtimes python3.10 python3.11 \
    --compatible-architectures "arm64" "x86_64"
```

Verrà visualizzato un output simile al seguente:

```
{
    "Content": {
        "Location": "https://awslambda-us-east-2-layers.s3.us-east-2.amazonaws.com/
snapshots/123456789012/my-layer-4aaa2fbb-ff77-4b0a-ad92-5b78a716a96a?
versionId=27iWyA73cCAYqyH...",
        "CodeSha256": "tv9jJO+rPbXUUXuRKi7CwHzKtLDkDRJLB3cC3Z/ouXo=",
        "CodeSize": 169
    },
    "LayerArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer",
    "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer:1",
    "Description": "My layer",
    "CreatedDate": "2023-11-14T23:03:52.894+0000",
    "Version": 1,
    "CompatibleArchitectures": [
        "arm64",
```

```
        "x86_64"
     ],
    "LicenseInfo": "MIT",
    "CompatibleRuntimes": [
        "python3.10",
        "python3.11"
    ]
}
```

Ogni volta che si richiama `publish-layer-version`, viene creata una nuova versione del livello.

Eliminazione della versione di un livello

Per eliminare una versione di livello, utilizzate l'DeleteLayerVersionAPI. Ad esempio, puoi usare il comando `delete-layer-version` della CLI con il nome e la versione del livello specificati.

```
aws lambda delete-layer-version --layer-name my-layer --version-number 1
```

Quando elimini la versione di un livello, non puoi più configurare una funzione Lambda per utilizzarla. Tuttavia, qualsiasi funzione che già utilizza la versione continua ad averne accesso. Inoltre, Lambda non riutilizza mai i numeri di versione per il nome di un livello.

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 Livelli Lambda.

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'GetLayerVersionAPI nella versione del layer. Per le funzioni dell'Account AWS, è necessario disporre di questa autorizzazione nella policy 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 “Concessione dell'accesso ai livelli ad 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 livello che hai già aggiunto, anche dopo che la versione del livello è stata eliminata o dopo la revoca del tuo permesso di accesso al livello. Non è tuttavia possibile creare una nuova funzione che utilizza la versione di un livello eliminato.

> **Note**
>
> Assicurati che i livelli aggiunti a una funzione siano compatibili con il runtime e l'architettura del set di istruzioni della funzione.

Aggiunta di un livello a una funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione da configurare.
3. In Layers (Livelli), scegli Add a layer (Aggiungi un livello)

4. In Scegli un livello, scegli un'origine del livello:

    a. Per le origini dei livelli AWS o dei livelli personalizzati, scegli un livello dal menu a discesa. In Version (Versione), scegli una versione del livello dal menu a discesa.

    b. Per l'origine dei livelli Specifica un ARN, inserisci un ARN nella casella di testo e scegli Verifica. Quindi scegli 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. Selezionare 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 dei livelli delle funzioni negli account AWS.

Argomenti

Accesso al contenuto del livello dalla funzione

Se la tua funzione Lambda include livelli, Lambda estrae il contenuto del livello nella directory `/opt` nell'ambiente di esecuzione della funzione. Lambda estrae i livelli nell'ordine (dal basso verso l'alto) indicato dalla funzione. Lambda unisce le cartelle con lo stesso nome. Se lo stesso file viene visualizzato in più livelli, la funzione utilizza la versione dell'ultimo livello estratto.

Ogni runtime Lambda aggiunge cartelle di directory `/opt` specifiche alla variabile `PATH`. Il codice funzione può accedere al contenuto del livello senza dover specificare il percorso. Per ulteriori informazioni sulle impostazioni del percorso nell'ambiente di esecuzione Lambda, consulta the section called “Variabili di ambiente di runtime definite”.

Fai riferimento a the section called “Percorsi dei livelli per ciascun runtime Lambda” per sapere dove includere le librerie durante la creazione di un livello.

Se utilizzi un runtime Node.js o Python, puoi utilizzare l'editor di codice integrato nella console Lambda. Dovresti essere in grado di importare qualsiasi libreria che hai aggiunto come livello alla funzione corrente.

Ricerca di informazioni sul livello

Per trovare livelli nel tuo account compatibili con il runtime della tua funzione, usa l'ListLayersAPI. Ad esempio, puoi utilizzare il seguente comando `list-layers` dell'AWS Command Line Interface (CLI):

```
aws lambda list-layers --compatible-runtime python3.9
```

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",
                "CreatedDate": "2023-11-15T00:37:46.592+0000",
                "CompatibleRuntimes": [
                    "python3.9",
                    "python3.10",
                    "python3.11",
                ]
            }
        }
    ]
}
```

Per elencare tutti i livelli nell'account, ometti l'opzione `--compatible-runtime`. I dettagli della risposta mostrano la versione più recente di ogni livello.

Puoi anche ottenere la versione più recente di un layer utilizzando l'ListLayerVersionsAPI. 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",
            "CreatedDate": "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",
            "CreatedDate": "2023-11-15T00:27:46.592+0000",
            "CompatibleRuntimes": [
                "java11"
            ]
        }
    ]
}
```

Utilizzo AWS CloudFormation con livelli

Puoi usarlo AWS CloudFormation per creare un livello e associarlo 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).

```
---
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: DOC-EXAMPLE-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
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambda_ReadOnlyAccess
        - AWSXrayWriteOnlyAccess
      Layers:
        - !Ref MyLambdaLayer
```

Utilizzo AWS SAM con livelli

Puoi usare il AWS Serverless Application Model (AWS SAM) per automatizzare la creazione di 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://DOC-EXAMPLE-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://DOC-EXAMPLE-BUCKET/my-function
      Layers:
        - !Ref MyLambdaLayer
```

Potenzia 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 AWS CloudFormation, AWS Serverless Application Model (AWS SAM) e Terraform.

Ti viene addebitato il tempo di esecuzione 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.

Per un tutorial sulle estensioni e su come usarle con le funzioni Lambda, consultare il Workshop sulle estensioni di AWS Lambda.

Argomenti

- Utilizzo dell'API Lambda Extensions per creare estensioni
- API di telemetria Lambda

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 Lambda 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 `beforeCheckpoint`, il codice viene eseguito alla fine della fase `Init`.
- **`Restore`**(SnapStart solo): Quando richiami una SnapStartfunzione 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 `afterRestore()`, il codice viene eseguito alla fine della fase `Restore`. Ti sarà addebitata la durata degli hook di runtime `afterRestore()`. Il runtime (JVM) deve essere caricato e gli hook di runtime `afterRestore()` devono essere completati entro il limite di timeout (10 secondi). SnapStartTimeoutExceptionAltrimenti, 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, è possibile che la durata dell'esecuzione della funzione aumenti.

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.

Autorizzazioni

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 AWS 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. L'utilizzo dei livelli consente di condividere le estensioni all'interno dell'organizzazione o all'intera community 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 Livelli Lambda.

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. Scegli 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
ADD my-extension.zip /opt
CMD python ./my-function.py
```

Passaggi successivi

Per ulteriori informazioni sulle estensioni, si consiglia di utilizzare le seguenti risorse:

- Per un esempio di lavoro di base, consulta Creazione di estensioni per AWS Lambda nel blog Compute di AWS.
- Per informazioni sulle estensioni fornite dai partner AWS Lambda, consulta Introduzione delle estensioni AWS Lambda nel blog Compute di AWS.
- Per visualizzare le estensioni di esempio e gli script wrapper disponibili, consulta AWS LambdaExtensions on the AWS Samples repository. GitHub

AWS Lambda partner di estensioni

AWS Lambda ha collaborato con diverse entità di terze parti 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 funzioni Node.js o Python Lambda, fornendo visibilità e avvisi sulle prestazioni delle funzioni.
- Check Point CloudGuard: una soluzione di runtime basata su estensioni che offre una sicurezza completa del ciclo di vita per le applicazioni serverless.
- 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 sfrutta 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 in stile iniezione, acquisizione di account tramite furto di credenziali, bot dannosi e abuso delle 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 cloud autonoma, basata su 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.
- Thundra: fornisce report di telemetria asincrona, come tracce, metriche e log.
- Salt Security: semplifica la governance della postura delle API e la sicurezza delle API per le funzioni Lambda attraverso la configurazione e il supporto automatizzati per diversi runtime.

AWS estensioni gestite

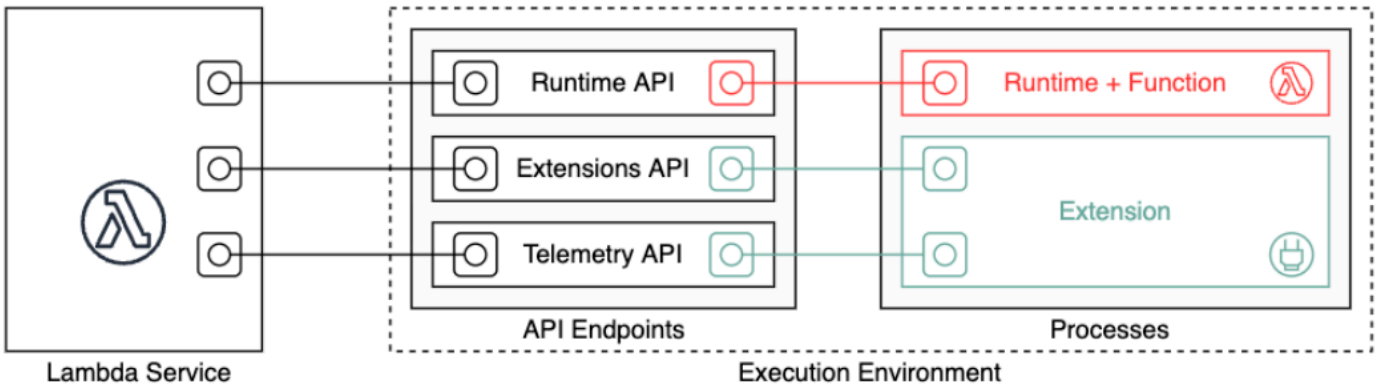
AWS fornisce le proprie estensioni gestite, tra cui:

- AWS AppConfig— Usa i flag delle funzionalità e i dati dinamici per aggiornare le 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 migliorarlo.
- CloudWatch Lambda Insights: monitora, risolvi i problemi e ottimizza le prestazioni delle funzioni Lambda tramite dashboard automatici.
- AWS Distro for OpenTelemetry (ADOT): consente alle funzioni di inviare dati di traccia a servizi di AWS monitoraggio come AWS X-Ray e a destinazioni che supportano OpenTelemetry come Honeycomb e Lightstep.
- AWS Parametri e segreti: consente ai clienti di recuperare in modo sicuro i parametri da Parameter Store e i segreti da.AWS Systems ManagerAWS Secrets Manager

Per ulteriori esempi di estensioni e progetti dimostrativi, consultare Estensioni AWS Lambda.

Utilizzo dell'API Lambda Extensions 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 utilizzare 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 esempio estensioni e script wrapper, consulta AWS Lambda Extensions on the Samples repository. AWS GitHub

Argomenti

- Ciclo di vita dell'ambiente di esecuzione Lambda
- Riferimento all'API delle estensioni

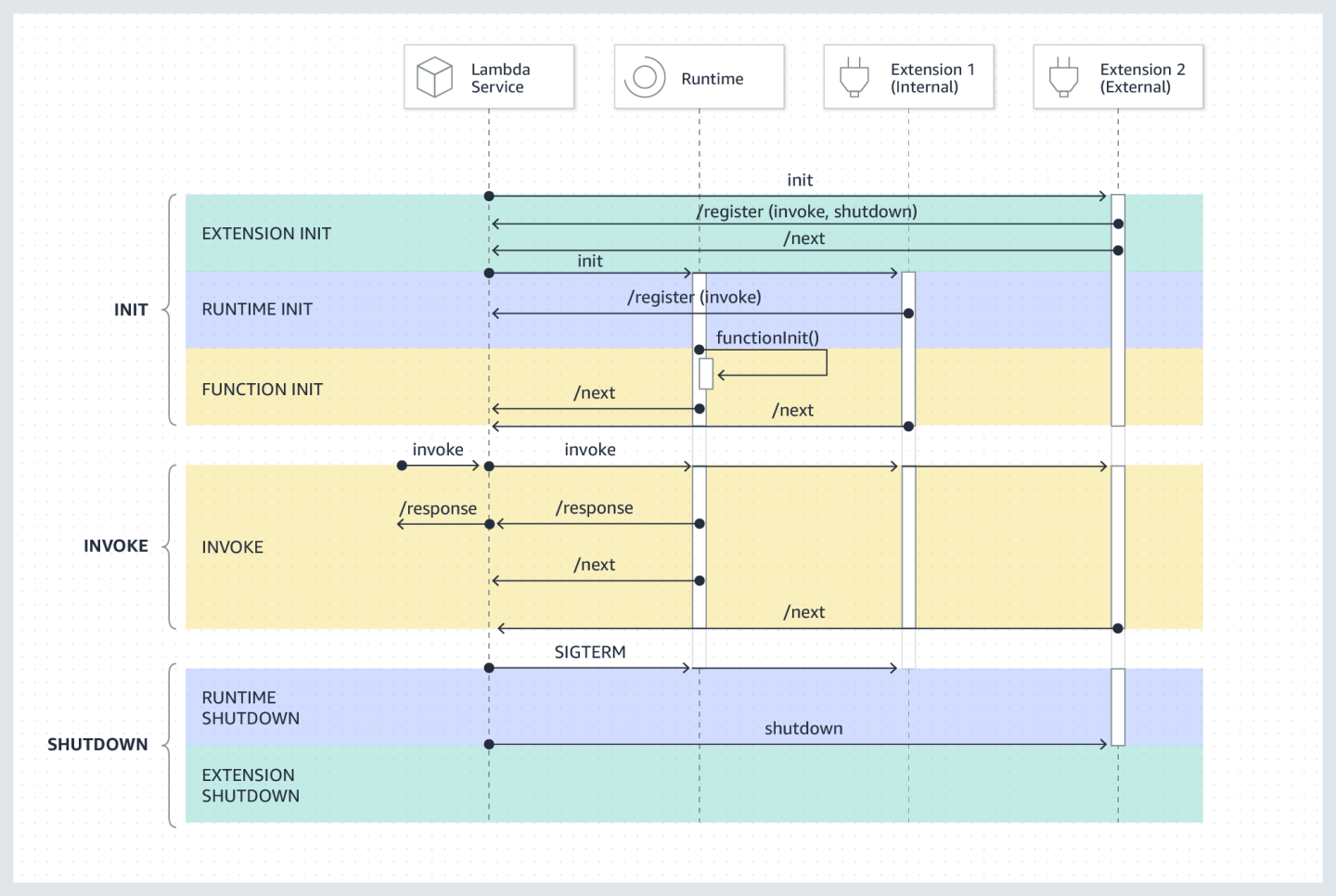
Ciclo di vita dell'ambiente di esecuzione Lambda

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 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.
- **`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

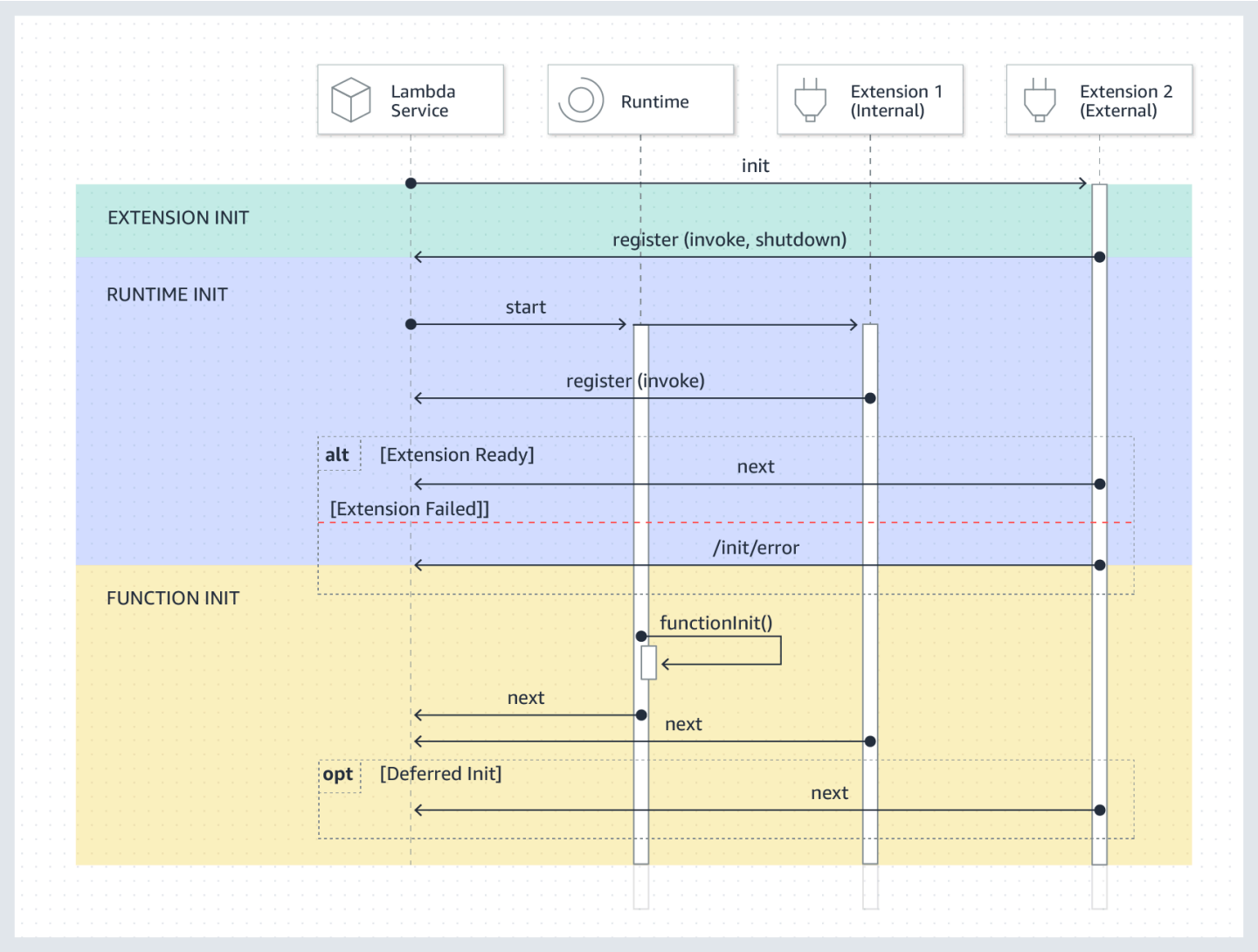
Durante la fase `Extension init`, per ricevere gli eventi ogni estensione deve registrarsi con Lambda. Lambda utilizza il nome file completo dell'estensione per confermare che l'estensione abbia completato la sequenza di bootstrap. Pertanto, ogni chiamata `Register` API deve includere l'intestazione `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`.



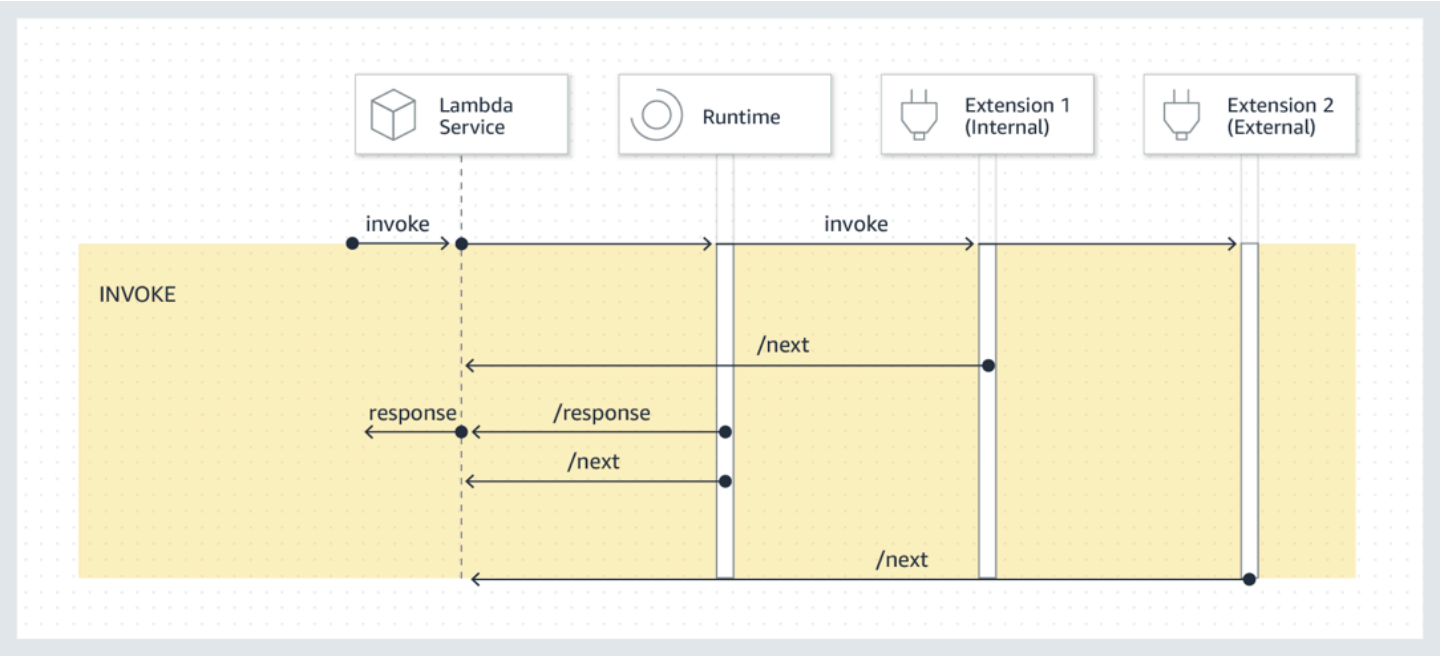
Invoca fase

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`.

Durante la chiamata, le estensioni esterne vengono eseguite in parallelo con la funzione. Continuano anche a funzionare dopo che la funzione è stata completata. In questo modo è possibile acquisire informazioni diagnostiche o inviare log, parametri e tracce in una posizione di propria 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 il tempo totale impiegato per il completamento del runtime e delle chiamate di tutte le estensioni e non viene calcolata fino al termine dell'esecuzione della funzione e di tutte le estensioni.

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 dei parametri delle funzioni 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

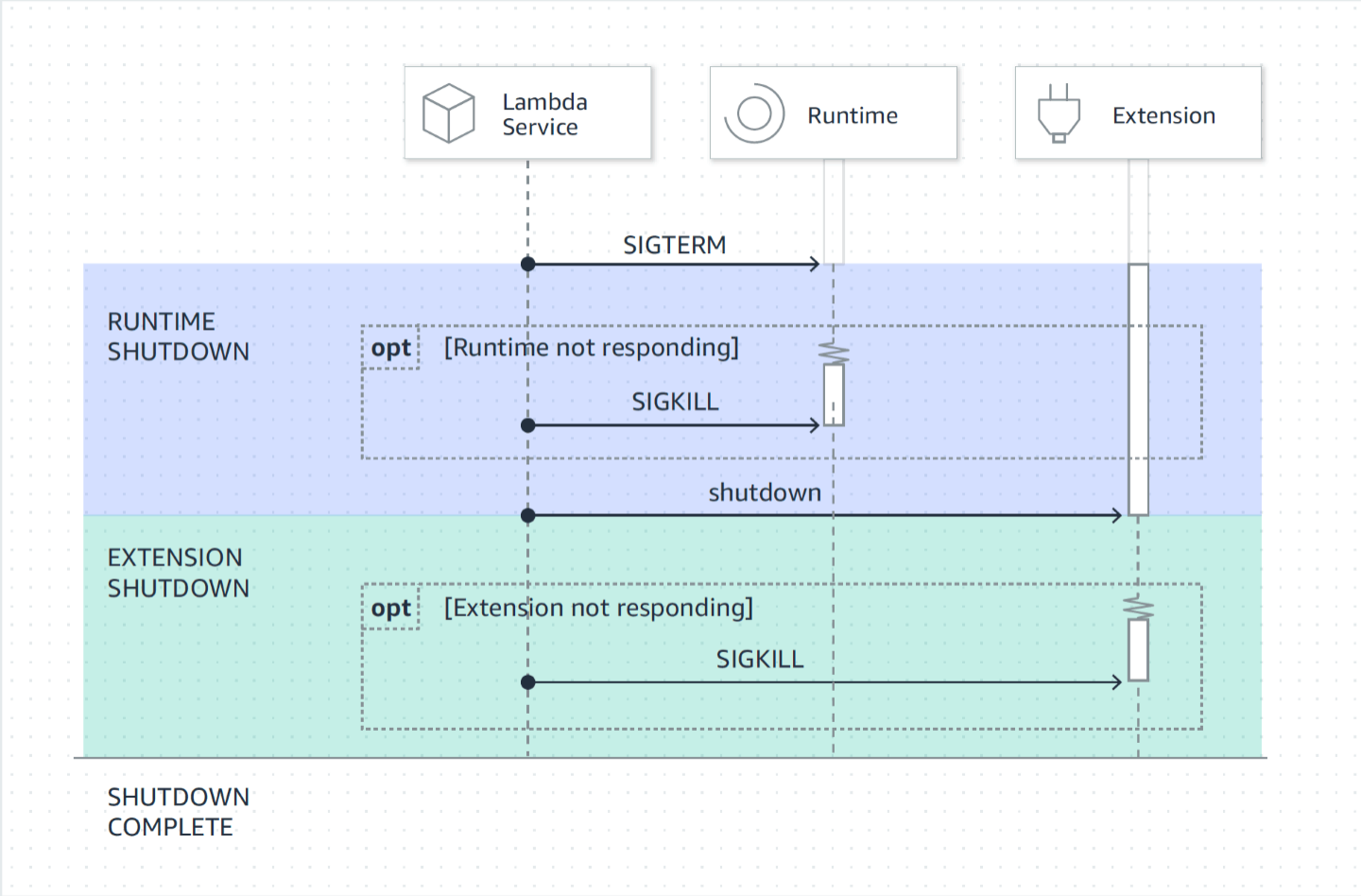
Quando Lambda sta per chiudere il runtime, invia uno `Shutdown` a ciascuna estensione esterna registrata. Le estensioni possono utilizzare questo tempo per le attività di pulizia finali. L'evento `Shutdown` viene inviato in risposta a una richiesta `Next` API.

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

Per una funzione con estensioni esterne, Lambda riserva fino a 300 ms (500 ms per un runtime con un'estensione interna) per consentire al processo di runtime di eseguire un arresto regolare. Lambda assegna la parte rimanente del limite di 2.000 ms per l'arresto delle estensioni esterne.

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:

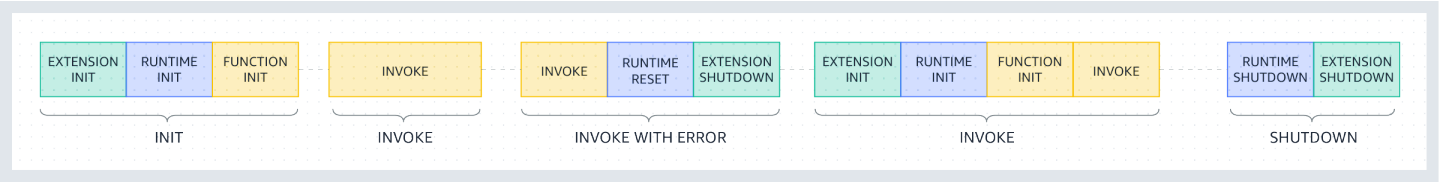
- `AWS_EXECUTION_ENV`
- `AWS_LAMBDA_LOG_GROUP_NAME`
- `AWS_LAMBDA_LOG_STREAM_NAME`
- `AWS_XRAY_CONTEXT_MISSING`
- `AWS_XRAY_DAEMON_ADDRESS`
- `LAMBDA_RUNTIME_DIR`
- `LAMBDA_TASK_ROOT`
- `_AWS_XRAY_DAEMON_ADDRESS`
- `_AWS_XRAY_DAEMON_PORT`
- `_HANDLER`

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 dell'invocazione 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.

Registri delle estensioni: Lambda invia l'output di registro delle estensioni CloudWatch a Logs. 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 alla registrazione dell'estensione conterrà l'ID dell'account associato alla funzione Lambda per cui si sta 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`.

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 del container. Stato non recuperabile. 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 del container. Stato non recuperabile. 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>. Per esempio:
  - Estensione. NoSuchGestore
  - Extension.api trovata KeyNot
  - Estensione. ConfigInvalid
  - Estensione. 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 del container. Stato non recuperabile. 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>. Per esempio:
  - Estensione. NoSuchGestore
  - Extension.api trovata KeyNot
  - Estensione. ConfigInvalid

- Estensione. 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

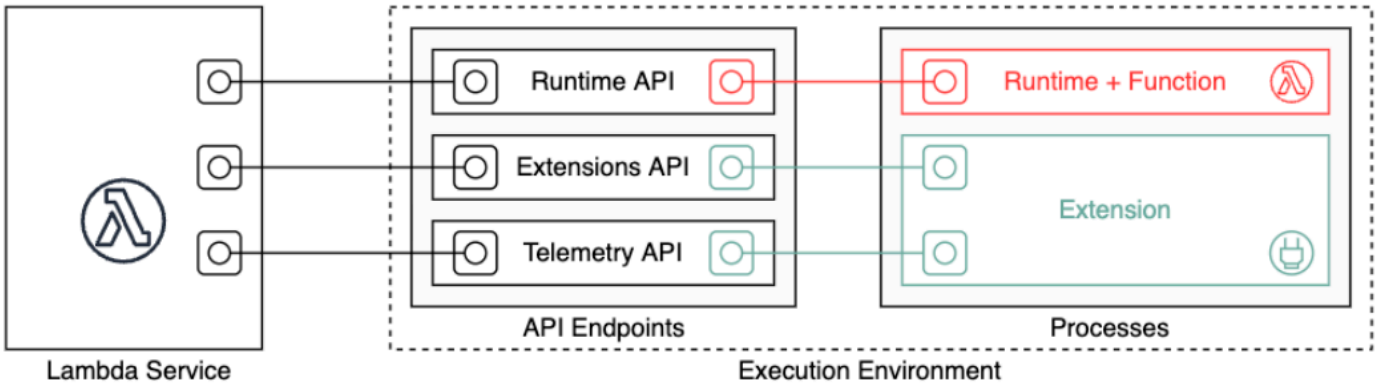
- 202 – Accettato
- 400 – Richiesta non valida
- 403 – Non consentito
- 500 – Errore del container. Stato non recuperabile. L'estensione dovrebbe uscire tempestivamente.

API di telemetria Lambda

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. Dopodiché avrai la flessibilità di elaborare, filtrare e trasmettere 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.

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.

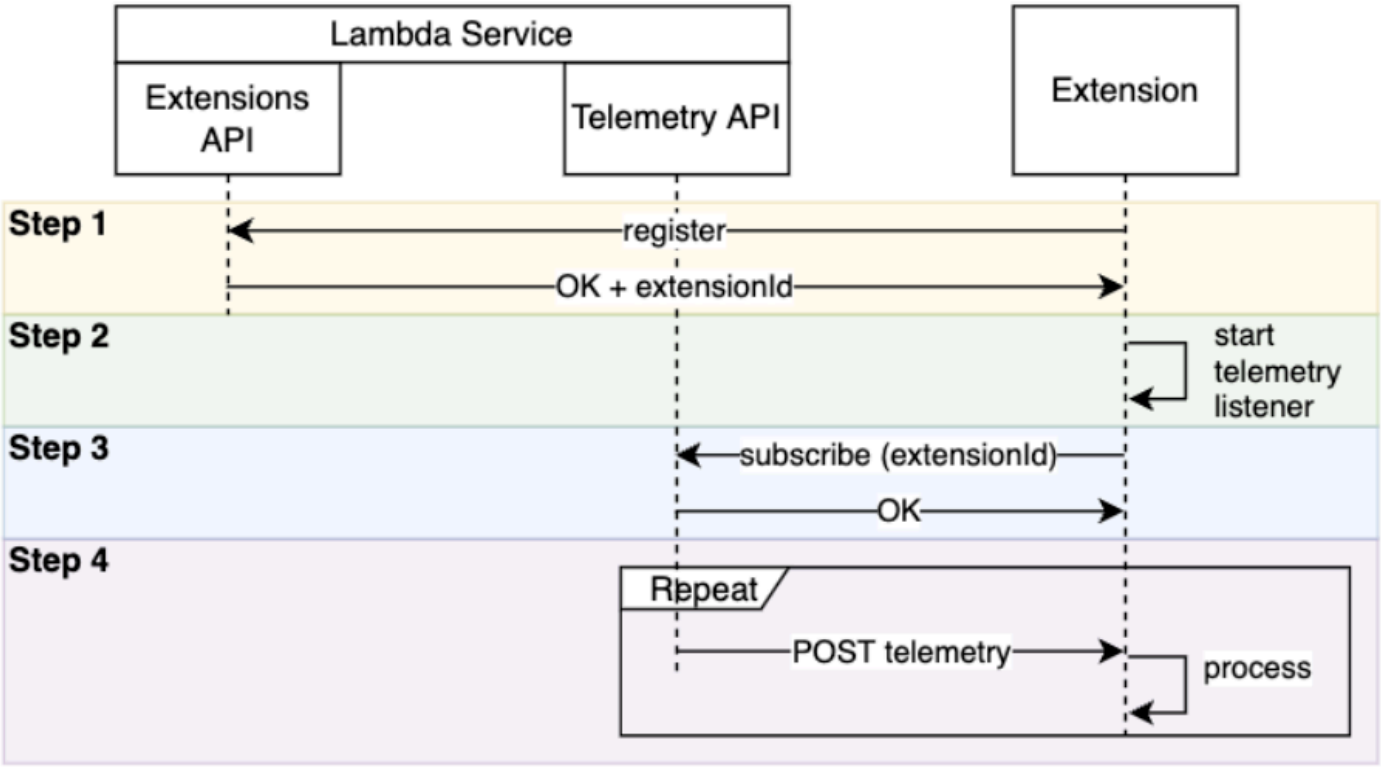
> **Note**
>
> Lambda invia log e metriche e tracce a X-Ray (se hai attivato il tracciamento) CloudWatch, anche se un'estensione si abbona ai flussi di telemetria.

Sections

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 ascoltatore 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 ascoltatore 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 ascoltatore 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 l'ascoltatore di telemetria. 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'ascoltatore. 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 ascoltatore di telemetria

L'estensione Lambda deve disporre di un ascoltatore che gestisca le richieste in entrata dall'API di telemetria. Il codice seguente mostra un esempio di implementazione dell'ascoltatore 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 i ricevere i dati di telemetria, definisci l'ascoltatore 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 ascoltatore o se la richiesta POST subisce un errore. Se riporta un arresto anomalo, l'ascoltatore 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
    }
}
```

Impostazioni di configurazione del buffering

Parametro	Descrizione	Valori predefiniti e limiti
`maxBytes`	Il volume massimo di dati di telemetria (in byte) da memorizzare nel buffer della memoria.	Predefinito: 262.144. Minimo: 262.144. Massimo: 1.048.576.

Parametro	Descrizione	Valori predefiniti e limiti
`maxItems`	Il numero massimo di eventi da memorizzare nel buffer.	Predefinito: 10.000 Minimo: 1.000 Massimo: 10.000.
`timeoutMs`	Il tempo massimo (in millisecondi) per memorizzare nel buffer un batch.	Predefinito: 1.000 Minimo: 25 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 * maxBytes + 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": "2022-12-13",
   "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 l'abbonamento con l'API Telemetry, un'estensione comincia in automatico a ricevere i dati di telemetria da Lambda attraverso 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`.

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.

Tipi di messaggi dell'API di telemetria

Categoria	Tipo di evento	Descrizione	Schema dei registri di eventi
Evento della piattaforma	`platform.initStart`	Inizializzazione della funzione avviata.	Schema the section called “`platform.initStart`”
Evento della piattaforma	`platform.initRuntimeDone`	Inizializzazione della funzione completata.	Schema the section called “`platform.initRuntimeDone`”

Categoria	Tipo di evento	Descrizione	Schema dei registri di eventi
Evento della piattaforma	`platform.initReport`	Un rapporto sull'inizializzazione della funzione.	Schema the section called “`platform.initReport`”
Evento della piattaforma	`platform.start`	L'invocazione della funzione è stata avviata.	Schema the section called “`platform.start`”
Evento della piattaforma	`platform.runtimeDone`	Il runtime ha terminato l'elaborazione di un evento con esito positivo o negativo.	Schema the section called “`platform.runtimeDone`”
Evento della piattaforma	`platform.report`	Un rapporto sull'invocazione di una funzione.	Schema the section called “`platform.report`”
Evento della piattaforma	`platform.restoreStart`	Il ripristino del runtime è iniziato.	Schema the section called “`platform.restoreStart`”
Evento della piattaforma	`platform.restoreRuntimeDone`	Il ripristino del runtime è stato completato.	Schema the section called “`platform.restoreRuntimeDone`”
Evento della piattaforma	`platform.restoreReport`	Rapporto di ripristino del runtime.	Schema the section called “`platform.restoreReport`”
Evento della piattaforma	`platform.telemetrySubscription`	L'estensione che ha eseguito la sottoscrizione all'API di telemetria.	Schema the section called “`platform.telemetrySubscription`”

Categoria	Tipo di evento	Descrizione	Schema dei registri di eventi
Evento della piattaforma	`platform.logsDropped`	Lambda ha eliminato le voci del log.	Schema the section called “`platform.logsDropped`”
Log delle funzioni	`function`	Una riga del log dal codice della funzione.	Schema the section called “`function`”
Log di estensioni	`extension`	Una riga di log dal codice dell'estensione.	Schema the section called “`extension`”

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/`. Per esempio:

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Per la definizione della specifica OpenAPI (OAS) della versione delle risposte alla sottoscrizione `2022-12-13`, 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
    - Campo obbligatorio: sì
    - Tipo: String
    - Valori validi: `"2022-12-13"` o `"2022-07-01"`
  - destinazione: le impostazioni di configurazione che definiscono la destinazione dell'evento di telemetria e il protocollo per la consegna degli eventi.
    - Campo obbligatorio: sì

- Tipo: oggetto

```
{
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080"
}
```

  - protocollo: il protocollo utilizzato da Lambda per inviare dati di telemetria.
    - Campo obbligatorio: sì
    - Tipo: String
    - Valori validi: `"HTTP"|"TCP"`
  - URI: l'URI a cui inviare i dati di telemetria.
    - Campo 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.
  - Campo obbligatorio: sì
  - Tipo: matrice di stringhe
  - Valori validi: `"platform"|"function"|"extension"`
- buffering: le impostazioni di configurazione per il buffering degli eventi.
  - Campo obbligatorio: no
  - Tipo: oggetto

```
{
   "buffering": {
        "maxItems": 1000,
        "maxBytes": 256*1024,
        "timeoutMs": 100
   }
}
```

  - maxItems – Il numero massimo di eventi da memorizzare nel buffer.
    - Campo obbligatorio: no
    - Tipo: integer

    - 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.
    - Campo obbligatorio: no
    - Tipo: integer
    - Predefinito: 262.144.
    - Minimo: 262.144.
    - Massimo: 1.048.576.
- timeoutMs – Il tempo massimo (in millisecondi) per il buffer di un batch.
    - Campo obbligatorio: no
    - Tipo: integer
    - 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": "2022-12-13",
   "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. Per 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 **Event** 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/`. Per esempio:

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Per la definizione della specifica OpenAPI (OAS) della versione delle risposte alla sottoscrizione `2022-12-13`, consulta quanto segue:

- HTTP — telemetry-api-http-schema.zip
- TCP — .zip telemetry-api-tcp-schema

La tabella seguente è un riepilogo di tutti i tipi di oggetti `Event` supportati dall'API di telemetria.

Tipi di messaggi dell'API di telemetria

Categoria	Tipo di evento	Descrizione	Schema dei registri di eventi
Evento della piattaforma	`platform.initStart`	Inizializzazione della funzione avviata.	Schema the section called “`platform.initStart`”
Evento della piattaforma	`platform.initRuntimeDone`	Inizializzazione della funzione completata.	Schema the section called “`platform.initRuntimeDone`”
Evento della piattaforma	`platform.initReport`	Un rapporto sull'inizializzazione della funzione.	Schema the section called “`platform.initReport`”
Evento della piattaforma	`platform.start`	L'invocazione della funzione è stata avviata.	Schema the section called “`platform.start`”

Categoria	Tipo di evento	Descrizione	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 the section called “`platform.runtimeDone`”
Evento della piattaforma	`platform.report`	Un rapporto sull'invocazione di una funzione.	Schema the section called “`platform.report`”
Evento della piattaforma	`platform.restoreStart`	Il ripristino del runtime è iniziato.	Schema the section called “`platform.restoreStart`”
Evento della piattaforma	`platform.restoreRuntimeDone`	Il ripristino del runtime è stato completato.	Schema the section called “`platform.restoreRuntimeDone`”
Evento della piattaforma	`platform.restoreReport`	Rapporto di ripristino del runtime.	Schema the section called “`platform.restoreReport`”
Evento della piattaforma	`platform.telemetrySubscription`	L'estensione che ha eseguito la sottoscrizione all'API di telemetria.	Schema the section called “`platform.telemetrySubscription`”
Evento della piattaforma	`platform.logsDropped`	Lambda ha eliminato le voci del log.	Schema the section called “`platform.logsDropped`”
Log delle funzioni	`function`	Una riga del log dal codice della funzione.	Schema the section called “`function`”

Categoria	Tipo di evento	Descrizione	Schema dei registri di eventi
Log di estensioni	`extension`	Una riga di log dal codice dell'estensione.	Schema the section called “`extension`”

Indice

- Status
- 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`
- instanceMaxMemory? – `Integer`
- phase: oggetto `the section called “InitPhase”`
- runtimeVersion?: `String`
- runtimeVersionArn? – `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
    }
}
```

`platform.initRuntimeDone`

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 “Status”
- spans? : elenco di oggetti the section called “Span”

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
            }
        ]
    }
}
```

platform.initReport

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 “Span”
- status: oggetto the section called “Status”

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:

```
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 “Status”
- spans? : elenco di oggetti the section called “Span”
- 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 inizializzazione della funzione. Un oggetto `platform.report` `Event` ha la seguente forma:

```
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 “Span”
- status: oggetto the section called “Status”
- 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 inizializzarlo da zero. Per ulteriori informazioni, consulta Lambda 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`
- instanceMaxMemory? – `String`
- runtimeVersion?: `String`
- runtimeVersionArn? – `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
    }
}
```

platform.restoreRuntimeDone

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 inizializzarlo da zero. Per ulteriori informazioni, consulta Lambda 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 “Span”
- status: oggetto the section called “Status”

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
            }
        ]
    }
}
```

platform.restoreReport

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 “Span”
- status: oggetto the section called “Status”

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.extension`

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"
    }
}
```

`platform.telemetrySubscription`

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.logsDropped`

Un evento `platform.logsDropped` contiene informazioni sugli eventi eliminati. Lambda emette l'`platform.logsDroppedevento` quando una funzione emette i log a una velocità troppo alta perché Lambda possa elaborarli. Quando Lambda non è in grado di inviare log a CloudWatch o verso l'estensione sottoscritta all'API di telemetria alla velocità di produzione della funzione, 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"
    }
```

```
}
```

`function`

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 “Configurazione dei formati di log JSON e testo normale”

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.

`extension`

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 “Configurazione dei formati di log JSON e testo normale”

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.

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
}
```

`Span`

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 span, consulta Span nella pagina Tracing API del sito Web di 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
      }
```

`Status`

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 dell'API di **`Event`** telemetria Lambda in Spans 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 (Otel) Spans. `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, consulta Span nella pagina Tracing API del sito Web Docs. OpenTelemetry

Sections

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
`Span.Name`	La tua estensione genera questo valore in base al campo `type`.
`Span.StartTime`	Utilizza `e.time`.
`Span.EndTime`	N/D, perché l'evento non è ancora stato completato.
`Span.Kind`	Imposta su `Server`.
`Span.Status`	Imposta su `Unset`.

OpenTelemetry	Schema dell'API di telemetria Lambda
`Span.TraceId`	Analizza l'intestazione AWS X-Ray trovata in `e.tracing.value` , quindi usa il valore `TraceId`.
`Span.ParentId`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Parent`.
`Span.SpanId`	Usa `e.tracing.spanId` se disponibile. Altrimenti, genera un nuovo `SpanId`.
`Span.SpanContext.TraceState`	N/D per un contesto di traccia X-Ray.
`Span.SpanContext.TraceFlags`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Sampled`.
`Span.Attributes`	La tua estensione può aggiungere qualsiasi valore personalizzato qui.

Mappatura degli eventi **`*RuntimeDone`**

OpenTelemetry	Schema dell'API di telemetria Lambda
`Span.Name`	L'estensione genera il valore in base al campo `type`.
`Span.StartTime`	Usa `e.time` dall'evento `*Start` corrispon dente. In alternativa, utilizzare `e.time - e.metrics.durationMs` .
`Span.EndTime`	N/D, perché l'evento non è ancora stato completato.
`Span.Kind`	Imposta su `Server`.

OpenTelemetry	Schema dell'API di telemetria Lambda
`Span.Status`	Se `e.status` non è uguale a `success`, imposta su `Error`. In caso contrario, imposta il valore su `0k`.
`Span.Events[]`	Utilizza `e.spans[]` .
`Span.Events[i].Name`	Utilizza `e.spans[i].name` .
`Span.Events[i].Time`	Utilizza `e.spans[i].start` .
`Span.TraceId`	Analizza l'intestazione AWS X-Ray trovata in `e.tracing.value` , quindi usa il valore `TraceId`.
`Span.ParentId`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Parent`.
`Span.SpanId`	Usa lo stesso `SpanId` dell'evento `*Start`. Se non disponibile, usa `e.tracing.spanId` o genera un nuovo `SpanId`.
`Span.SpanContext.TraceState`	N/D per un contesto di traccia X-Ray.
`Span.SpanContext.TraceFlags`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Sampled`.
`Span.Attributes`	La tua estensione può aggiungere qualsiasi valore personalizzato qui.

Mappatura degli eventi **`*Report`**

OpenTelemetry	Schema dell'API di telemetria Lambda
`Span.Name`	L'estensione genera il valore in base al campo `type`.
`Span.StartTime`	Usa `e.time` dall'evento `*Start` corrispondente. In alternativa, utilizzare `e.time - e.metrics.durationMs` .
`Span.EndTime`	Utilizza `e.time`.
`Span.Kind`	Imposta su `Server`.
`Span.Status`	Usa lo stesso valore dell'evento `*RuntimeDone` .
`Span.TraceId`	Analizza l'intestazione AWS X-Ray trovata in `e.tracing.value` , quindi usa il valore `TraceId`.
`Span.ParentId`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Parent`.
`Span.SpanId`	Usa lo stesso `SpanId` dell'evento `*Start`. Se non disponibile, usa `e.tracing.spanId` o genera un nuovo `SpanId`.
`Span.SpanContext.TraceState`	N/D per un contesto di traccia X-Ray.
`Span.SpanContext.TraceFlags`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Sampled`.
`Span.Attributes`	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
`Span.Name`	L'estensione genera il valore in base al campo `type`.
`Span.StartTime`	Usa `e.time` dall'evento `*Start` corrispondente. In alternativa, utilizzare `e.time - e.metrics.durationMs` .
`Span.EndTime`	N/D, perché l'evento non è ancora stato completato.
`Span.Kind`	Imposta su `Server`.
`Span.Status`	Se `e.status` non è uguale a `success`, imposta su `Error`. In caso contrario, imposta il valore su `Ok`.
`Span.TraceId`	Analizza l'intestazione AWS X-Ray trovata in `e.tracing.value` , quindi usa il valore `TraceId`.
`Span.ParentId`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Parent`.

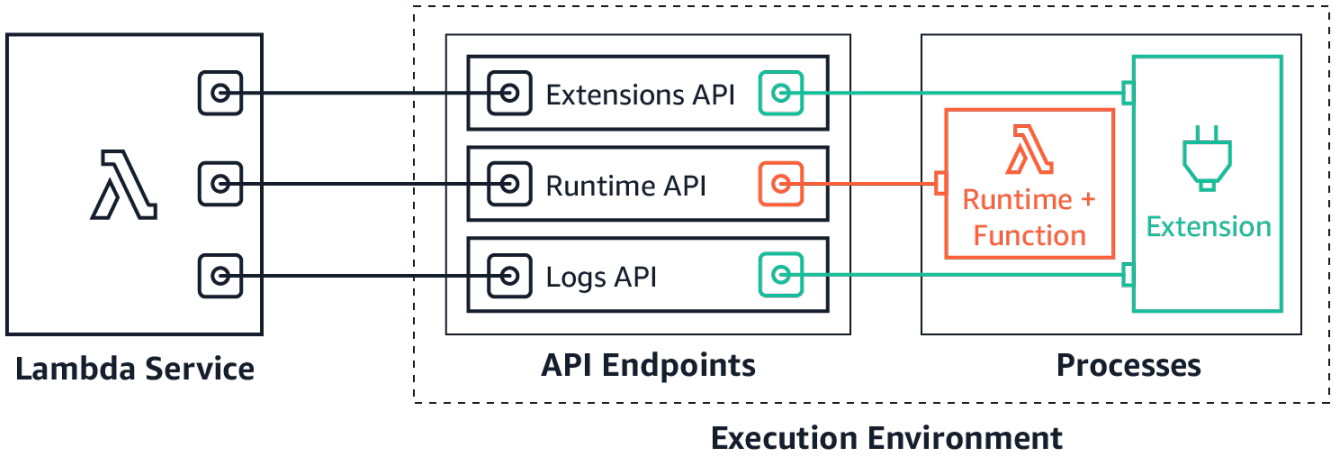
OpenTelemetry	Schema dell'API di telemetria Lambda
`Span.SpanId`	Usa lo stesso `SpanId` dell'evento `*Start`. Se non disponibile, usa `e.tracing.spanId` o genera un nuovo `SpanId`.
`Span.SpanContext.TraceState`	N/D per un contesto di traccia X-Ray.
`Span.SpanContext.TraceFlags`	Analizza l'intestazione X-Ray trovata in `e.tracing.value`, quindi usa il valore `Sampled`.
`Span.Attributes`	La tua estensione può aggiungere qualsiasi valore personalizzato qui.
`ChildSpan[i].Name`	Utilizza `e.spans[i].name`.
`ChildSpan[i].StartTime`	Utilizza `e.spans[i].start`.
`ChildSpan[i].EndTime`	Utilizza `e.spans[i].start + e.spans[i].durations`.
`ChildSpan[i].Kind`	Uguale all'elemento padre `Span.Kind`.
`ChildSpan[i].Status`	Uguale all'elemento padre `Span.Status`.
`ChildSpan[i].TraceId`	Uguale all'elemento padre `Span.TraceId`.
`ChildSpan[i].ParentId`	Usa elemento padre `Span.SpanId`.
`ChildSpan[i].SpanId`	Genera un nuovo `SpanId`.
`ChildSpan[i].SpanContext.TraceState`	N/D per un contesto di traccia X-Ray.
`ChildSpan[i].SpanContext.TraceFlags`	Uguale all'elemento padre `Span.SpanContext.TraceFlags`.

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.

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 si piattaforma Lambda, che registrano eventi ed errori relativi a richiami ed estensioni.

> **Note**
>
> Lambda invia tutti i log a CloudWatch, anche quando un'estensione sottoscrive uno o più flussi di log.

Argomenti

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-Identifier`). 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*maxBytes+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`.

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 illustra come inviare i log a una destinazione personalizzata, consulta Utilizzo delle estensioni AWS Lambda per inviare i log a una destinazione personalizzata nel blog Compute di AWS.

Per esempi di codice in Python e Go che mostrano come sviluppare un'estensione Lambda di base e sottoscrivere l'API Logs, consulta AWS LambdaExtensions on the Samples repository. AWS 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 dallaAWS_LAMBDA_RUNTIME_API variabile di ambiente. Per inviare una richiesta API, utilizzare il prefisso `2020-08-15/` prima del percorso API. Per esempio:

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs
```

La specifica OpenAPI per la versione Logs API 2020-08-15 è disponibile qui: .zip logs-api-request

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:

- logs-api-http-responseHTTP: .zip
- TCP: .zip logs-api-tcp-response

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

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 log di funzioni. { "time": "2020-08-20T12:31:32.123Z", "type": "function", "record": "ERROR encountered. 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"}
}
```

La piattaforma. initRuntimeDoneil messaggio di registro 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 una piattaforma di successo. initRuntimeDonemessaggio di registro per il tipo di `snap-start` inizializzazione.

```
{
  "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 traccia AWS X-Ray è attiva, il log 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"
}
```

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 un`platform.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 Piattaforma di esempio. restoreRuntimeDone messaggio (solo tipo di **`snap-start`** inizializzazione)

La piattaforma. restoreRuntimeDoneil messaggio di registro mostra se la `Restore` fase ha avuto successo o meno. 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 una piattaforma di successo. restoreRuntimeDonemessaggio di registro.

```
{
  "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

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 distribuisci gli aggiornamenti alla tua funzione direttamente con l'API Lambda o con un client come AWS CLI il, 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 per 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: autorizzazione negata/Impossibile caricare tale file

Errore: EACCES: permission denied, open '/var/task/index.js' (EACCES: autorizzazione negata, aperto “/var/task/index.js”)

Errore: cannot load such file -- function (impossibile caricare tale file - funzione)

Errore: [Errno 13] Permission denied: '/var/task/function.py' ([Errno 13] Autorizzazione negata: “/var/task/function.py”)

Il runtime Lambda necessita dell'autorizzazione per leggere i file nel pacchetto di distribuzione. Nella notazione ottale delle autorizzazioni Linux, Lambda richiede 644 permessi per i file non eseguibili (rw-r--r--) e 755 permessi () per le directory e i file eseguibili. rwxr-xr-x

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 eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 755 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta Set, View, Change, or Remove Permissions on an Object nella documentazione di Microsoft Windows.

Generale: si verifica un errore quando si chiama UpdateFunctionCode

Errore: si è verificato un errore (RequestEntityTooLargeException) durante la chiamata dell' UpdateFunctionCodeoperazione

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, archivialo in Amazon S3 e utilizza i parametri S3Bucket e S3Key.

> **Note**
>
> Quando caricate un file direttamente con AWS SDK o in altro modo 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 specifichi un oggetto Amazon S3 in una chiamata o utilizzi il pacchetto e distribuisci i comandi nella o AWS CLI nella CLI. UpdateFunctionCode 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" dal gruppo "Function".

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 #<:0x000055b76ccebf98> LambdaHandler

Errore: Nessun metodo pubblico denominato HandleRequest con firma del metodo appropriata trovato sulla classe function.Handler

Errore: impossibile trovare il metodo "HandleRequest" nel tipo "Function.Handler" dal gruppo "Function"

Il nome del metodo dell'handler nella configurazione dell'handler della funzione non corrisponde al codice. Ogni runtime definisce una convenzione di denominazione per gli handler, ad esempio il *`nome del file`*.*`nome del metodo`*. 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.

- Compilazione 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

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: o 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":" USFey5 7ATNx5bsm... cyPiPn

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": "nodejs20.x",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Environment": {
        "Variables": {
            "BUCKET": "DOC-EXAMPLE-BUCKET",
            "KEY": "file.txt"
```

```
        }
    },
    ...
}
```

In questo esempio, l'oggetto è di 39 caratteri e quando archiviato occupa 39 byte (senza spazi) come stringa `{"BUCKET":"DOC-EXAMPLE-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 che hai fornito 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: la funzione specificata ConcurrentExecutions per la funzione riduce il numero di account al di UnreservedConcurrentExecution sotto del valore minimo

Errore: il valore MemorySize '' non è riuscito a soddisfare 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 quote di concorrenza e memoria ridotte. Per risolvere gli errori relativi alla simultaneità, puoi richiedere un aumento della quota. Non è possibile richiedere un aumento della quota.

- Concorrenza: potresti ricevere un errore se provi a creare una funzione utilizzando la concorrenza riservata o fornita o se la richiesta di concorrenza per funzione () PutFunctionConcurrencysupera la quota di concorrenza del tuo account.
- Memoria: se la quantità di memoria allocata per la funzione supera la quota di memoria dell'account si verificano degli errori.

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).

IAM: lambda: InvokeFunction non autorizzato

Errore: User: arn:aws:iam: :123456789012:user/developer non 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 politica AWS gestita al tuo utente o aggiungi una politica personalizzata AWSLambdaRoleche 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 un bootstrap (Runtime) valido. InvalidEntrypoint)

Errore: impossibile trovare bootstrap validi: [/var/task/bootstrap/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

ErroreResourceConflictException: 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 a zero la concorrenza disponibile, il che limita tutte le future chiamate. 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 con provisioning

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 fornita, puoi controllare la metrica di ProvisionedConcurrencyInvocations Amazon CloudWatch . Un valore diverso da zero indica che la funzione sta elaborando le chiamate in ambienti di esecuzione inizializzati.
- Verifica se la concorrenza delle funzioni supera il livello configurato di concorrenza fornita 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.

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-point/fsap-015cxmplb72b405fd.

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 punto di accesso arn:aws:elasticfilesystem:us-east- 2:123456789012:access-point/fsap-015cxmplb72b405fd. 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 il punto di accesso {arn:aws:elasticfilesystem:us-east- 2:123456789012:access-point/fsap-015cxmplb72b405fd} a causa del timeout di montaggio.

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. Per ulteriori informazioni, consulta Prestazioni.

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.

Lambda: l'esecuzione richiede troppo tempo

Problema: l'esecuzione della funzione richiede troppo tempo.

Se l'esecuzione del codice richiede molto più tempo 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: i log o le tracce non vengono visualizzati

Problema: i registri non vengono visualizzati nei CloudWatch registri.

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 autorizzazioni alla tua funzione, esegui anche un semplice aggiornamento del codice o della 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 una chiamata di funzione.

Lambda: non vengono visualizzati tutti i log della mia funzione

Problema: i registri delle funzioni non sono presenti in CloudWatch Logs, anche se le mie autorizzazioni sono corrette

Se Account AWS raggiungi i limiti di quota dei CloudWatch log, CloudWatch rallenta la registrazione delle funzioni. Quando ciò accade, alcuni dei log generati dalle tue funzioni potrebbero non apparire in Logs. CloudWatch

Se la funzione emette i log a una velocità troppo elevata per consentire a Lambda di elaborarli, ciò può anche far sì che gli output di log non vengano visualizzati in Logs. CloudWatch Quando Lambda non riesce a inviare i log CloudWatch alla velocità con cui vengono prodotti dalla funzione, li elimina per evitare che l'esecuzione della funzione rallenti.

Se la 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 di 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 un 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 registri 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 restituisce prima che il codice finisca 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 della funzione Lambda in Node.js.

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 di scripting includono l' AWS SDK e vengono periodicamente aggiornati alla versione più recente. La versione corrente per ogni runtime è elencata nella pagina runtime.

Per utilizzare una versione più recente dell' AWS SDK o per bloccare le funzioni su una versione specifica, puoi raggruppare la libreria con il codice della funzione o creare un layer Lambda. Per informazioni dettagliate sulla creazione di un pacchetto di distribuzione con dipendenze, vedere i seguenti argomenti:

Node.js

- Distribuisci funzioni Lambda in Node.js con archivi di file .zip

Python

- Utilizzo di archivi di file .zip per le funzioni Lambda in Python

Ruby

- Utilizzo di archivi di file .zip per le funzioni Lambda in Ruby

Java

- Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR

Go

- Distribuisci 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

- Implementa le funzioni PowerShell Lambda con 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.

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 Virtual Private Cloud (Amazon VPC).

Gli errori di connettività di rete possono derivare da problemi relativi alla configurazione di routing del VPC, alle regole dei gruppi di sicurezza, alle autorizzazioni dei ruoli AWS Identity and Access Management (IAM) o alla traduzione degli indirizzi di rete (NAT) del 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.

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 a Internet per le funzioni VPC”.

Se alcune delle connessioni TCP sono scadute, ciò potrebbe essere 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 funzione richiede l'accesso ai AWS servizi senza utilizzare Internet

Problema: la funzione Lambda richiede l'accesso ai AWS servizi senza utilizzare Internet.

Per connettere una funzione ai AWS servizi 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 elastica di rete 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

Riceverai questo errore se cerchi 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.

AWS Lambda applicazioni

Un' AWS Lambda applicazione è una combinazione di funzioni Lambda, sorgenti di eventi e altre risorse che collaborano per eseguire attività. È possibile utilizzare AWS CloudFormation altri strumenti per raccogliere i componenti dell'applicazione in un unico pacchetto che può essere distribuito e gestito come un'unica risorsa. Le applicazioni rendono portabili i tuoi progetti Lambda e ti consentono di integrarli con strumenti di sviluppo aggiuntivi, come AWS CodePipeline AWS CodeBuild, e l'interfaccia a riga di AWS Serverless Application Model comando (CLI AWS SAM).

AWS Serverless Application Repository fornisce una raccolta di applicazioni Lambda che è possibile distribuire nell'account con pochi clic. Il repository include sia ready-to-use applicazioni che esempi che puoi usare come punto di partenza per i tuoi progetti. Puoi anche inviare i tuoi progetti per l'inclusione.

AWS CloudFormation consente di creare un modello che definisce le risorse dell'applicazione e gestire l'applicazione come stack. Puoi aggiungere o modificare le risorse con maggior sicurezza nel tuo stack di applicazioni. Se una qualsiasi parte di un aggiornamento fallisce, torna AWS CloudFormation automaticamente alla configurazione precedente. Con AWS CloudFormation i parametri, è possibile creare più ambienti per l'applicazione a partire dallo stesso modello. AWS SAMsi estende AWS CloudFormation con una sintassi semplificata incentrata sullo sviluppo di applicazioni Lambda.

La AWS CLI e la CLI di AWS SAM SAM sono gli strumenti a riga di comando per la gestione degli stack di applicazioni Lambda. Oltre ai comandi per la gestione degli stack di applicazioni con l' AWS CloudFormation API, AWS CLI supporta comandi di livello superiore che semplificano attività come il caricamento di pacchetti di distribuzione e l'aggiornamento dei modelli. La AWS SAM CLI fornisce funzionalità aggiuntive, tra cui la convalida dei modelli, il test locale e l'integrazione con i sistemi CI/CD.

Quando si crea un'applicazione, è possibile creare il relativo repository Git utilizzando uno dei due CodeCommit o una AWS CodeStar connessione a GitHub. CodeCommit ti consente di utilizzare la console IAM per gestire le chiavi SSH e le credenziali HTTP per i tuoi utenti. CodeConnections ti consente di connetterti al tuo GitHub account. Per ulteriori informazioni sulle connessioni, consulta Cosa sono le connessioni? nella Guida per l'utente della console Strumenti per sviluppatori.

Per ulteriori informazioni sulla progettazione delle applicazioni Lambda, consulta Progettazione delle applicazioni in Serverless Land.

Argomenti

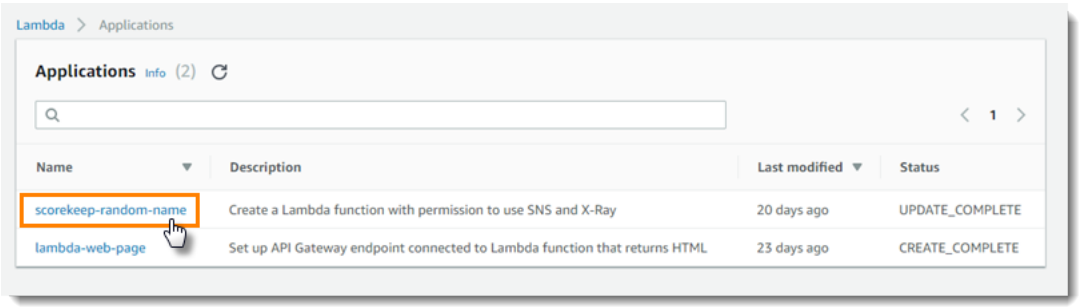
- Gestione delle applicazioni nella console di AWS Lambda
- Crea distribuzioni continue per le funzioni Lambda
- Utilizzo di Lambda con Kubernetes

Gestione delle applicazioni nella console di AWS Lambda

La console AWS Lambda ti consente di monitorare e gestire le tue applicazioni Lambda. Il menu Applications (Applicazioni) elenca gli stack AWS CloudFormation con le funzioni Lambda. Il menu include gli stack che avvii in AWS CloudFormation utilizzando la console AWS CloudFormation, la AWS Serverless Application Repository, la AWS CLI, o la CLI AWS SAM.

Per visualizzare un'applicazione Lambda

1. Aprire la pagina Applicazioni della console Lambda.
2. Scegliere un'applicazione.



La panoramica mostra le seguenti informazioni sulla tua applicazione.

- Template AWS CloudFormation (Modello AWS CloudFormation) o SAM template (Modello SAM) - Il modello che definisce la tua applicazione.
- Resources (Risorse) - Le risorse AWS definite nel modello dell'applicazione. Per gestire le funzioni Lambda dell'applicazione, scegli un nome di funzione dall'elenco.

Monitoraggio delle applicazioni

La scheda Monitoraggio mostra 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).

Per impostazione predefinita, la console Lambda mostra un pannello di controllo di base. Puoi personalizzare questa pagina definendo pannelli di controllo personalizzati nel modello della tua applicazione. 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.

Monitoraggio personalizzato dei pannelli di controllo

Personalizza la pagina di monitoraggio dell'applicazione aggiungendo uno o più CloudWatch dashboard Amazon al modello di applicazione con il tipo di AWS::CloudWatch::Dashboardrisorsa. 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

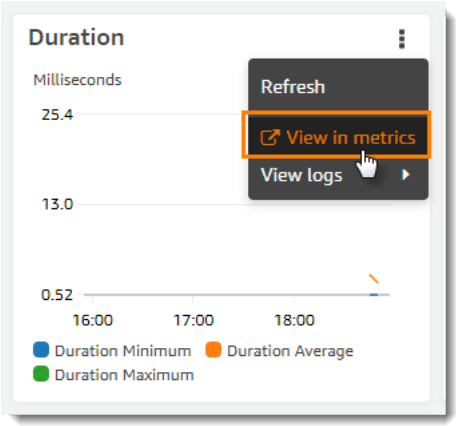
```
Resources:
  MyDashboard:
    Type: AWS::CloudWatch::Dashboard
    Properties:
      DashboardName: my-dashboard
      DashboardBody: |
        {
            "widgets": [
                {
                    "type": "metric",
                    "width": 12,
                    "height": 6,
                    "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
            }
        }
    ]
}
```

Puoi ottenere la definizione di un widget del pannello di controllo di monitoraggio predefinito dalla console CloudWatch.

Per visualizzare la definizione di un widget

1. Aprire la pagina Applicazioni della console Lambda.
2. Scegliere un'applicazione che ha il pannello di controllo standard.
3. Selezionare Monitoring (Monitoraggio).
4. In qualsiasi widget, scegliere View in metrics (Visualizza nei parametri) dal menu a discesa.



5. Scegliere Source (Origine).

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

Crea distribuzioni continue per le funzioni Lambda

Utilizzare le distribuzioni in sequenza per controllare i rischi associati all'introduzione di nuove versioni della funzione Lambda. In una distribuzione in sequenza, il sistema distribuisce automaticamente la nuova versione della funzione e invia gradualmente una quantità crescente di traffico alla nuova versione. La quantità di traffico e la velocità dell'aumento sono parametri che è possibile configurare.

È possibile configurare una distribuzione continua utilizzando e. AWS CodeDeploy AWS SAM CodeDeploy è un servizio che automatizza le distribuzioni di applicazioni su piattaforme di elaborazione Amazon come Amazon EC2 e. AWS LambdaPer ulteriori informazioni, consulta Cos'è? CodeDeploy . Utilizzando CodeDeploy to deploy la funzione Lambda, è possibile monitorare facilmente lo stato della distribuzione e avviare un rollback in caso di problemi.

AWS SAM è un framework open source per la creazione di applicazioni serverless. Si crea un AWS SAM modello (in formato YAML) per specificare la configurazione dei componenti necessari per la distribuzione continua. AWS SAM utilizza il modello per creare e configurare i componenti. Per ulteriori informazioni, consulta la pagina Che cos'è AWS SAM?.

In una distribuzione continua, AWS SAM esegue le seguenti attività:

- Configura la funzione Lambda e crea un alias.

  La configurazione del routing degli alias è la funzionalità sottostante che implementa la distribuzione in sequenza.
- Crea un' CodeDeploy applicazione e un gruppo di distribuzione.

  Il gruppo di distribuzione gestisce la distribuzione in sequenza e il rollback (se necessario).
- Rileva quando viene creata una nuova versione della funzione Lambda.
- Si attiva CodeDeploy per avviare la distribuzione della nuova versione.

Esempio di AWS SAM modello Lambda

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: nodejs18.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' CodeDeploy applicazione sposta il traffico dalla versione originale della funzione Lambda alla nuova versione. Il valore `Linear10PercentEvery2Minutes` sposta un ulteriore dieci percento del traffico alla nuova versione ogni due minuti.

Per l'elenco delle configurazioni di distribuzione predefinite, consulta Configurazioni di distribuzione.

Per un tutorial dettagliato sull'utilizzo CodeDeploy con le funzioni Lambda, consulta Implementare una funzione Lambda aggiornata con. CodeDeploy

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 AWS servizi come Lambda, Amazon Elastic Container Registry (Amazon ECR), Amazon Simple Storage Service (Amazon S3) e Amazon. SageMaker 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 ha un DevOps processo per distribuire le risorse in un cluster Amazon Elastic Kubernetes Service (Amazon EKS) utilizzando i 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 kubectl, 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 GitOps tuo progetto utilizza Kubernetes. In questo modello, Kubernetes riconcilia continuamente il repository git (stato desiderato) con le risorse in esecuzione all'interno del cluster (stato corrente). Se ci sono differenze, il GitOps processo apporta automaticamente modifiche al cluster. Puoi utilizzare GitOps Kubernetes per distribuire e gestire le funzioni Lambda tramite Crossplane, utilizzando strumenti e concetti Kubernetes familiari come 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.

- Deploying Lambda with EKS and 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.

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 facilitare la distribuzione e la pulizia, un AWS SAM modello e risorse di supporto.

Node.js

Applicazioni Lambda di esempio in Node.js

- blank-nodejs — Una funzione Node.js 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.
- efs-nodejs - Una funzione che utilizza un file system Amazon EFS in un Amazon VPC. Questo esempio include un VPC, un file system, destinazioni di montaggio e un punto di accesso configurati per l'utilizzo con Lambda.

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

- java17-examples: una funzione Java che dimostra come utilizzare un record Java per rappresentare un oggetto di dati dell'evento di input.
- 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' AWS SDK 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.
- Utilizza API Gateway per richiamare una funzione Lambda: una funzione Java che esegue la scansione di una tabella Amazon DynamoDB che contiene informazioni sui dipendenti. Quindi utilizza Amazon Simple Notification Service per inviare un messaggio di testo ai dipendenti per festeggiare i loro anniversari di lavoro. Questo esempio usa API Gateway per richiamare la funzione.

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 di più nelle guide Micronaut/Lambda.
- Demo dell'applicazione Quarkus senza server: un esempio che mostra come utilizzare Quarkus in un runtime Java gestito con e senza, o come immagine nativa GraalVM con un runtime personalizzato. SnapStart Scopri di più nella guida Quarkus/Lambda 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. Per ulteriori informazioni sull'architettura e sui casi d'uso di un'applicazione, leggere gli argomenti in questo capitolo.

Argomenti

- Applicazione di esempio di funzione vuota per AWS Lambda

Applicazione di esempio di funzione vuota per AWS Lambda

L'applicazione di esempio di una funzione vuota dimostra le operazioni comuni di Lambda con una funzione che invoca l'API Lambda. Mostra l'uso della registrazione, delle variabili di ambiente, del AWS X-Ray tracciamento, dei livelli, dei test unitari e dell' AWS SDK. Esplora questa applicazione per imparare a creare funzioni Lambda nel tuo linguaggio di programmazione o usala come punto di partenza per i tuoi progetti.

Le varianti di questa applicazione di esempio sono disponibili per i seguenti linguaggi:

Varianti

- Node.js – blank-nodejs.
- Python – blank-python.
- Ruby – blank-ruby.
- Java – blank-java.
- Go – blank-go.
- C# – blank-csharp.
- PowerShell — powershell bianco.

Gli esempi in questo argomento evidenziano il codice della versione Node.js, ma i dettagli sono generalmente applicabili a tutte le varianti.

È possibile distribuire il campione in pochi minuti con and. AWS CLI AWS CloudFormation Segui le istruzioni contenute nel file README per scaricarlo, configurarlo e distribuirlo nel tuo account.

Sections

- Architettura e codice del gestore
- Automazione della distribuzione con e AWS CloudFormationAWS CLI
- Strumentazione con AWS X-Ray
- Gestione delle dipendenze con i livelli

Architettura e codice del gestore

L'applicazione di esempio è composta da codice di funzione, un AWS CloudFormation modello e risorse di supporto. Quando si distribuisce l'esempio, si utilizzano i seguenti AWS servizi:

- AWS Lambda — Esegue il codice della funzione, invia i log ai CloudWatch registri e invia i dati di traccia a X-Ray. La funzione richiama anche l'API Lambda per ottenere dettagli sulle quote e sull'utilizzo dell'account nella regione corrente.
- AWS X-Ray – Raccoglie i dati di tracciamento, indicizza i tracciamenti per la ricerca e genera una mappa del servizio.
- Amazon CloudWatch: archivia log e metriche.
- AWS Identity and Access Management (IAM) — Concede l'autorizzazione.
- Amazon Simple Storage Service (Amazon S3) – Memorizza il pacchetto di implementazione della funzione durante l'implementazione.
- AWS CloudFormation – Crea le risorse dell'applicazione e distribuisce il codice della funzione.

Per ogni servizio sono previsti costi standard. Per ulteriori informazioni, consulta la sezione Prezzi di AWS.

Il codice della funzione mostra un flusso di lavoro di base per l'elaborazione di un evento. Il gestore accetta un evento Amazon Simple Queue Service (Amazon SQS) come input e itera attraverso i record che contiene, aggiungendo al log il contenuto di ogni messaggio. Aggiunge ai registri il contenuto dell'evento, l'oggetto contesto e le variabili di ambiente. Quindi effettua una chiamata con l' AWS SDK e restituisce la risposta al runtime Lambda.

Example blank-nodejs/function/index.js – Codice del gestore

```
// Handler
exports.handler = async function(event, context) {
    event.Records.forEach(record => {
        console.log(record.body);
    });

    console.log('## ENVIRONMENT VARIABLES: ' + serialize(process.env));
    console.log('## CONTEXT: ' + serialize(context));
    console.log('## EVENT: ' + serialize(event));

    return getAccountSettings();
};

// Use SDK client
var getAccountSettings = function() {
    return lambda.send(new GetAccountSettingsCommand());
};
```

```
var serialize = function(object) {
    return JSON.stringify(object, null, 2);
};
```

L'applicazione di esempio non include una coda Amazon SQS per l'invio di eventi, ma utilizza un evento da Amazon SQS (event.json) per illustrare come vengono elaborati gli eventi. Per aggiungere una coda Amazon SQS all'applicazione, consulta Utilizzo di Lambda con Amazon SQS.

Automazione della distribuzione con e AWS CloudFormationAWS CLI

Le risorse dell'applicazione di esempio sono definite in un AWS CloudFormation modello e distribuite con. AWS CLI Il progetto include semplici script di shell che automatizzano il processo di configurazione, distribuzione, invocazione e terminazione dell'applicazione.

Il modello di applicazione utilizza un tipo di risorsa AWS Serverless Application Model (AWS SAM) per definire il modello. AWS SAM semplifica la creazione di modelli per applicazioni serverless automatizzando la definizione di ruoli di esecuzione, API e altre risorse.

Il modello definisce le risorse nello stackdell'applicazione. Ciò include la funzione, il suo ruolo di esecuzione e un livello Lambda che fornisce le dipendenze dalle librerie della funzione. Lo stack non include il bucket AWS CLI utilizzato durante la distribuzione o il gruppo di log Logs. CloudWatch

Example blank-nodejs/template.yml – Risorse serverless

```
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:
      Handler: index.handler
      Runtime: nodejs20.x
      CodeUri: function/.
      Description: Call the AWS Lambda API
      Timeout: 10
      # Function's execution role
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambda_ReadOnlyAccess
        - AWSXrayWriteOnlyAccess
```

```
      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:
        - nodejs20.x
```

Quando distribuisci l'applicazione, AWS CloudFormation applica la AWS SAM trasformazione al modello per generare un AWS CloudFormation modello con tipi standard come e. `AWS::Lambda::Function` `AWS::IAM::Role`

Example Modello elaborato

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "An AWS Lambda application that calls the Lambda API.",
  "Resources": {
    "function": {
      "Type": "AWS::Lambda::Function",
      "Properties": {
        "Layers": [
          {
            "Ref": "libs32xmpl61b2"
          }
        ],
        "TracingConfig": {
          "Mode": "Active"
        },
        "Code": {
          "S3Bucket": "lambda-artifacts-6b000xmpl1e9bf2a",
          "S3Key": "3d3axmpl473d249d039d2d7a37512db3"
        },
        "Description": "Call the AWS Lambda API",
        "Tags": [
          {
            "Value": "SAM",
            "Key": "lambda:createdBy"
          }
```

```
        ],
```

In questo esempio, la proprietà `Code` specifica un oggetto in un bucket Amazon S3. Questo corrisponde al percorso locale nella proprietà `CodeUri` nel modello di progetto:

```
      CodeUri: function/.
```

Per caricare i file di progetto su Amazon S3, lo script di implementazione utilizza i comandi nella AWS CLI. Il comando `cloudformation package` pre-elabora il modello, carica gli artefatti e sostituisce i percorsi locali con le posizioni degli oggetti su Amazon S3. Il `cloudformation deploy` comando distribuisce il modello elaborato con un set di AWS CloudFormation modifiche.

Example blank-nodejs/3-deploy.sh – Pacchetto e implementazione

```
#!/bin/bash
set -eo pipefail
ARTIFACT_BUCKET=$(cat bucket-name.txt)
aws cloudformation package --template-file template.yml --s3-bucket $ARTIFACT_BUCKET --
output-template-file out.yml
aws cloudformation deploy --template-file out.yml --stack-name blank-nodejs --
capabilities CAPABILITY_NAMED_IAM
```

La prima volta che si esegue questo script, viene creato uno AWS CloudFormation stack denominato. `blank-nodejs` Se si apportano modifiche al codice della funzione o al modello, è possibile eseguirlo di nuovo per aggiornare lo stack.

Lo script di pulizia (blank-nodejs/5-cleanup.sh) elimina lo stack e, facoltativamente, elimina il bucket di distribuzione e i log delle funzioni.

Strumentazione con AWS X-Ray

La funzione di esempio è configurata per il tracciamento AWS X-Ray. Con la modalità di tracciamento impostata su attiva, Lambda registra le informazioni temporali per un sottoinsieme di invocazioni e le invia a X-Ray. X-Ray elabora i dati per generare una mappa servizio che mostra un nodo client e due nodi di servizio.

Il primo nodo di servizio (`AWS::Lambda`) rappresenta il servizio Lambda, che convalida la richiesta di invocazione e la invia alla funzione. Il secondo nodo, `AWS::Lambda::Function`, rappresenta la funzione stessa.

Per registrare ulteriori dettagli, la funzione di esempio utilizza l'SDK X-Ray. Con modifiche minime al codice della funzione, l'SDK X-Ray registra i dettagli sulle chiamate effettuate con l' AWS SDK ai servizi. AWS

Example blank-nodejs/function/index.js – Strumentazione

```
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());
```

La strumentazione del client AWS SDK aggiunge un nodo aggiuntivo alla mappa dei servizi e ulteriori dettagli nelle tracce. In questo esempio, la mappa del servizio mostra la funzione di esempio che richiama l'API Lambda per ottenere dettagli sull'archiviazione e sull'utilizzo della concorrenza nella regione corrente.

Il tracciamento mostra i dettagli temporali della chiamata, con sottosegmenti per l'inizializzazione della funzione, l'invocazione e l'overhead. Il sottosegmento di invocazione ha un sottosegmento per la chiamata SDK all' AWS operazione API. `GetAccountSettings`

Puoi includere l'SDK X-Ray e altre librerie nel pacchetto di implementazione della funzione o distribuirle separatamente in un livello Lambda. Per Node.js, Ruby e Python, il runtime Lambda include l'SDK AWS nell'ambiente di esecuzione.

Gestione delle dipendenze con i livelli

È possibile installare le librerie localmente e includerle nel pacchetto di distribuzione caricato su Lambda, ma ciò ha i suoi svantaggi. Dimensioni dei file maggiori aumentano i tempi di distribuzione e possono impedire di testare le modifiche al codice della funzione nella console di Lambda. Per mantenere il pacchetto di distribuzione di dimensioni ridotte ed evitare di caricare dipendenze non modificate, l'app di esempio crea un ivello Lambda e lo associa alla funzione.

Example blank-nodejs/template.yml – Livello delle dipendenze

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
```

```
      Runtime: nodejs20.x
      CodeUri: function/.
      Description: Call the AWS Lambda API
      Timeout: 10
      # Function's execution role
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambda_ReadOnlyAccess
        - AWSXrayWriteOnlyAccess
      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:
        - nodejs20.x
```

Lo script `2-build-layer.sh` installa le dipendenze della funzione con npm e le inserisce in una cartella con la struttura richiesta dal runtime di Lambda.

Example 2-build-layer.sh – Preparazione del livello

```
#!/bin/bash
set -eo pipefail
mkdir -p lib/nodejs
rm -rf node_modules lib/nodejs/node_modules
npm install --production
mv node_modules lib/nodejs/
```

La prima volta che si distribuisce l'applicazione di esempio, AWS CLI impacchetta il layer separatamente dal codice della funzione e li distribuisce entrambi. Nelle distribuzioni successive, l'archivio del layer viene caricato solo se i contenuti della cartella `lib` sono stati modificati.

Usare Lambda con un SDK AWS

AWS I kit di sviluppo software (SDK) sono disponibili per molti linguaggi di programmazione più 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
AWS SDK for C++	AWS SDK for C++ esempi di codice
AWS CLI	AWS CLI esempi di codice
AWS SDK for Go	AWS SDK for Go esempi di codice
AWS SDK for Java	AWS SDK for Java esempi di codice
AWS SDK for JavaScript	AWS SDK for JavaScript esempi di codice
SDK AWS for Kotlin	SDK AWS for Kotlin esempi di codice
AWS SDK for .NET	AWS SDK for .NET esempi di codice
AWS SDK for PHP	AWS SDK for PHP esempi di codice
AWS Tools for PowerShell	Strumenti per esempi di PowerShell codice
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) esempi di codice
AWS SDK for Ruby	AWS SDK for Ruby esempi di codice
AWS SDK for Rust	AWS SDK for Rust esempi di codice
SDK AWS per SAP ABAP	SDK AWS per SAP ABAP esempi di codice
SDK AWS per Swift	SDK AWS per Swift esempi di codice

Per esempi specifici relativi a Lambda, consulta Esempi di codice per Lambda con SDK AWS.

Esempio di disponibilità

Non riesci a trovare quello che ti serve? Richiedi un esempio di codice utilizzando il link Provide feedback (Fornisci un feedback) nella parte inferiore di questa pagina.

Esempi di codice per Lambda con SDK AWS

I seguenti esempi di codice mostrano come usare Lambda con un kit di sviluppo AWS software (SDK).

Le operazioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le operazioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati e negli esempi tra servizi.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica richiamando più funzioni all'interno dello stesso servizio.

Esempi cross-service: applicazioni di esempio che funzionano su più servizi Servizi AWS.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Nozioni di base

Hello Lambda

L'esempio di codice seguente mostra come iniziare a utilizzare Lambda.

.NET

AWS SDK for .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 ListFunctionssezione AWS SDK for .NET API Reference.

C++

SDK per C++

> **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.

Codice per il file CMake C MakeLists .txt.

```
# 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, consulta API ListFunctionsReference AWS SDK for C++ .

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.

```
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() {
 sdkConfig, err := config.LoadDefaultConfig(context.TODO())
 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(context.TODO(),
 &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%v\n", *function.FunctionName)
  }
 }
}
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK for Go API Reference.

Java

SDK per Java 2.x

> **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.

```
package com.example.lambda;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.LambdaException;
import software.amazon.awssdk.services.lambda.model.ListFunctionsResponse;
import software.amazon.awssdk.services.lambda.model.FunctionConfiguration;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListLambdaFunctions {
    public static void main(String[] args) {
        Region region = Region.US_WEST_2;
        LambdaClient awsLambda = LambdaClient.builder()
                .region(region)
                .build();

        listFunctions(awsLambda);
        awsLambda.close();
    }
```

```
    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 ListFunctionssezione 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 ListFunctionssezione AWS SDK for JavaScript API Reference.

Esempi di codice

- Azioni per Lambda tramite SDK AWS
  - Utilizzo CreateAlias con un AWS SDK o una CLI
  - Utilizzo CreateFunction con un AWS SDK o una CLI
  - Utilizzo DeleteAlias con un AWS SDK o una CLI
  - Utilizzo DeleteFunction con un AWS SDK o una CLI
  - Utilizzo DeleteFunctionConcurrency con un AWS SDK o una CLI
  - Utilizzo DeleteProvisionedConcurrencyConfig con un AWS SDK o una CLI
  - Utilizzo GetAccountSettings con un AWS SDK o una CLI
  - Utilizzo GetAlias con un AWS SDK o una CLI
  - Utilizzo GetFunction con un AWS SDK o una CLI
  - Utilizzo GetFunctionConcurrency con un AWS SDK o una CLI
  - Utilizzo GetFunctionConfiguration con un AWS SDK o una CLI
  - Utilizzo GetPolicy con un AWS SDK o una CLI
  - Utilizzo GetProvisionedConcurrencyConfig con un AWS SDK o una CLI
  - Utilizzo Invoke con un AWS SDK o una CLI
  - Utilizzo ListFunctions con un AWS SDK o una CLI
  - Utilizzo ListProvisionedConcurrencyConfigs con un AWS SDK o una CLI
  - Utilizzo ListTags con un AWS SDK o una CLI
  - Utilizzo ListVersionsByFunction con un AWS SDK o una CLI
  - Utilizzo PublishVersion con un AWS SDK o una CLI

- 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
- Richiamo a una funzione Lambda da un browser
- Trasforma i dati per la tua applicazione con S3 Object Lambda
- Utilizzo di un'API Gateway per richiamare una funzione Lambda
- Utilizzo di Step Functions per richiamare le funzioni Lambda
- Utilizzo degli eventi pianificati per richiamare una funzione Lambda

Azioni per Lambda tramite SDK AWS

I seguenti esempi di codice mostrano come eseguire singole azioni Lambda con AWS gli SDK. Questi estratti chiamano l'API Lambda e sono estratti di codice da programmi più grandi che devono essere eseguiti in modo contestuale. Ogni esempio include un collegamento a GitHub, dove è possibile trovare le istruzioni per la configurazione e l'esecuzione del codice.

Gli esempi seguenti includono solo le operazioni più comunemente utilizzate. Per un elenco completo, consulta la Documentazione di riferimento delle API AWS Lambda.

Esempi

- Utilizzo CreateAlias con un AWS SDK o una CLI
- Utilizzo CreateFunction con un AWS SDK o una CLI
- Utilizzo DeleteAlias con un AWS SDK o una CLI
- Utilizzo DeleteFunction con un AWS SDK o una CLI
- Utilizzo DeleteFunctionConcurrency con un AWS SDK o una CLI
- Utilizzo DeleteProvisionedConcurrencyConfig con un AWS SDK o una CLI
- Utilizzo GetAccountSettings con un AWS SDK o una CLI
- Utilizzo GetAlias con un AWS SDK o una CLI
- Utilizzo GetFunction con un AWS SDK o una CLI
- Utilizzo GetFunctionConcurrency con un AWS SDK o una CLI
- Utilizzo GetFunctionConfiguration con un AWS SDK o una CLI

- Utilizzo GetPolicy con un AWS SDK o una CLI
- Utilizzo GetProvisionedConcurrencyConfig con un AWS SDK o una CLI
- Utilizzo Invoke con un AWS SDK o una CLI
- Utilizzo ListFunctions con un AWS SDK o una CLI
- Utilizzo ListProvisionedConcurrencyConfigs con un AWS SDK o una CLI
- Utilizzo ListTags con un AWS SDK o una CLI
- Utilizzo ListVersionsByFunction con un AWS SDK o una CLI
- Utilizzo PublishVersion con un AWS SDK o una CLI
- Utilizzo PutFunctionConcurrency con un AWS SDK o una CLI
- Utilizzo PutProvisionedConcurrencyConfig con un AWS SDK o una CLI
- Utilizzo RemovePermission con un AWS SDK o una CLI
- Utilizzo TagResource con un AWS SDK o una CLI
- Utilizzo UntagResource con un AWS SDK o una CLI
- Utilizzo UpdateAlias con un AWS SDK o una CLI
- Utilizzo UpdateFunctionCode con un AWS SDK o una CLI
- Utilizzo UpdateFunctionConfiguration con un AWS SDK o una CLI

Utilizzo **`CreateAlias`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`CreateAlias`.

CLI

AWS CLI

Per creare un alias per una funzione Lambda

L'`create-alias`esempio seguente crea un alias denominato `LIVE` che punta alla versione 1 della funzione `my-function` Lambda.

```
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 CreateAliasCommand Reference.AWS CLI

PowerShell

Strumenti per 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 chiamata che riceve.

```
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 in Cmdlet Reference. CreateAliasAWS Tools for PowerShell

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`CreateFunction`** con un AWS SDK o una CLI

I seguenti esempi di codice 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 nel seguente esempio di codice:

- Nozioni di base sulle funzioni

.NET

AWS SDK for .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 CreateFunctionsezione AWS SDK for .NET API Reference.

C++

SDK per C++

> **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.

```
        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_8);
#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 CreateFunctionsezione AWS SDK for C++ API Reference.

CLI

AWS CLI

Per 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 nodejs18.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": "2023-10-14T22:26:11.234+0000",
    "Handler": "my-function.handler",
    "Runtime": "nodejs18.x",
    "Description": ""
}
```

Per ulteriori informazioni, consulta Configurazione della funzione Lambda AWS nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta CreateFunction AWS CLICommand 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.

```
// 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(functionName string, handlerName
 string,
 iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.CreateFunction(context.TODO(),
 &lambda.CreateFunctionInput{
  Code:         &types.FunctionCode{ZipFile: zipPackage.Bytes()},
  FunctionName: aws.String(functionName),
  Role:         iamRoleArn,
  Handler:      aws.String(handlerName),
  Publish:      true,
  Runtime:      types.RuntimePython38,
 })
 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(context.TODO(),
 &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 CreateFunctionsezione AWS SDK for Go API Reference.

Java

SDK per Java 2.x

> **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 software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.CreateFunctionRequest;
import software.amazon.awssdk.services.lambda.model.FunctionCode;
import software.amazon.awssdk.services.lambda.model.CreateFunctionResponse;
import software.amazon.awssdk.services.lambda.model.GetFunctionRequest;
import software.amazon.awssdk.services.lambda.model.GetFunctionResponse;
import software.amazon.awssdk.services.lambda.model.LambdaException;
import software.amazon.awssdk.services.lambda.model.Runtime;
import software.amazon.awssdk.services.lambda.waiters.LambdaWaiter;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;

/**
 * This code example requires a ZIP or JAR that represents the code of the
 * Lambda function.
 * If you do not have a ZIP or JAR, please refer to the following document:
 *
 * https://github.com/aws-doc-sdk-examples/tree/master/javav2/usecases/
creating_workflows_stepfunctions
```

```
 *
 * Also, set up your development environment, including your credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class CreateFunction {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <functionName> <filePath> <role> <handler>\s

                Where:
                    functionName - The name of the Lambda function.\s
                    filePath - The path to the ZIP or JAR where the code is
 located.\s
                    role - The role ARN that has Lambda permissions.\s
                    handler - The fully qualified method name (for example,
 example.Handler::handleRequest). \s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String functionName = args[0];
        String filePath = args[1];
        String role = args[2];
        String handler = args[3];
        Region region = Region.US_WEST_2;
        LambdaClient awsLambda = LambdaClient.builder()
                .region(region)
                .build();

        createLambdaFunction(awsLambda, functionName, filePath, role, handler);
        awsLambda.close();
    }
```

```
    public static void createLambdaFunction(LambdaClient awsLambda,
            String functionName,
            String filePath,
            String role,
            String handler) {

        try {
            LambdaWaiter waiter = awsLambda.waiter();
            InputStream is = new FileInputStream(filePath);
            SdkBytes fileToUpload = SdkBytes.fromInputStream(is);

            FunctionCode code = FunctionCode.builder()
                    .zipFile(fileToUpload)
                    .build();

            CreateFunctionRequest functionRequest =
 CreateFunctionRequest.builder()
                    .functionName(functionName)
                    .description("Created by the Lambda Java API")
                    .code(code)
                    .handler(handler)
                    .runtime(Runtime.JAVA8)
                    .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);
            System.out.println("The function ARN is " +
 functionResponse.functionArn());

        } catch (LambdaException | FileNotFoundException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Per i dettagli sull'API, consulta la CreateFunctionsezione 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 CreateFunctionsezione AWS SDK for JavaScript API Reference.

Kotlin

SDK per Kotlin

> **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.

```
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.Java8
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val functionResponse = awsLambda.createFunction(request)
        awsLambda.waitUntilFunctionActive {
            functionName = myFunctionName
        }
        return functionResponse.functionArn
    }
}
```

- Per i dettagli sull'API, CreateFunctionconsulta 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 CreateFunctionsezione AWS SDK for PHP API Reference.

PowerShell

Strumenti per 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 designazione: :Namespace. AssemblyName ClassName::. MethodName È necessario sostituire in modo appropriato le parti relative al nome dell'assembly (senza il suffisso .dll), allo spazio dei nomi, al nome della classe e al nome del metodo nelle specifiche del gestore. 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 mybucket -Key MyFunctionBinaries.zip -File .
\MyFunctionBinaries.zip
Publish-LMFunction -Description "My C# Lambda Function" `
```

```
        -FunctionName MyFunction `
        -BucketName mybucket `
        -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 in Cmdlet Reference. CreateFunctionAWS Tools for PowerShell

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="python3.8",
                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 AWSSDK 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
```

```
 def initialize
   @lambda_client = Aws::Lambda::Client.new
   @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. This
 #                      must include the file name and the function name.
 # @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 CreateFunctionsezione AWS SDK for Ruby API Reference.

Rust

SDK per Rust

> **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.

```
    /**
     * 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 CreateFunctionguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **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.

```
    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_lmdcodesigningcfgno00.
        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, CreateFunctionconsulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DeleteAlias`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`DeleteAlias`.

CLI

AWS CLI

Per eliminare un alias di una funzione Lambda

L'`delete-alias`esempio 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 DeleteAliasCommand Reference.AWS CLI

PowerShell

Strumenti per PowerShell

Esempio 1: Questo esempio elimina la funzione Lambda Alias menzionata nel comando.

```
Remove-LMAlias -FunctionName "MylambdaFunction123" -Name "NewAlias"
```

- Per i dettagli sull'API, vedere DeleteAliasin AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DeleteFunction`** con un AWS SDK o una CLI

I seguenti esempi di codice 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 nel seguente esempio di codice:

- Nozioni di base sulle funzioni

.NET

AWS SDK for .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 DeleteFunctionsezione AWS SDK for .NET API Reference.

C++

SDK per C++

> **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.

```
        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 DeleteFunctionsezione AWS SDK for C++ API Reference.

CLI

AWS CLI

Esempio 1: eliminazione di 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 CLICommand 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.

```
// 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(functionName string) {
 _, err := wrapper.LambdaClient.DeleteFunction(context.TODO(),
 &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 DeleteFunctionsezione AWS SDK for Go API Reference.

Java

SDK per Java 2.x

> **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 software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.model.DeleteFunctionRequest;
import software.amazon.awssdk.services.lambda.model.LambdaException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteFunction {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <functionName>\s

                Where:
                    functionName - The name of the Lambda function.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String functionName = args[0];
        Region region = Region.US_EAST_1;
```

```
        LambdaClient awsLambda = LambdaClient.builder()
                .region(region)
                .build();

        deleteLambdaFunction(awsLambda, functionName);
        awsLambda.close();
    }

    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 DeleteFunctionsezione 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 DeleteFunctionsezione AWS SDK for JavaScript API Reference.

Kotlin

SDK per Kotlin

> **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.

```
suspend fun delLambdaFunction(myFunctionName: String) {
    val request =
        DeleteFunctionRequest {
            functionName = myFunctionName
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        awsLambda.deleteFunction(request)
        println("$myFunctionName was deleted")
    }
}
```

- Per i dettagli sull'API, DeleteFunctionconsulta 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 DeleteFunctionsezione AWS SDK for PHP API Reference.

PowerShell

Strumenti per 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 DeleteFunctionin AWS Tools for PowerShell Cmdlet 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 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 AWSSDK 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

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  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 DeleteFunctionsezione AWS SDK for Ruby API Reference.

Rust

SDK per Rust

> **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.

```
   /** 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 DeleteFunctionguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **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.

```
   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, DeleteFunctionconsulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DeleteFunctionConcurrency`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`DeleteFunctionConcurrency`.

CLI

AWS CLI

Per rimuovere il limite di esecuzione simultanea riservato da una funzione

L'`delete-function-concurrency`esempio 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 Reserved Concurrency for a Lambda Function nella Lambda Developer Guide.AWS

- Per i dettagli sull'API, consulta DeleteFunctionConcurrency in Command Reference.AWS CLI

PowerShell

Strumenti per PowerShell

Esempio 1: questo esempio rimuove la Function Concurrency della Funzione Lambda.

```
Remove-LMFunctionConcurrency -FunctionName "MylambdaFunction123"
```

- Per i dettagli sull'API, vedere DeleteFunctionConcurrency in Cmdlet ReferenceAWS Tools for PowerShell .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DeleteProvisionedConcurrencyConfig`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`DeleteProvisionedConcurrencyConfig`.

CLI

AWS CLI

Per eliminare una configurazione di concorrenza fornita

L'`delete-provisioned-concurrency-config`esempio seguente elimina la configurazione di concorrenza fornita per l'`GREEN`alias della funzione specificata.

```
aws lambda delete-provisioned-concurrency-config \
    --function-name my-function \
    --qualifier GREEN
```

- Per i dettagli sull'API, vedere DeleteProvisionedConcurrencyConfigin Command Reference.AWS CLI

PowerShell

Strumenti per PowerShell

Esempio 1: Questo esempio rimuove la configurazione Provisioned Concurrency per un alias specifico.

```
Remove-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
Qualifier "NewAlias1"
```

- Per i dettagli sull'API, vedere DeleteProvisionedConcurrencyConfigin AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetAccountSettings`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`GetAccountSettings`.

CLI

AWS CLI

Per recuperare i dettagli sul tuo account in una regione AWS

L'`get-account-settings`esempio 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 AWS Lambda Limits nella Lambda Developer AWS Guide.

- Per i dettagli sull'API, consulta GetAccountSettings in AWS CLI Command Reference.

PowerShell

Strumenti per 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 AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetAlias`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`GetAlias`.

CLI

AWS CLI

Per recuperare i dettagli sull'alias di una funzione

L'`get-alias`esempio seguente visualizza i dettagli per l'alias denominato nella `LIVE` funzione `my-function` Lambda.

```
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 GetAliasCommand Reference.AWS CLI

PowerShell

Strumenti per PowerShell

Esempio 1: questo esempio recupera i pesi di Routing Config 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 in Cmdlet Reference. GetAliasAWS Tools for PowerShell

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetFunction`** con un AWS SDK o una CLI

I seguenti esempi di codice 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 nel seguente esempio di codice:

- Nozioni di base sulle funzioni

.NET

AWS SDK for .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 GetFunctionsezione AWS SDK for .NET API Reference.

C++

SDK per C++

> **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.

```
        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 GetFunctionsezione AWS SDK for C++ API Reference.

CLI

AWS CLI

Per 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": "2019-09-24T18:20:35.054+0000",
        "Runtime": "nodejs10.x",
        "Description": ""
    }
}
```


Per ulteriori informazioni, consulta Configurazione della funzione Lambda AWS nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta GetFunction AWS CLICommand 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.

```
// 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(functionName string) types.State {
 var state types.State
 funcOutput, err := wrapper.LambdaClient.GetFunction(context.TODO(),
 &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 GetFunctionsezione AWS SDK for 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 getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la GetFunctionsezione AWS SDK for JavaScript 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 getFunction($functionName)
    {
        return $this->lambdaClient->getFunction([
            'FunctionName' => $functionName,
        ]);
    }
```

- Per i dettagli sull'API, consulta la GetFunctionsezione AWS SDK for PHP API Reference.

Python

SDK per Python (Boto3)

> **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:
    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 AWSSDK 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

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @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
```

- Per i dettagli sull'API, consulta la GetFunctionsezione AWS SDK for Ruby API Reference.

Rust

SDK per Rust

> **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.

```
    /** 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 GetFunctionguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **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.

```
    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, GetFunctionconsulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetFunctionConcurrency`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`GetFunctionConcurrency`.

CLI

AWS CLI

Per visualizzare l'impostazione di concorrenza riservata per una funzione

L'`get-function-concurrency`esempio seguente recupera l'impostazione della concorrenza riservata per la funzione specificata.

```
aws lambda get-function-concurrency \
    --function-name my-function
```

Output:

```
{
    "ReservedConcurrentExecutions": 250
}
```

- Per i dettagli sull'API, vedere GetFunctionConcurrency in AWS CLI Command Reference.

PowerShell

Strumenti per PowerShell

Esempio 1: questo esempio ottiene la concorrenza riservata per la funzione Lambda

```
Get-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -Select *
```

Output:

```
ReservedConcurrentExecutions
----------------------------
100
```

- Per i dettagli sull'API, vedere GetFunctionConcurrency in Cmdlet ReferenceAWS Tools for PowerShell .

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetFunctionConfiguration`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`GetFunctionConfiguration`.

CLI

AWS CLI

Per recuperare le impostazioni specifiche della versione di una funzione Lambda

L'`get-function-configuration`esempio 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-
uy3l9qyq",
    "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 della funzione Lambda AWS nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta GetFunctionConfiguration in AWS CLI Command Reference.

PowerShell

Strumenti per PowerShell

Esempio 1: questo esempio restituisce la configurazione specifica della versione di una funzione Lambda.

```
Get-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Qualifier
 "PowershellAlias"
```

Output:

```
CodeSha256                  : uWOW0R7z+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 AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetPolicy`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`GetPolicy`.

CLI

AWS CLI

Per recuperare la policy IAM basata sulle risorse per una funzione, una versione o un alias

L'`get-policy`esempio seguente visualizza informazioni sulle politiche relative alla funzione `my-function` Lambda.

```
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 Lambda nella AWS Lambda Developer Guide.AWS

- Per i dettagli sull'API, consulta Command Reference. GetPolicyAWS CLI

PowerShell

Strumenti per PowerShell

Esempio 1: questo esempio mostra la politica 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:lamb
east-1:123456789102:function:test"}]}
```

- Per i dettagli sull'API, vedere GetPolicyin AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetProvisionedConcurrencyConfig`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`GetProvisionedConcurrencyConfig`.

CLI

AWS CLI

Per visualizzare una configurazione di concorrenza fornita

L'`get-provisioned-concurrency-config`esempio seguente visualizza i dettagli della configurazione di concorrenza fornita per l'BLUEalias 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, vedere GetProvisionedConcurrencyConfigin AWS CLI Command Reference.

PowerShell

Strumenti per PowerShell

Esempio 1: questo esempio ottiene la configurazione simultanea 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 GetProvisionedConcurrencyConfigin Cmdlet Reference.AWS Tools for PowerShell

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`Invoke`** con un AWS SDK o una CLI

I seguenti esempi di codice 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 nel seguente esempio di codice:

- Nozioni di base sulle funzioni

.NET

AWS SDK for .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 sulle API, consulta Invoke nella Documentazione di riferimento delle API AWS SDK for .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 sulle API, consulta Invoke nella Documentazione di riferimento delle API AWS SDK for C++ .

CLI

AWS CLI

Esempio 1: come richiamare una funzione Lambda in modo sincrono

L'esempio di `invoke` seguente richiama la funzione `my-function` in modo sincrono. L'`cli-binary-format`opzione 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 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 Chiamata sincrona nella Guida per gli sviluppatori di AWS Lambda.

Esempio 2: come richiamare una funzione Lambda in modo asincrono

L'esempio di `invoke` seguente richiama la funzione `my-function` in modo asincrono. L'`cli-binary-format`opzione 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 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 Chiamata asincrona 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 altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
// 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(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(context.TODO(),
 &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 sulle API, consulta Invoke nella Documentazione di riferimento delle API AWS SDK for 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.

```
import org.json.JSONObject;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.model.InvokeRequest;
import software.amazon.awssdk.core.SdkBytes;
```

```
import software.amazon.awssdk.services.lambda.model.InvokeResponse;
import software.amazon.awssdk.services.lambda.model.LambdaException;

public class LambdaInvoke {

    /*
     * Function names appear as
     * arn:aws:lambda:us-west-2:335556666777:function:HelloFunction
     * you can retrieve the value by looking at the function in the AWS Console
     *
     * Also, set up your development environment, including your credentials.
     *
     * For information, see this documentation topic:
     *
     * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.
     * html
     */

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <functionName>\s

                Where:
                    functionName - The name of the Lambda function\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String functionName = args[0];
        Region region = Region.US_WEST_2;
        LambdaClient awsLambda = LambdaClient.builder()
                .region(region)
                .build();

        invokeFunction(awsLambda, functionName);
        awsLambda.close();
    }
```

```
    public static void invokeFunction(LambdaClient awsLambda, String
 functionName) {

        InvokeResponse res = null;
        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);

            // Setup an InvokeRequest.
            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 sulle API, consulta Invoke nella Documentazione di riferimento delle 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 sulle API, consulta Invoke nella Documentazione di riferimento delle API AWS SDK for 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 sulle API, consulta Invoke nella Documentazione di riferimento delle API SDK AWS 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 sulle API, consulta Invoke nella Documentazione di riferimento delle API AWS SDK for 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 sulle API, consulta Invoke nella Documentazione di riferimento delle API SDK AWS 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

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @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 sulle API, consulta Invoke nella Documentazione di riferimento delle API AWS SDK for Ruby .

Rust

SDK per Rust

> **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.

```
    /** 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 della documentazione di riferimento dell'API SDK AWS per Rust.

SAP ABAP

SDK per SAP ABAP

> **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.

```
   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_lmdinvrequestcontex.
       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'SDK AWS per l'API SAP ABAP.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`ListFunctions`** con un AWS SDK o una CLI

I seguenti esempi di codice 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 nel seguente esempio di codice:

- Nozioni di base sulle funzioni

.NET

AWS SDK for .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 ListFunctionssezione AWS SDK for .NET API Reference.

C++

SDK per C++

> **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.

```
        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 ListFunctionssezione AWS SDK for 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": "$LATEST",
            "CodeSha256": "dBG9m8SGdmlEjw/JYXlhhvCrAv5TxvXsbL/RMr0fT/I=",
            "FunctionName": "helloworld",
            "MemorySize": 128,
            "RevisionId": "1718e831-badf-4253-9518-d0644210af7b",
            "CodeSize": 294,
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:helloworld",
            "Handler": "helloworld.handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-
role-zgur6bf4",
            "Timeout": 3,
            "LastModified": "2023-09-23T18:32:33.857+0000",
            "Runtime": "nodejs18.x",
            "Description": ""
        },
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "$LATEST",
            "CodeSha256": "sU0cJ2/hOZevwV/lTxCuQqK3gDZP3i8gUoqUUVRmY6E=",
            "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-uy3l9qyq",
```

```
            "Timeout": 3,
            "LastModified": "2023-10-01T16:47:28.490+0000",
            "Runtime": "nodejs18.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": "2023-10-01T19:40:41.643+0000",
            "Runtime": "python3.11",
            "Description": ""
        }
    ]
```

```
}
```

Per ulteriori informazioni, consulta Configurazione della funzione Lambda AWS nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta ListFunctions AWS CLICommand 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.

```
// 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(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(context.TODO())
  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 ListFunctionssezione AWS SDK for 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 ListFunctionssezione AWS SDK for JavaScript 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 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 ListFunctionssezione AWS SDK for PHP API Reference.

PowerShell

Strumenti per 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 ListFunctionsin AWS Tools for PowerShell Cmdlet 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 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 AWSSDK 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

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @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
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error executing #{function_name}:\n
 #{e.message}")
  end
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK for Ruby API Reference.

Rust

SDK per Rust

> **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.

```
   /** 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 ListFunctionsguida di riferimento all'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **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.

```
   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, ListFunctionsconsulta AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`ListProvisionedConcurrencyConfigs`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`ListProvisionedConcurrencyConfigs`.

CLI

AWS CLI

Per ottenere un elenco delle configurazioni di concorrenza fornite

L'`list-provisioned-concurrency-configs`esempio seguente elenca le configurazioni di concorrenza assegnate 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, vedere ListProvisionedConcurrencyConfigsin AWS CLI Command Reference.

PowerShell

Strumenti per PowerShell

Esempio 1: questo esempio recupera l'elenco delle configurazioni di concorrenza assegnate per una funzione Lambda.

```
Get-LMProvisionedConcurrencyConfigList -FunctionName "MylambdaFunction123"
```

- Per i dettagli sull'API, vedere in Cmdlet Reference. ListProvisionedConcurrencyConfigsAWS Tools for PowerShell

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`ListTags`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`ListTags`.

CLI

AWS CLI

Per recuperare l'elenco dei tag per una funzione Lambda

L'`list-tags`esempio seguente visualizza i tag associati alla funzione `my-function` Lambda.

```
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 Lambda Functions nella Lambda Developer AWS Guide.

- Per i dettagli sull'API, consulta Command ListTagsReference AWS CLI .

PowerShell

Strumenti per 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 ListTagsin AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`ListVersionsByFunction`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`ListVersionsByFunction`.

CLI

AWS CLI

Per recuperare un elenco di versioni di una funzione

L'`list-versions-by-function`esempio seguente visualizza l'elenco delle versioni della funzione `my-function` Lambda.

```
aws lambda list-versions-by-function \
    --function-name my-function
```

Output:

```
{
    "Versions": [
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "$LATEST",
            "CodeSha256": "sU0cJ2/hOZevwV/lTxCuQqK3gDZP3i8gUoqUUVRmY6E=",
            "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-uy3l9qyq",
            "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-uy3l9qyq",
            "Timeout": 3,
            "LastModified": "2019-09-26T20:28:40.438+0000",
            "Runtime": "nodejs10.x",
            "Description": "new version"
        },
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "2",
            "CodeSha256": "sU0cJ2/hOZevwV/lTxCuQqK3gDZP3i8gUoqUUVRmY6E=",
            "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-uy3l9qyq",
            "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 ListVersionsByFunctionCommand Reference.AWS CLI

PowerShell

Strumenti per PowerShell

Esempio 1: questo esempio restituisce l'elenco delle configurazioni specifiche della versione per ogni versione della funzione Lambda.

```
Get-LMVersionsByFunction -FunctionName "MylambdaFunction123"
```

Output:

```
FunctionName        Runtime   MemorySize Timeout CodeSize LastModified
      RoleName
------------        -------   ---------- ------- -------- ------------
      --------
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 ListVersionsByFunctionin AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`PublishVersion`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`PublishVersion`.

CLI

AWS CLI

Per pubblicare una nuova versione di una funzione

L'`publish-version`esempio seguente pubblica una nuova versione della funzione `my-function` Lambda.

```
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 PublishVersionCommand Reference.AWS CLI

PowerShell

Strumenti per PowerShell

Esempio 1: questo esempio crea una versione per l'istantanea esistente di Lambda Function Code

```
Publish-LMVersion -FunctionName "MylambdaFunction123" -Description "Publishing
 Existing Snapshot of function code as a  new version through Powershell"
```

- Per i dettagli sull'API, vedere PublishVersionin AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`PutFunctionConcurrency`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`PutFunctionConcurrency`.

CLI

AWS CLI

Per configurare un limite di concorrenza riservato per una funzione

L'`put-function-concurrency`esempio 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 Reserved Concurrency for a Lambda Function nella Lambda Developer Guide.AWS

- Per i dettagli sull'API, consulta PutFunctionConcurrency in Command Reference.AWS CLI

PowerShell

Strumenti per PowerShell

Esempio 1: Questo esempio applica le impostazioni di concorrenza per la funzione nel suo insieme.

```
Write-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -
ReservedConcurrentExecution 100
```

- Per i dettagli sull'API, vedere PutFunctionConcurrency in AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **PutProvisionedConcurrencyConfig** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzarePutProvisionedConcurrencyConfig.

CLI

AWS CLI

Per allocare la concorrenza fornita

L'put-provisioned-concurrency-configesempio seguente alloca 100 valute simultanee assegnate per l'alias della BLUE 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, vedere PutProvisionedConcurrencyConfigin Command Reference.AWS CLI

PowerShell

Strumenti per PowerShell

Esempio 1: Questo esempio aggiunge una configurazione di concorrenza fornita all'alias di una funzione

```
Write-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
ProvisionedConcurrentExecution 20 -Qualifier "NewAlias1"
```

- Per i dettagli sull'API, vedere PutProvisionedConcurrencyConfigin AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`RemovePermission`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`RemovePermission`.

CLI

AWS CLI

Per rimuovere le autorizzazioni da una funzione Lambda esistente

L'`remove-permission`esempio seguente rimuove l'autorizzazione a richiamare 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 Lambda nella AWS Lambda Developer Guide.AWS

- Per i dettagli sull'API, consulta Command Reference. RemovePermissionAWS CLI

PowerShell

Strumenti per 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 RemovePermissionin AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`TagResource`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`TagResource`.

CLI

AWS CLI

Per aggiungere tag a una funzione Lambda esistente

L'`tag-resource`esempio 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 Lambda Developer AWS Guide.

- Per i dettagli sull'API, consulta Command TagResourceReference AWS CLI .

PowerShell

Strumenti per 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 TagResourcein AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`UntagResource`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`UntagResource`.

CLI

AWS CLI

Per rimuovere i tag da una funzione Lambda esistente

L'`untag-resource`esempio seguente rimuove il tag con il `DEPARTMENT` tag key name dalla funzione `my-function` Lambda.

```
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 Lambda Developer AWS Guide.

- Per i dettagli sull'API, consulta Command UntagResourceReference AWS CLI .

PowerShell

Strumenti per 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 UntagResourcein AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`UpdateAlias`** con un AWS SDK o una CLI

I seguenti esempi di codice mostrano come utilizzare`UpdateAlias`.

CLI

AWS CLI

Per aggiornare l'alias di una funzione

L'`update-alias`esempio seguente aggiorna l'alias denominato in modo `LIVE` che punti alla versione 3 della funzione `my-function` Lambda.

```
aws lambda update-alias \
    --function-name my-function \
    --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 UpdateAliasCommand Reference.AWS CLI

PowerShell

Strumenti per 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 UpdateAliasin AWS Tools for PowerShell Cmdlet Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`UpdateFunctionCode`** con un AWS SDK o una CLI

I seguenti esempi di codice 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:

- Nozioni di base sulle funzioni

.NET

AWS SDK for .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 UpdateFunctionCode in AWS SDK for .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;

#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
            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 UpdateFunctionCode in AWS SDK for 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-
uy3l9qyq",
    "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 della funzione Lambda AWS nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta UpdateFunctionCode in 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.

```
// 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(functionName string, zipPackage
 *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.UpdateFunctionCode(context.TODO(),
 &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(context.TODO(),
 &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 UpdateFunctionCode in AWS SDK for 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 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 UpdateFunctionCode in AWS SDK for 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 UpdateFunctionCode in AWS SDK for PHP API Reference.

PowerShell

Strumenti per PowerShell

Esempio 1: aggiorna la funzione denominata MyFunction '' con il nuovo 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 mybucket -Key
 UpdatedCode.zip
```

- Per i dettagli sull'API, consulta UpdateFunctionCode in AWS Tools for PowerShell Cmdlet 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 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 in AWS SDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **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
  attr_accessor :lambda_client

  def initialize
```

```
    @lambda_client = Aws::Lambda::Client.new
    @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
```

- Per i dettagli sull'API, consulta UpdateFunctionCode in AWS SDK for Ruby API Reference.

Rust

SDK per Rust

> **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.

```
    /** 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!("{}_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 sull'API, consulta UpdateFunctionCode in AWS SDK for Rust API reference.

SAP ABAP

SDK per SAP ABAP

> **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.

```
    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_lmdcodesigningcfgno00.
        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 sull'API, consulta UpdateFunctionCode in AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`UpdateFunctionConfiguration`** con un AWS SDK o una CLI

I seguenti esempi di codice 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:

- Nozioni di base sulle funzioni

.NET

AWS SDK for .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 UpdateFunctionConfiguration in AWS SDK for .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 UpdateFunctionConfiguration in AWS SDK for 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-
uy3l9qyq",
    "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 della funzione Lambda AWS nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta UpdateFunctionConfiguration in 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.

```
// 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(functionName string,
 envVars map[string]string) {
 _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(context.TODO(),
 &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 UpdateFunctionConfiguration in AWS SDK for 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 updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

- Per i dettagli sull'API, consulta UpdateFunctionConfiguration in AWS SDK for 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 UpdateFunctionConfiguration in AWS SDK for PHP API Reference.

PowerShell

Strumenti per 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 AWS Tools for PowerShell Cmdlet 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 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 in AWS SDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **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
  attr_accessor :lambda_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @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
```

- Per i dettagli sull'API, consulta UpdateFunctionConfiguration in AWS SDK for Ruby API Reference.

Rust

SDK per Rust

> **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.

```
/** 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 sull'API, consulta UpdateFunctionConfiguration in AWS SDK for Rust API reference.

SAP ABAP

SDK per SAP ABAP

> **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.

```
   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 sull'API, consulta UpdateFunctionConfiguration in AWS SDK for SAP ABAP API reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Scenari per Lambda con SDK AWS

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 in Lambda. Ogni scenario include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire il codice.

Esempi

- Conferma automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda utilizzando un SDK AWS
- Esegui automaticamente la migrazione di utenti Amazon Cognito noti con una funzione Lambda utilizzando un SDK AWS
- Inizia a creare e richiamare funzioni Lambda utilizzando un SDK AWS
- Scrivi dati di attività personalizzati con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito tramite un SDK AWS

Conferma automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda utilizzando un SDK AWS

Il seguente esempio di codice 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 `PreSignUp` trigger.
- Registra un utente con Amazon Cognito.
- La funzione Lambda analizza una tabella DynamoDB e conferma automaticamente gli utenti noti.
- Accedi come nuovo utente, quindi ripulisci 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.

```
// 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(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(
  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(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(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(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(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(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(stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup()
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))
```

```
 stackOutputs, err := runner.helper.GetStackOutputs(stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]
 runner.helper.PopulateUserTable(stackOutputs["TableName"])

 runner.AddPreSignUpTrigger(stackOutputs["UserPoolId"],
 stackOutputs["AutoConfirmFunctionArn"])
 runner.resources.triggers = append(runner.resources.triggers, actions.PreSignUp)
 userName, password := runner.SignUpUser(stackOutputs["UserPoolClientId"],
 stackOutputs["TableName"])
 runner.helper.ListRecentLogEvents(stackOutputs["AutoConfirmFunction"])
 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
  runner.SignInUser(stackOutputs["UserPoolClientId"], userName, password))

 runner.resources.Cleanup()

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Gestisci il `PreSignUp` grilletto con una funzione Lambda.

```
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() {
 sdkConfig, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Crea una struttura che esegua attività comuni.

```
// IScenarioHelper defines common functions used by the workflows in this
 example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(stackName string) (actions.StackOutputs, error)
 PopulateUserTable(tableName string)
 GetKnownUsers(tableName string) (actions.UserList, error)
 AddKnownUser(tableName string, user actions.User)
 ListRecentLogEvents(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(stackName string)
 (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(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(tableName)
```

```
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured
 format.
func (helper ScenarioHelper) GetKnownUsers(tableName string) (actions.UserList,
 error) {
 knownUsers, err := helper.dynamoActor.Scan(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(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(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(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(functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(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 racchiuda le azioni di Amazon Cognito.

```
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(userPoolId string,
 triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(context.TODO(),
 &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(context.TODO(),
 &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(clientId string, userName string, password
 string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(context.TODO(),
 &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(clientId string, userName string, password
 string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(context.TODO(),
 &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(clientId string, userName string)
 (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(context.TODO(),
 &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(clientId string, code string,
 userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(context.TODO(),
 &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(userAccessToken string) error {
 _, err := actor.CognitoClient.DeleteUser(context.TODO(),
 &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(userPoolId string, userName string,
 userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(context.TODO(),
 &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(userPoolId string, userName
 string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(context.TODO(),
 &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 racchiuda le azioni di DynamoDB.

```
// 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(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(context.TODO(),
 &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(tableName string) (UserList, error) {
```

```
 var userList UserList
 output, err := actor.DynamoClient.Scan(context.TODO(), &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(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(context.TODO(), &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 Logs. CloudWatch

```
type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
```

```
func (actor CloudWatchLogsActions) GetLatestLogStream(functionName string)
 (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(context.TODO(),
 &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(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(context.TODO(),
 &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. AWS 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(stackName string) StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(context.TODO(),
 &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
}
```

Pulisci le risorse.

```
// 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() {
 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(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(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 delle API AWS SDK for Go .
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPiscina

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. 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 SDK AWS

Il seguente esempio di codice mostra come migrare automaticamente utenti noti di Amazon Cognito con una funzione Lambda.

- Configura un pool di utenti per chiamare una funzione Lambda per il `MigrateUser` trigger.
- Accedi ad Amazon Cognito con un nome utente e un indirizzo e-mail non inclusi nel pool di utenti.
- La funzione Lambda analizza una tabella DynamoDB e migra automaticamente gli utenti noti nel pool di utenti.
- Esegui il flusso relativo alla password dimenticata per reimpostare la password per l'utente migrato.
- Accedi come nuovo utente, quindi ripulisci 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 (
 "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(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(
  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(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(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(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(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(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(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(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(stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup()
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]

 runner.AddMigrateUserTrigger(stackOutputs["UserPoolId"],
 stackOutputs["MigrateUserFunctionArn"])
 runner.resources.triggers = append(runner.resources.triggers,
 actions.UserMigration)
 resetNeeded, user := runner.SignInUser(stackOutputs["TableName"],
 stackOutputs["UserPoolClientId"])
 if resetNeeded {
  runner.helper.ListRecentLogEvents(stackOutputs["MigrateUserFunction"])
  runner.ResetPassword(stackOutputs["UserPoolClientId"], user)
 }

 runner.resources.Cleanup()

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Gestisci il `MigrateUser` grilletto con una funzione Lambda.

```
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 output.Items == nil || 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() {
 sdkConfig, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Crea una struttura che esegua attività comuni.

```
// IScenarioHelper defines common functions used by the workflows in this
 example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(stackName string) (actions.StackOutputs, error)
 PopulateUserTable(tableName string)
 GetKnownUsers(tableName string) (actions.UserList, error)
 AddKnownUser(tableName string, user actions.User)
 ListRecentLogEvents(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(stackName string)
 (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(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(tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured
 format.
func (helper ScenarioHelper) GetKnownUsers(tableName string) (actions.UserList,
 error) {
 knownUsers, err := helper.dynamoActor.Scan(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(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(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(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(functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(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 racchiuda le azioni di Amazon Cognito.

```
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(userPoolId string,
 triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(context.TODO(),
 &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(context.TODO(),
 &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(clientId string, userName string, password
 string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(context.TODO(),
 &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(clientId string, userName string, password
 string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(context.TODO(),
 &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(clientId string, userName string)
 (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(context.TODO(),
 &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(clientId string, code string,
 userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(context.TODO(),
 &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(userAccessToken string) error {
 _, err := actor.CognitoClient.DeleteUser(context.TODO(),
 &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(userPoolId string, userName string,
 userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(context.TODO(),
 &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(userPoolId string, userName
 string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(context.TODO(),
 &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 racchiuda le azioni di DynamoDB.

```
// 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(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(context.TODO(),
 &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(tableName string) (UserList, error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(context.TODO(), &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(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(context.TODO(), &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 Logs. CloudWatch

```
type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(functionName string)
 (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(context.TODO(),
 &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(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(context.TODO(),
 &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. AWS 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(stackName string) StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(context.TODO(),
 &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
}
```

Pulisci le risorse.

```
// 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() {
 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(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(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 delle API AWS SDK for Go .
  - ConfirmForgotPassword
  - DeleteUser
  - ForgotPassword
  - InitiateAuth
  - SignUp
  - UpdateUserPiscina

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Inizia a creare e richiamare funzioni Lambda utilizzando un SDK AWS

Gli esempi di codice seguenti mostrano come:

- Crea un ruolo IAM e una funzione Lambda, quindi carica il codice del gestore.
- Richiamare la funzione con un singolo parametro e ottenere i risultati.
- Aggiorna il codice della funzione e configuralo con una variabile di ambiente.
- Richiamare 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 sull'utilizzo di Lambda, consulta Creare una funzione Lambda con la console.

.NET

AWS SDK for .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)
                    .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.Write(".");
        }
        while (config.State != State.Active);

        Console.WriteLine($"\nThe function, {functionName} has been created.");
        Console.WriteLine($"The 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.Write("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.Write(".");
        }
        while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

        await lambdaWrapper.UpdateFunctionConfigurationAsync(
            functionName,
            calculatorHandler,
            new Dictionary<string, string> { { "LOG_LEVEL", "DEBUG" } });

        do
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
        }
        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("\tOr 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.DefaultLambdaJsonSeri

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.DefaultLambdaJsonSeri

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 delle API AWS SDK for .NET .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCodice
  - UpdateFunctionConfigurazione

C++

SDK per C++

> **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 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.
#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_8);
#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 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::JsonView> 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;

#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
            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;
            }
        }
        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 delle API AWS SDK for C++ .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCodice
  - UpdateFunctionConfigurazione

Go

SDK per Go V2

> **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.

Creare uno scenario interattivo che ti mostri come iniziare a usare le funzioni Lambda.

```
// 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() {
 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()
 funcName := scenario.CreateFunction(role)
 scenario.InvokeIncrement(funcName)
 scenario.UpdateFunction(funcName)
 scenario.InvokeCalculator(funcName)
 scenario.ListFunctions()
 scenario.Cleanup(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() *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(context.TODO(), &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(context.TODO(), &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(context.TODO(), &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(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(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(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(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(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(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(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(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(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() {
 count := scenario.questioner.AskInt(
  "Let's list functions for your account. How many do you want to see?",
 demotools.NotEmpty{})
 functions := scenario.functionWrapper.ListFunctions(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(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(context.TODO(),
   &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(context.TODO(),
 &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(context.TODO(), &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(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.")
 }
}
```

Creare una struttura che racchiude le singole operazioni Lambda.

```
// 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(functionName string) types.State {
 var state types.State
 funcOutput, err := wrapper.LambdaClient.GetFunction(context.TODO(),
 &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(functionName string, handlerName
 string,
 iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
 var state types.State
```

```
 _, err := wrapper.LambdaClient.CreateFunction(context.TODO(),
 &lambda.CreateFunctionInput{
  Code:         &types.FunctionCode{ZipFile: zipPackage.Bytes()},
  FunctionName: aws.String(functionName),
  Role:         iamRoleArn,
  Handler:      aws.String(handlerName),
  Publish:      true,
  Runtime:      types.RuntimePython38,
 })
 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(context.TODO(),
 &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(functionName string, zipPackage
 *bytes.Buffer) types.State {
 var state types.State
```

```
 _, err := wrapper.LambdaClient.UpdateFunctionCode(context.TODO(),
 &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(context.TODO(),
 &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(functionName string,
 envVars map[string]string) {
 _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(context.TODO(),
 &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(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(context.TODO())
  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(functionName string) {
 _, err := wrapper.LambdaClient.DeleteFunction(context.TODO(),
 &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(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(context.TODO(),
 &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"`
}
```

Creare una struttura che implementa funzioni per facilitare l'esecuzione dello scenario.

```
// 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
}
```

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 delle API AWS SDK for Go .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCodice
  - UpdateFunctionConfigurazione

Java

SDK per Java 2.x

> **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.

```
/*
 *  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> <filePath> <role> <handler> <bucketName> <key>
\s

                Where:
                    functionName - The name of the Lambda function.\s
                    filePath - The path to the .zip or .jar where the code is
 located.\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 != 6) {
           System.out.println(usage);
           System.exit(1);
       }

       String functionName = args[0];
       String filePath = args[1];
       String role = args[2];
       String handler = args[3];
       String bucketName = args[4];
       String key = args[5];

       Region region = Region.US_WEST_2;
       LambdaClient awsLambda = LambdaClient.builder()
               .region(region)
               .build();

       System.out.println(DASHES);
       System.out.println("Welcome to the AWS Lambda example scenario.");
       System.out.println(DASHES);

       System.out.println(DASHES);
       System.out.println("1. Create an AWS Lambda function.");
       String funArn = createLambdaFunction(awsLambda, functionName, filePath,
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();
    }

    public static String createLambdaFunction(LambdaClient awsLambda,
            String functionName,
            String filePath,
            String role,
            String handler) {

        try {
            LambdaWaiter waiter = awsLambda.waiter();
```

```
            InputStream is = new FileInputStream(filePath);
            SdkBytes fileToUpload = SdkBytes.fromInputStream(is);

            FunctionCode code = FunctionCode.builder()
                    .zipFile(fileToUpload)
                    .build();

            CreateFunctionRequest functionRequest =
CreateFunctionRequest.builder()
                    .functionName(functionName)
                    .description("Created by the Lambda Java API")
                    .code(code)
                    .handler(handler)
                    .runtime(Runtime.JAVA8)
                    .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 | FileNotFoundException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    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);
        }
    }

    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);
        }
    }

    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);
        }
    }

    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);
        }
    }

    public static void updateFunctionConfiguration(LambdaClient awsLambda, String
functionName, String handler) {
        try {
            UpdateFunctionConfigurationRequest configurationRequest =
UpdateFunctionConfigurationRequest.builder()
                    .functionName(functionName)
```

```
                    .handler(handler)
                    .runtime(Runtime.JAVA11)
                    .build();

            awsLambda.updateFunctionConfiguration(configurationRequest);

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    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 delle API AWS SDK for Java 2.x .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCodice
  - UpdateFunctionConfigurazione

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.

```
    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);
};
```

Richiamare 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 = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

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 delle API AWS SDK for JavaScript .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCodice
  - UpdateFunctionConfigurazione

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.Java8
        }

    // Create a Lambda function using a waiter
    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val functionResponse = awsLambda.createFunction(request)
        awsLambda.waitUntilFunctionActive {
            functionName = myFunctionName
        }
        return functionResponse.functionArn.toString()
    }
}

suspend fun getFunction(functionNameVal: String) {
```

```
    val functionRequest =
        GetFunctionRequest {
            functionName = functionNameVal
        }

    LambdaClient { region = "us-west-2" }.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-west-2" }.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-west-2" }.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-west-2" }.use { awsLambda ->
        val response = awsLambda.updateFunctionCode(functionCodeRequest)
        awsLambda.waitUntilFunctionUpdated {
            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.Java11
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        awsLambda.updateFunctionConfiguration(configurationRequest)
    }
}

suspend fun delFunction(myFunctionName: String) {
    val request =
        DeleteFunctionRequest {
            functionName = myFunctionName
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
```

```
        awsLambda.deleteFunction(request)
        println("$myFunctionName was deleted")
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Kotlin.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCodice
  - UpdateFunctionConfigurazione

PHP

SDK per PHP

> **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.

```
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";

        echo "\nNow let's create an S3 bucket and upload our Lambda code there.
\n";
        $bucketName = "test-example-bucket-$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 delle API AWS SDK for PHP .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCodice
  - UpdateFunctionConfigurazione

Python

SDK per Python (Boto3)

> **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.

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="python3.8",
                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']}
{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 delle API SDK AWS per Python (Boto3).
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCodice

- UpdateFunctionConfigurazione

Ruby

SDK per Ruby

> **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 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)
    role_policy = {
      'Version': "2012-10-17",
      'Statement': [
        {
          'Effect': "Allow",
          'Principal': {
            'Service': "lambda.amazonaws.com"
          },
          'Action': "sts:AssumeRole"
        }
      ]
    }
    case action
    when "create"
      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
        }
      )
      $iam_client.wait_until(:role_exists, { role_name: iam_role_name }) do |w|
        w.max_attempts = 5
        w.delay = 5
      end
      @logger.debug("Successfully created IAM role: #{role['role']['arn']}")
      @logger.debug("Enforcing a 10-second sleep to allow IAM role to activate
 fully.")
      sleep(10)
      return role, role_policy.to_json
    when "destroy"
      $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}")
    else
      raise "Incorrect action provided. Must provide 'create' or 'destroy'"
    end
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error creating role or attaching policy:\n
 #{e.message}")
  end
```

Definisci un gestore Lambda che incrementa un numero fornito come parametro di chiamata.

```
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
```

Comprimi la funzione Lambda in un pacchetto di implementazione:

```
  # Creates a Lambda deployment package in .zip format.
  # This zip can be passed directly as a string to Lambda when creating the
 function.
  #
  # @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) {
      |zipfile|
      zipfile.add("lambda_function.rb", "#{source_file}.rb")
    }
    @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. This
 #                      must include the file name and the function name.
 # @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
```

Richiama 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
```

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
```

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 executing #{function_name}:\n
#{e.message}")
  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 delle API AWS SDK for Ruby .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCodice
  - UpdateFunctionConfigurazione

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.

Il file Cargo.toml con dipendenze utilizzato 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 src/ations.rs nella cassa.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

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::{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 ToString for Operation {
    fn to_string(&self) -> String {
        match self {
            Operation::Plus => "plus".to_string(),
            Operation::Minus => "minus".to_string(),
            Operation::Times => "times".to_string(),
            Operation::DividedBy => "divided-by".to_string(),
        }
    }
}

/**
 * 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),
        )
    }

    // snippet-start:[lambda.rust.scenario.prepare_function]
    /**
     * 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())
    }
    // snippet-end:[lambda.rust.scenario.prepare_function]

    // snippet-start:[lambda.rust.scenario.create_function]
    /**
     * 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)
    }
    // snippet-end:[lambda.rust.scenario.create_function]

    /**
     * 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)
    }

    // snippet-start:[lambda.rust.scenario.get_function]
    /** 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)
    }
    // snippet-end:[lambda.rust.scenario.get_function]

    // snippet-start:[lambda.rust.scenario.list_functions]
    /** 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)
    }
    // snippet-end:[lambda.rust.scenario.list_functions]

    // snippet-start:[lambda.rust.scenario.invoke]
    /** 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)
    }
    // snippet-end:[lambda.rust.scenario.invoke]

    // snippet-start:[lambda.rust.scenario.update_function_code]
    /** 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)
    }
    // snippet-end:[lambda.rust.scenario.update_function_code]

    // snippet-start:[lambda.rust.scenario.update_function_configuration]
    /** 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)
    }
    // snippet-end:[lambda.rust.scenario.update_function_configuration]

    // snippet-start:[lambda.rust.scenario.delete_function]
    /** 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)
    }
    // snippet-end:[lambda.rust.scenario.delete_function]

    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.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0

/*
## 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"))
}

// snippet-start:[lambda.rust.scenario.log_invoke_output]
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")
    }
}
// snippet-end:[lambda.rust.scenario.log_invoke_output]

async fn main_block(
    opt: &Opt,
    manager: &LambdaManager,
    code_location: String,
) -> Result<(), anyhow::Error> {
    let invoke = manager.invoke(Increment(opt.inc)).await?;
    log_invoke_output(&invoke, "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 sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per Rust.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCodice
  - UpdateFunctionConfigurazione

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_rolename = iv_role_name
                   iv_assumerolepolicydocument = lv_policy_document
                   iv_description = 'Grant lambda permission to write to logs'
               ).
           MESSAGE 'IAM role created.' TYPE 'I'.
           WAIT UP TO 10 SECONDS.           " Make sure that the IAM role is
ready for use. "
         CATCH /aws1/cx_iamentityalrdyexex.
           MESSAGE 'IAM role already exists.' TYPE 'E'.
         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_rolename  = 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 = lo_create_role_output->get_role( )->get_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_lmdinvrequestcontex.
            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_lmdinvrequestcontex.
            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 'E'.
        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 '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.

        " 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 'E'.
          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 sulle API, consulta i seguenti argomenti nella Documentazione di riferimento delle API SDK AWS per SAP ABAP.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCodice
  - UpdateFunctionConfigurazione

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. 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 tramite un SDK AWS

Il seguente esempio di codice mostra come scrivere dati di attività personalizzati 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 `PostAuthentication` trigger.
- Accedi il nuovo utente ad Amazon Cognito.
- La funzione Lambda scrive informazioni personalizzate nei CloudWatch log e in una tabella DynamoDB.
- Ottieni e visualizza dati personalizzati dalla tabella DynamoDB, quindi ripulisci 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.

```
// 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(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(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(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(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(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(
  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(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(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(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(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(stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup()
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]
 runner.helper.PopulateUserTable(stackOutputs["TableName"])
 userName, password := runner.AddUserToPool(stackOutputs["UserPoolId"],
 stackOutputs["TableName"])

 runner.AddActivityLogTrigger(stackOutputs["UserPoolId"],
 stackOutputs["ActivityLogFunctionArn"])
 runner.SignInUser(stackOutputs["UserPoolClientId"], userName, password)
 runner.helper.ListRecentLogEvents(stackOutputs["ActivityLogFunction"])
 runner.GetKnownUserLastLogin(stackOutputs["TableName"], userName)

 runner.resources.Cleanup()

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
```

```
 log.Println(strings.Repeat("-", 88))
}
```

Gestisci il `PostAuthentication` grilletto con una funzione Lambda.

```
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() {
 sdkConfig, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  log.Panicln(err)
 }
```

```
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Crea una struttura che esegua attività comuni.

```
// IScenarioHelper defines common functions used by the workflows in this
 example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(stackName string) (actions.StackOutputs, error)
 PopulateUserTable(tableName string)
 GetKnownUsers(tableName string) (actions.UserList, error)
 AddKnownUser(tableName string, user actions.User)
 ListRecentLogEvents(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(stackName string)
 (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(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(tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured
 format.
func (helper ScenarioHelper) GetKnownUsers(tableName string) (actions.UserList,
 error) {
 knownUsers, err := helper.dynamoActor.Scan(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(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(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(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(functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(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 racchiuda le azioni di Amazon Cognito.

```
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(userPoolId string,
 triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(context.TODO(),
 &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(context.TODO(),
 &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(clientId string, userName string, password
 string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(context.TODO(),
 &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(clientId string, userName string, password
 string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(context.TODO(),
 &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(clientId string, userName string)
 (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(context.TODO(),
 &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(clientId string, code string,
 userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(context.TODO(),
 &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(userAccessToken string) error {
 _, err := actor.CognitoClient.DeleteUser(context.TODO(),
 &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(userPoolId string, userName string,
 userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(context.TODO(),
 &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(userPoolId string, userName
 string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(context.TODO(),
 &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 racchiuda le azioni di DynamoDB.

```
// 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(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(context.TODO(),
 &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(tableName string) (UserList, error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(context.TODO(), &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(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(context.TODO(), &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 Logs. CloudWatch

```
type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(functionName string)
 (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(context.TODO(),
 &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(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(context.TODO(),
 &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. AWS 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(stackName string) StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(context.TODO(),
 &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
}
```

Pulisci le risorse.

```
// 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() {
 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(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(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 delle API AWS SDK for Go .
  - AdminCreateUtente
  - AdminSetUserPassword
  - DeleteUser
  - InitiateAuth
  - UpdateUserPiscina

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esempi serverless per AWS Lambda con SDK

I seguenti esempi di codice mostrano come usare Lambda con AWS gli SDK.

Esempi

- Connessione a un database Amazon RDS in una funzione Lambda
- Richiamare una funzione Lambda da un trigger Kinesis
- Richiama una funzione Lambda da un trigger DynamoDB
- Richiama una funzione Lambda da un trigger di Amazon DocumentDB
- Richiamo di una funzione Lambda da un trigger Amazon S3
- Richiamo 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 degli errori degli elementi batch per le 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

I seguenti esempi di codice 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.

Go

SDK per Go V2

> **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 Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Golang v2 code here.
*/

package main

import (
 "context"
 "database/sql"
 "encoding/json"
 "fmt"

 "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 = "DatabaseName"
 var dbUser string = "DatabaseUser"
 var dbHost string = "mysqldb.123456789012.us-east-1.rds.amazonaws.com"
 var dbPort int = 3306
 var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
 var region string = "us-east-1"

 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)
}
```

JavaScript

SDK per (v2 JavaScript)

> **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 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)
  }
};
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Richiamare 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

AWS SDK for .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.DefaultLambdaJsonSeri

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.

Consumo di un evento Kinesis con Lambda utilizzando 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 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 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.

Consumo di un evento Kinesis con Lambda utilizzando 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.

Consumo di un evento Kinesis con Lambda utilizzando 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 SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Richiama una funzione Lambda da un trigger DynamoDB

I seguenti esempi di codice mostrano 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

AWS SDK for .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.

Consumo di un evento DynamoDB con Lambda utilizzando.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.DefaultLambdaJsonSeri

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 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 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 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 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 (v3) JavaScript

> **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

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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.

Consumo di un evento DynamoDB con Lambda utilizzando 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\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 di più su. 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 Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

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 di più su. 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 Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

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 di più su. 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 Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

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 SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Richiama una funzione Lambda da un trigger di Amazon DocumentDB

I seguenti esempi di codice mostrano 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.

Go

SDK per Go V2

> **Note**
>
> C'è altro su. 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 Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

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))
}
```

JavaScript

SDK per (v3 JavaScript)

> **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));
};
```

Python

SDK per Python (Boto3)

> **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 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 di più su. 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 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
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Richiamo 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

AWS SDK for .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.DefaultLambdaJsonSeri

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.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

exports.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.

Consumo di un evento S3 con Lambda utilizzando 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\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 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 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.

Consumo di un evento S3 con Lambda utilizzando 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 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, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Richiamo 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

AWS SDK for .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.DefaultLambdaJsonSeri

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.

Consumo di un evento SNS con Lambda utilizzando 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 (v3 JavaScript)

> **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.

Consumo di un evento SNS con Lambda utilizzando 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 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
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. 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

AWS SDK for .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.DefaultLambdaJsonSeri

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.

Consumo di un evento SQS con Lambda utilizzando 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 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 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
```

```
  rescue StandardError => err
    puts "An error occurred"
    raise err
  end
end
```

Rust

SDK per Rust

> **Note**
>
> C'è dell'altro 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 Lambda utilizzando 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 SDK AWS 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

AWS SDK for .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.DefaultLambdaJsonSeri
```

```
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 degli errori degli elementi batch Kinesis con Lambda utilizzando 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 altro da sapere. 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 degli errori degli 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 altro da sapere. 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 degli errori degli elementi batch Kinesis con Lambda utilizzando 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 sapere. 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 Kinesis con Lambda utilizzando 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 SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Segnalazione degli errori degli elementi batch per le funzioni Lambda con un trigger DynamoDB

I seguenti esempi di codice mostrano come implementare una risposta batch parziale per le 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

AWS SDK for .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 degli errori degli elementi batch di DynamoDB con Lambda utilizzando.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.DefaultLambdaJsonSeri

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 altro su. 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 con Lambda utilizzando 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 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 con Lambda utilizzando 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.io.Serializable;
import java.util.ArrayList;
import java.util.List;
```

```
public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 Serializable> {

    @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 (v3) JavaScript

> **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

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBBatchItemFailure, DynamoDBStreamEvent } from "aws-lambda";

export const handler = async (event: DynamoDBStreamEvent):
 Promise<DynamoDBBatchItemFailure[]> => {
```

```
    const batchItemsFailures: DynamoDBBatchItemFailure[] = []
    let curRecordSequenceNumber

    for(const record of event.Records) {
        curRecordSequenceNumber = record.dynamodb?.SequenceNumber

        if(curRecordSequenceNumber) {
            batchItemsFailures.push({
                itemIdentifier: curRecordSequenceNumber
            })
        }
    }

    return batchItemsFailures
}
```

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 degli errori degli elementi batch di DynamoDB con Lambda utilizzando 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\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 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 con Lambda utilizzando 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 altro su. 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 con Lambda utilizzando Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
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 sapere. 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 con Lambda utilizzando Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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 SDK AWS 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

AWS SDK for .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.DefaultLambdaJsonSeri
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 degli errori degli elementi batch SQS con Lambda utilizzando Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main
```

```
import (
 "context"
 "encoding/json"
 "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 /* Your message processing condition here */ {
   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 altro da sapere. 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>();
        String messageId = "";
        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
            try {
                //process your message
                messageId = message.getMessageId();
            } catch (Exception e) {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.add(new
 SQSBatchResponse.BatchItemFailure(messageId));
            }
        }
        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 utilizzando Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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 utilizzando Lambda. 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 degli errori degli elementi batch 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 altro da sapere. 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:
                # process message
            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
```

Rust

SDK per Rust

> **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 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, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esempi interservizi per AWS Lambda che utilizzano gli SDK

Le seguenti applicazioni di esempio utilizzano AWS gli SDK per combinare Lambda con altri. Servizi AWS Ogni esempio include un collegamento a GitHub, dove è possibile trovare istruzioni su come configurare ed eseguire l'applicazione.

Esempi

- Creazione di una REST API di API Gateway per monitorare i dati COVID-19
- Creazione di una REST API per la libreria di prestiti
- Creazione di un'applicazione di messaggistica con Step Functions
- 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
- Richiamo a una funzione Lambda da un browser
- Trasforma i dati per la tua applicazione con S3 Object Lambda

- Utilizzo di un'API Gateway per richiamare una funzione Lambda
- Utilizzo di Step Functions per richiamare le funzioni Lambda
- Utilizzo degli eventi pianificati per richiamare una funzione Lambda

Creazione di una REST API di API Gateway per monitorare i dati COVID-19

Il seguente esempio di codice mostra come creare una REST API che simula un sistema per monitorare i casi quotidiani di COVID-19 negli Stati Uniti, utilizzando dati fittizi.

Python

SDK per Python (Boto3)

Mostra come usare AWS Chalice con per AWS SDK for Python (Boto3) creare un'API REST serverless che utilizzi Amazon API Gateway e Amazon DynamoDB. AWS Lambda La REST API simula un sistema che monitora i casi giornalieri di COVID-19 negli Stati Uniti, 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

- API Gateway
- AWS CloudFormation
- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. 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 for 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.
- Utilizzalo 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

- API Gateway
- Aurora
- Lambda
- Secrets Manager

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. 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 for 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 scansionare il 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, consultaUsare Lambda con un SDK AWS. 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

Nell'esempio di codice seguente viene illustrato come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

.NET

AWS SDK for .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

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- 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

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- 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

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- 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

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- 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

- API Gateway
- DynamoDB

- Lambda
- Amazon Rekognition
- 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

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- 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

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. 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 for 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 API Gateway
- 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

- API Gateway

- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. 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

AWS SDK for .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 for 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, consultaUsare Lambda con un SDK AWS. 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 for 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, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Trasforma i dati per la tua applicazione con S3 Object Lambda

Il seguente esempio di codice mostra come trasformare i dati per la tua applicazione con S3 Object Lambda.

.NET

AWS SDK for .NET

Mostra come aggiungere codice personalizzato alle richieste S3 GET standard per modificare l'oggetto richiesto recuperato da S3 in modo che l'oggetto soddisfi le esigenze del client o dell'applicazione richiedente.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Lambda
- Amazon S3

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo di un'API Gateway per richiamare 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 richiamata 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

- API Gateway
- 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 richiamata 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 for JavaScript v3 .

Servizi utilizzati in questo esempio

- API Gateway

- 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 percorsi 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 API Gateway.
- Creare una risorsa REST destinata alla funzione Lambda.
- Concedere l'autorizzazione affinché l'API Gateway richiami 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

- API Gateway
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo di Step Functions per richiamare le funzioni Lambda

I seguenti esempi di codice mostrano 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

JavaScript

SDK per JavaScript (v3)

Mostra come creare un flusso di lavoro AWS serverless utilizzando AWS Step Functions and. AWS SDK for JavaScript Ogni fase del flusso di lavoro viene implementata utilizzando una AWS Lambda funzione.

Lambda è un servizio di calcolo che consente di eseguire il codice senza effettuare il provisioning o la gestione di server. Step Functions è un servizio di orchestrazione serverless che consente di combinare funzioni Lambda e altri servizi AWS per la creazione di applicazioni business-critical.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK for JavaScript v3 .

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, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo degli eventi pianificati per richiamare 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

- 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 for JavaScript v3 .

Servizi utilizzati in questo esempio

- 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
- EventBridge
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Quote di Lambda

> **Important**
>
> Account AWS I nuovi hanno quote di concorrenza e memoria ridotte. AWS aumenta automaticamente queste quote in base all'utilizzo.

Calcolo 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
Storage per funzioni caricate (archivi di file .zip) e livelli. Ogni versione di funzione e di livello consuma spazio di storage. Per le best practice da seguire per la gestione dell'archiviazione del codice, consulta Monitoraggio dell'archiviazione del codice Lambda in Serverless Land.	75 GB	Terabyte
Storage per le funzioni definite come immagini di container. Queste immagini sono memorizza te in Amazon ECR.	Consulta Service Quotas di Amazon ECR.	

Risorsa	Quota predefinita	Può essere aumentato fino a
Interfacce di rete elastiche per Virtual Private Cloud (VPC) **Note** Questa quota è condivisa con altri servizi, ad esempio, Amazon Elastic File System (Amazon EFS). Consulta Quote Amazon VPC.	250	Migliaia

Per ulteriori informazioni sulla simultaneità e su come Lambda ridimensiona la simultaneità della funzione in risposta al traffico, consulta Comprendere il ridimensionamento delle funzioni 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.

Note

La documentazione Lambda, i messaggi di log e la console utilizzano l'abbreviazione MB (anziché MiB) per fare riferimento a 1.024 KB.

Risorsa	Quota
Allocazione di memoria della funzione	Da 128 MB a 10.240 MB, in incrementi di 1 MB. Nota: Lambda alloca la potenza della CPU in proporzione alla quantità di memoria configurata. È possibile

Risorsa	Quota
	aumentare o diminuire la memoria e la potenza della CPU assegnate alla funzione utilizzando l'impostazione Memory (MB). A 1.769 MB, una funzione ha l'equivalente di una vCPU.
Timeout della funzione.	900 secondi (15 minuti)
Variabili di ambiente della funzione	4 KB, per tutte le variabili di ambiente associate alla funzione, in forma aggregata
Policy basata sulle risorse della funzione	20 KB
Livelli della funzione	cinque livelli
Limite di dimensionamento della simultaneità delle funzioni	Per ogni funzione, 1.000 ambienti di esecuzione ogni 10 secondi
Payload dell'invocazione (richiesta e risposta)	6 MB ciascuno per richiesta e risposta (sincrono) 20 MB per ogni risposta in streaming (sincrona). La dimensione del payload per le risposte in streaming può essere aumentata rispetto ai valori predefiniti. Contatta AWS Support per ulteriori informazioni.) 256 KB (asincrono) 1 MB per la dimensione totale combinata dei valori della riga di richiesta e dell'intestazione

Risorsa	Quota
Larghezza di banda per le risposte in streaming	Senza limite per i primi 6 MB di risposta della funzione Per risposte superiori a 6 MB, 2 MB/s per il resto della risposta
Dimensioni del pacchetto di implementazione (archivio di file .zip)	50 MB (compresso, per il caricamento diretto) 250 MB (decompresso) Questa quota si applica a tutti i file caricati, inclusi i livelli e i tempi di esecuzione personalizzati. 3 MB (editor della console)
Impostazioni dell'immagine di container	16 KB
Dimensione del pacchetto del codice dell'immagine di container	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
Descrittori di file	1,024
Processi/thread dell'esecuzione	1,024

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 richieste al secondo. In altre parole, il limite totale di chiamate è 10 volte il limite di simultaneità. Per informazioni, consulta Comprendere il ridimensionamento delle funzioni Lambda.
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. Per informazioni, consulta Comprendere il ridimensionamento delle funzioni Lambda.
Richieste di invocazione per versione di funzione o alias (richieste al secondo)	10 x simultaneità fornita allocata **Note** Questa quota si applica solo alle funzioni che utilizzano la simultaneità con provisioning.
Richieste API GetFunction	100 richieste al secondo. Non può essere aumentato.
Richieste API GetPolicy	15 richieste al secondo. Non può essere aumentato.
Resto delle richieste API del piano di controllo (escluse le chiamate e le richieste) GetFunction GetPolicy	15 richieste al secondo su tutte le API (non 15 richieste al secondo per API). Non può essere aumentato.

Altri servizi

Le quote per altri servizi, come AWS Identity and Access Management (IAM), Amazon CloudFront (Lambda @Edge) e Amazon Virtual Private Cloud (Amazon VPC), possono influire sulle 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 di altri servizi AWS.

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
Support SnapStart per nuove regioni	Lambda SnapStartè ora disponibile nelle seguenti regioni: Europa (Spagna), Europa (Zurigo), Asia Pacifico (Melbourne), Asia Pacifico (Hyderabad) e Medio Oriente (Emirati Arabi Uniti).	12 gennaio 2024
AWS aggiornamenti delle politiche gestite	Service Quotas ha aggiornato una politica AWS gestita esistente ()`AWSLambdaVPCAccessExecutionRole` .	5 gennaio 2024
Runtime di Python 3.12	Ora Lambda supporta Python 3.12 come runtime gestito e immagine di base del container. Per ulteriori informazioni, consulta Python 3.12 runtime ora disponibile nel blog di AWS Lambda Compute AWS .	14 dicembre 2023
Runtime Java 21	Lambda ora supporta Java 21 come runtime gestito e immagine di base del container (`java21`).	16 novembre 2023

Runtime di Node.js 20.x	Lambda ora supporta Node.js 20 come runtime gestito e immagine di base del container (`nodejs20.x`). Per ulteriori informazioni, consulta il runtime di Node.js 20.x ora disponibile nel AWS Lambda Compute Blog. AWS	14 novembre 2023
Runtime provided.al2023	Lambda ora supporta Amazon Linux 2023 come runtime gestito e immagine di base del container. Per ulteriori informazioni, consulta la sezione Introduzione al runtime di Amazon Linux 2023 AWS Lambda sul blog di AWS Compute.	9 novembre 2023
Supporto IPv6 per sottoreti dual-stack	Lambda ora supporta il traffico IPv6 in uscita verso le sottoreti dual-stack. Per ulteriori informazioni, consultare Supporto IPv6.	12 ottobre 2023
Test di funzioni e applicazioni serverless	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 Test di funzioni e applicazioni serverless.	16 giugno 2023

Runtime Ruby 3.2	Lambda ora supporta un nuovo runtime per Ruby 3.2. Per ulteriori informazioni, consulta la sezione Compilazione di funzioni Lambda con Ruby.	7 giugno 2023
Streaming delle risposte	Lambda ora supporta lo streaming delle risposte dalle funzioni. Per ulteriori informazioni, consulta la pagina Configurazione di una funzione Lambda per lo streaming delle risposte.	6 aprile 2023
Parametri di chiamata asincrona	Lambda rilascia parametri di chiamata asincrona. Per ulteriori informazioni, consulta Parametri di chiamata asincrona.	9 febbraio 2023
Controlli della versione di runtime	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 Aggiornamenti di runtime Lambda.	23 gennaio 2023

Lambda SnapStart	Usa Lambda SnapStart per ridurre i tempi di avvio delle funzioni Java senza fornire risorse aggiuntive o implement are complesse ottimizza zioni delle prestazioni. Per ulteriori informazioni, consulta Migliorare le prestazioni di avvio con SnapStart Lambda.	28 novembre 2022
Runtime di Node.js 18	Lambda ora supporta un nuovo runtime per Node.js 18. Node.js 18 utilizza Amazon Linux 2. Per ulteriori dettagli, consultare Compilazione di funzioni Lambda con Node.js.	18 novembre 2022
lambda: SourceFunctionArn chiave di condizione	Per una AWS risorsa, la chiave `lambda:SourceFunctionArn` condition filtra l'accesso alla risorsa tramite l'ARN di una funzione Lambda. Per maggiori dettagli, consulta Utilizzo delle credenziali dell'ambiente di esecuzione Lambda.	1 luglio 2022
Runtime di Node.js 16	Lambda ora supporta un nuovo runtime per Node.js 16. Node.js 16 utilizza Amazon Linux 2. Per ulteriori dettagli, consultare Compilazione di funzioni Lambda con Node.js.	11 maggio 2022

URL della funzione Lambda	Lambda ora supporta gli URL della funzione, che sono endpoint HTTP(S) dedicati per le funzioni Lambda. Per informazioni dettagliate, consulta URL della funzione Lambda.	6 aprile 2022
Eventi di test condivisi nella console AWS Lambda	Lambda ora supporta la condivisione di eventi di test con altri utenti nello stesso Account AWS. Per informazioni dettagliate, consulta Test delle funzioni Lambda nella console.	16 marzo 2022
PrincipalOrgId nelle politiche basate sulle risorse	Lambda ora supporta la concessione delle autorizzazioni a un'organizzazione in AWS Organizations. Per informazioni dettagliate, consulta Utilizzo delle policy basate su risorse per AWS Lambda.	11 marzo 2022
Runtime .NET 6	Lambda ora supporta un nuovo runtime per .NET 6. Per informazioni dettagliate, consultare Runtime di Lambda.	23 febbraio 2022
Filtraggio di eventi per le origini di eventi Kinesis, DynamoDB e Amazon SQS	Lambda ora supporta il filtraggio di eventi per le origini di eventi Kinesis, DynamoDB e Amazon SQS. Per informazioni dettagliate, consulta Filtro eventi Lambda.	24 novembre 2021

Autenticazione mTLS per Amazon MSK e origine eventi Apache Kafka autogestito	Lambda ora supporta l'autenticazione mTLS per Amazon MSK e le origini eventi Apache Kafka autogestito. Per informazioni dettagliate, consultare Uso di Lambda con Amazon MSK.	19 novembre 2021
Lambda su Graviton2	Lambda ora supporta Graviton2 per le funzioni che utilizzano l'architettura arm64. Per informazioni dettagliate, consulta Architetture del set di istruzioni Lambda.	29 settembre 2021
Runtime di Python 3.9	Lambda ora supporta un nuovo runtime per Python 3.9. Per informazioni dettagliate, consultare Runtime di Lambda.	16 agosto 2021
Nuove versioni di runtime per Node.js, Python e Java	Sono disponibili nuove versioni del runtime per Node.js, Python e Java. Per informazioni dettagliate, consultare Runtime di Lambda.	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
Autenticazione SASL/PLAIN per Kafka autogestito su Lambda	SASL/PLAIN è ora un meccanismo di autentica zione supportato per le origini eventi Kafka autogestite sui. I clienti Lambda che già utilizzano SASL/PLAIN sul proprio cluster Kafka autogesti to possono ora utilizzare facilmente Lambda per creare applicazioni consumer senza dover modificare il modo in cui si autenticano. Per informazi oni dettagliate, consultare Uso di Lambda con Apache Kafka autogestito.	29 giugno 2021

API estensioni Lambda	Disponibilità generale per le estensioni Lambda. Utilizza le estensioni per potenziar e le funzioni Lambda. È possibile utilizzare le estension i fornite dai Lambda Partners oppure creare le tue estension i Lambda. Per informazioni dettagliate, consultare API Extensions di Lambda.	24 maggio 2021
Nuova esperienza della console Lambda	La console Lambda è stata riprogettata per migliorare le prestazioni e la coerenza.	2 marzo 2021
Runtime di Node.js 14	Lambda ora supporta un nuovo runtime per Node.js 14. Node.js 14 utilizza Amazon Linux 2. Per ulteriori dettagli, consultare Compilazione di funzioni Lambda con Node.js.	27 gennaio 2021
Immagini di container Lambda	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 applicazi oni. Per ulteriori dettagli, consultare Uso di immagini di container con Lambda.	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 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 è possibile utilizzare le chiavi di condizione specifiche di Lambda per le impostazioni 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 consumer del flusso Kinesis HTTP/2	Ora puoi utilizzare le seguenti impostazioni di concorrenza per gli utenti Kinesis con fan-out avanzato (flussi HTTP/2):, ,, e. ParallelizationFactor MaximumRetryAttempts MaximumRecordAgeIn Seconds DestinationConfig BisectBatchOnFunctionError Per i dettagli, consulta Utilizzo AWS Lambda con Amazon Kinesis.	7 luglio 2020

Periodo di batch per i consumer dei flussi Kinesis HTTP/2	Ora puoi configurare una finestra batch (MaximumBatchingWindowInsecondi) per i flussi HTTP/2. Lambda legge i record dal flusso fino a quando non ha raccolto un batch completo o fino alla scadenza del periodo di batch. Per i dettagli, 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 for. AWS Cloud Development Kit (AWS CDK) TypeScript AWS CDK È un framework che consente di definire le risorse dell'applicazione in Python TypeScript, Java o.NET.	1 giugno 2020
Support per il runtime di.NET Core 3.1.0 in AWS Lambda	AWS Lambda ora supporta il runtime di.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 una funzione con AWS 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 il ciclo di vita degli stati Lambda aggiornato per le reti VPC sul 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 codici motivo a StateReasonCode (InvalidSubnet, InvalidSecurityGroup) e LastUpdateStatusReasonCode (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 raggiunge re livelli di concorrenza elevati senza esaurire le connessio ni al database. Per informazi oni 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'orig ine 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 le funzioni connesse con VPC, ma potrebbe richieder e un aggiornamento dell'auto mazione. 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'argomen to relativo alla configurazione della gestione degli errori per l'invocazione asincrona.	25 novembre 2019

Gestione degli errori per le origini eventi di flusso	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.	25 novembre 2019

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, consultar e l'argomento relativo alla configurazione delle destinazioni per l'invocazione asincrona .	25 novembre 2019
Nuovi runtime per Node.js, Python e Java	Nuovi runtime sono disponibili 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 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 inserisco no modifiche al repository, la pipeline inclusa le compila e le distribuisce automaticamente. Per istruzioni, consulta Gestione delle applicazioni nella console Lambda.	3 ottobre 2019

Miglioramenti delle prestazioni per le funzioni connesse al VPC	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 configura re 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. Per i dettagli, consulta Funzioni di monitoraggio nella AWS Lambda console.	18 giugno 2019
Amazon Linux 2018.03	L'ambiente di esecuzione Lambda è in fase di aggiornam ento per consentire l'uso di Amazon Linux 2018.03. Per ulteriori dettagli, consultare Ambiente di esecuzione.	21 maggio 2019

Node.js 10	Per Node.js 10 è disponibile il nuovo runtime nodejs10.x. Questo runtime usa Node.js 10.15 e viene periodicamente aggiornato con le ultime versioni di Node.js 10. Node.js 10 è anche il primo runtime per utilizzare Amazon Linux 2. Per ulteriori dettagli, consultar e Compilazione di funzioni Lambda con Node.js.	13 maggio 2019
GetLayerVersionByArn API	Utilizzate l'API GetLayerV ersionByArn per scaricare le informazioni sulla versione del layer con la versione ARN come input. Rispetto a GetLayerVersion, GetLayerV ersionByArn consente di utilizzare l'ARN direttame nte anziché 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 informazi oni dettagliate, consultar 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 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

Utilizzo dei consumer del flusso Kinesis HTTP/2 come trigger	È possibile utilizzare i consumatori del flusso Kinesis HTTP/2 per l'invio di eventi a AWS Lambda. I consumatori del flusso hanno throughput in lettura dedicato da ogni shard nel flusso di dati e 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 Compilazione 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 Regione (Stati Uniti orientali)	AWS Lambda è ora disponibile nella regione AWS GovCloud (Stati Uniti orientali).	12 novembre 2018

AWS SAM Gli argomenti sono stati spostati in una guida per sviluppatori separata	Alcuni argomenti si sono concentrati 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 Applications (Applicazioni) della console Lambda. Questa pagina mostra lo stato dello AWS 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

Support per il linguaggio PowerShell Core in AWS Lambda	AWS Lambda ora supporta il linguaggio PowerShell Core. Per ulteriori informazioni, consulta Modello di programmazione per la creazione di funzioni Lambda in. PowerShell	11 settembre 2018
Support per il runtime di.NET Core 2.1.0 in AWS Lambda	AWS Lambda ora supporta il runtime di.NET Core 2.1.0. Per ulteriori informazioni, consultare Interfaccia a riga di comando (CLI) di .NET Core.	9 luglio 2018
Aggiornamenti ora disponibili tramite RSS	Ora è possibile iscriverti a un feed RSS per seguire la pubblicazione di nuove versioni di questa guida.	5 luglio 2018
Supporto per Amazon SQS come origine evento	AWS Lambda ora supporta Amazon Simple Queue Service (Amazon SQS) come fonte di eventi. Per ulteriori informazioni, consultare Richiamo delle funzioni Lambda.	28 giugno 2018
Regione Cina (Ningxia)	AWS Lambda è ora disponibile nella regione Cina (Ningxia). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina Regioni ed endpoint 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 il runtime del Node.js 8.10	AWS Lambda ora supporta la versione 8.10 del runtime di Node.js. Per ulteriori informazioni, consulta Compilazione di funzioni Lambda con Node.js.	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 Compilazione di funzioni Lambda con Go e Compilazione di funzioni Lambda con C#.	15 gennaio 2018
Nuovo progetto della console	AWS Lambda ha introdotto una nuova console Lambda per semplificare l'esperienza e ha aggiunto un editor di codice Cloud9 per migliorare le tue capacità di debug e rivedere il codice della funzione. Per ulteriori informazioni, consulta Modifica il codice utilizzando l'editor della console Lambda.	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 Configurazione della concorrenza riservata per una funzione.	30 novembre 2017
Trasferimento del traffico con gli alias	AWS Lambda ora supporta lo spostamento del traffico con alias. Per ulteriori informazioni, consulta Crea distribuzioni continue per le funzioni Lambda.	28 novembre 2017

Modifica	Descrizione	Data
Distribuzione graduale del codice	AWS Lambda ora supporta l'implementazione sicura di nuove versioni della funzione Lambda sfruttando Code Deploy. Per ulteriori informazioni, consultare Distribuzione graduale del codice.	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 Regioni ed endpoint nella Riferimenti generali di AWS.	9 novembre 2017
Introduzione a SAM Local	AWS Lambda introduce SAM Local (ora noto come SAM CLI), AWS CLI uno strumento che fornisce un ambiente per sviluppare, testare e analizzare le applicazioni serverless a livello locale prima di caricarle nel runtime Lambda. Per ulteriori informazioni, consultare Esecuzione di test e debug di applicazioni serverless.	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 Regioni ed endpoint 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 Regioni ed endpoint 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 prestazio ni delle applicazioni Lambda. Per ulteriori informazioni, consultare Visualizza le chiamate alla funzione Lambda utilizzando AWS X-Ray.	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 Regioni ed endpoint nella Riferimenti generali di AWS.	28 marzo 2017
AWS Lambda ora supporta il runtime di Node.js v6.10	AWS Lambda ha aggiunto il supporto per Node.js runtime v6.10. Per ulteriori informazioni, consulta Compilazione di funzioni Lambda con Node.js.	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 Regioni ed endpoint nella Riferimenti generali di AWS.	1 febbraio 2017
AWS Lambda supporto per il runtime di.NET, Lambda @Edge (Preview), Dead Letter Queues e distribuzione automatizzata di applicazioni serverless.	AWS Lambda ha aggiunto il supporto per C#. Per ulteriori informazioni, consulta Compilazione di funzioni Lambda con C#. Lambda @Edge consente di eseguire funzioni Lambda nelle posizioni AWS Edge in risposta agli eventi. CloudFront Per ulteriori informazioni, consulta Utilizzo AWS Lambda con CloudFront Lambda @Edge.	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. Per ulteriori informazioni, consultare Uso di AWS Lambda con Amazon Lex.	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 Regioni ed endpoint nella Riferimenti generali di AWS.	21 novembre 2016

Modifica	Descrizione	Data
È stato introdotto AWS SAM per la creazione e la distribuzione di applicazioni basate su Lambda e l'utilizzo di variabili di ambiente per le impostazioni di configurazione delle funzioni Lambda.	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'applica zione, è sufficiente specificare le risorse necessarie come parte dell'applicazione, insieme alle policy di autorizza zione associate in un file di modello AWS CloudForm ation , scritto in JSON o YAML, creare un pacchetto degli artefatti di distribuzione e distribuire il modello. Per ulteriori informazioni, consultare AWS Lambda applicazioni. 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 Usa le variabili di ambiente Lambda per configurare i valori nel codice.	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 Regioni ed endpoint 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 Regioni ed endpoint 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 Node.js runtime v4.3	AWS Lambda ha aggiunto il supporto per Node.js runtime v4.3. Per ulteriori informazioni, consulta Compilazione di funzioni Lambda con Node.js.	07 aprile 2016

Modifica	Descrizione	Data
Regione Europa (Francoforte)	Lambda è ora disponibile nella regione Europa (Francoforte). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina Regioni ed endpoint nella Riferimenti generali di AWS.	14 marzo 2016
Supporto per VPC	Ora è possibile configurare una funzione Lambda per accedere alle risorse del proprio VPC. Per ulteriori informazioni, consultare Offrire alle funzioni Lambda l'accesso alle risorse in un Amazon VPC.	11 febbraio 2016
Il runtime Lambda è stato aggiornato.	L'ambiente di esecuzione è stato aggiornato.	4 novembre 2015

Modifica	Descrizione	Data
Supporto per il controllo delle versioni, Python per lo sviluppo di codice per le funzioni Lambda, eventi pianificati e aumento del runtime	Ora è possibile sviluppare il codice della funzione Lambda tramite Python. Per ulteriori informazioni, consultare Compilazione di funzioni Lambda con Python. 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 Versioni delle funzioni Lambda. 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 Utilizzo di Lambda con Amazon Scheduler EventBridge . 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.	08 ottobre 2015
Supporto per DynamoDB Streams	DynamoDB Streams è ora disponibile a livello generale e può essere utilizzato in tutte le regioni in cui è disponibi 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 aggiornam enti eseguiti sulla tabella DynamoDB. Per un esempio di procedura guidata, consulta la pagina Tutorial: Utilizzo AWS Lambda con flussi Amazon DynamoDB.	14 luglio 2015

Modifica	Descrizione	Data
Lambda ora supporta l'invocazione di funzioni Lambda con client compatibili REST.	Fino ad ora, per richiamare la funzione Lambda dall'applicazione Web, mobile o IoT, erano necessari AWS gli SDK (ad esempio, SDK AWS per Java, SDK AWS per Android o SDK per iOS). AWS Ora Lambda supporta l'invocazione di una funzione Lambda con client compatibili REST attraverso un'API personalizzata che è possibile creare utilizzando Gateway Amazon API. È possibile inviare richieste all'URL dell'endpoint della funzione Lambda. È possibile configurare la sicurezza sull'endpoint per consentire l'accesso aperto, sfruttare AWS Identity and Access Management (IAM) per autorizzare l'accesso o utilizzare le chiavi API per misurare l'accesso alle proprie funzioni Lambda da parte di altri. Per un esercizio di Nozioni di base di esempio, consultare Richiamo di una funzione Lambda utilizzando un endpoint Amazon API Gateway. Per ulteriori informazioni su Gateway Amazon API, consultare https://aws.amazon.com/api-gateway/.	09 luglio 2015
La console Lambda ora fornisce piani con cui creare facilmente funzioni Lambda e testarle.	La console Lambda fornisce un set di piani. Ciascuno di essi fornisce una configurazione di origini eventi e un codice di esempio per la funzione Lambda che è possibile utilizzare per creare facilmente applicazioni basate su Lambda. Tutti gli esercizi delle Nozioni di base di Lambda ora utilizzano i piani. Per ulteriori informazioni, consultare Guida introduttiva a Lambda.	09 luglio 2015
Lambda ora supporta Java per la creazione delle funzioni Lambda.	Ora è possibile creare codice Lambda in Java. Per ulteriori informazioni, consultare Compilazione di funzioni Lambda con Java.	15 giugno 2015

Modifica	Descrizione	Data
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. La documentazione è stata aggiornata di conseguenza. Per informazioni, consultare gli argomenti seguenti: Guida introduttiva a Lambda AWS 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.