aws

Guia do Desenvolvedor

AWS Lambda

AWS Lambda: Guia do Desenvolvedor

Table of Contents

O que é o AWS Lambda?

É possível usar o AWS Lambda para executar código sem provisionar ou gerenciar servidores.

O Lambda executa seu código em uma infraestrutura de computação de alta disponibilidade e executa toda a administração dos recursos computacionais, incluindo manutenção do servidor e do sistema operacional, provisionamento e escalabilidade automática da capacidade e registro em log do código. Com o Lambda, tudo o que você precisa fazer é fornecer seu código em uma dos runtimes de linguagens compatíveis com o Lambda.

Você organiza seu código emFunções do Lambda. O serviço do Lambda executa a função somente quando necessário e escala automaticamente. Você paga apenas pelo tempo de computação consumido. Não haverá cobranças quando o código não estiver em execução. Para obter mais informações, consulte Preços do AWS Lambda.

> **Tip**
>
> Para saber como construir soluções com tecnologia sem servidor, confira o Guia do desenvolvedor com tecnologia sem servidor.

Quando usar o Lambda

O Lambda é um serviço de computação ideal para cenários de aplicações que precisam aumentar rapidamente a escala verticalmente e reduzir a escala verticalmente a zero quando não estão sob demanda. Por exemplo, é possível usar o Lambda para:

- Processamento de arquivos: use o Amazon Simple Storage Service (Amazon S3) para acionar o processamento de dados do Lambda em tempo real após a realização de um upload.
- Processamento de fluxo: use o Lambda e o Amazon Kinesis para processar dados de streaming em tempo real para monitoramento de atividades de aplicações, processamento de pedidos de transações, análise de clickstream, limpeza de dados, filtragem de logs, indexação, análise de mídias sociais, telemetria de dados de dispositivos da Internet das Coisas (IoT) e medição.
- Aplicações Web: combine o Lambda com outros serviços da AWS para desenvolver aplicações Web avançadas que diminuem e aumentam a escala verticalmente de forma automática, além de executar em uma configuração altamente disponível em diversos datacenters.

- Back-ends IoT: desenvolva back-ends com tecnologia sem servidor usando o Lambda para tratar de solicitações de API da Web, de dispositivos móveis, de IoT e de terceiros.
- Back-ends móveis: desenvolva back-ends usando o Lambda e o Amazon API Gateway para autenticar e processar solicitações de API. Use o AWS Amplify para facilitar a integração com os frontends do iOS, Android, Web e React Native.

Ao usar o Lambda, você é responsável apenas pelo seu código. O Lambda gerencia a frota de computação que oferece um equilíbrio de memória, CPU, rede e outros recursos para executar seu código. Como o Lambda gerencia esses recursos, não é possível fazer login para calcular instâncias ou personalizar o sistema operacional noRuntime fornecido. O Lambda executa atividades operacionais e administrativas em seu nome, incluindo gerenciamento de capacidade, monitoramento e registro de suas funções do Lambda.

Atributos principais

Os principais recursos a seguir ajudam você a desenvolver aplicações do Lambda escaláveis, seguras e facilmente extensíveis:

Variáveis de ambiente

Usa variáveis de ambiente para ajustar o comportamento da função sem atualizar o código.

Versões

Gerencia a implantação de suas funções com versões para que, por exemplo, uma nova função possa ser usada para testes beta sem afetar os usuários da versão de produção estável.

Imagens de contêiner

Cria uma imagem de contêiner para uma função do Lambda básica usando uma imagem básica da fornecida pela AWS ou uma imagem básica alternativa para que você possa reutilizar as ferramentas de contêiner existentes ou implantar workloads maiores que dependem de dependências consideráveis, como machine learning.

Camadas

Bibliotecas de pacotes e outras dependências para reduzir o tamanho dos arquivos de implantação e acelerar a implantação do seu código.

Extensões do Lambda

Amplia suas funções do Lambda com ferramentas para monitoramento, observabilidade, segurança e governança.

URLs da função

Adiciona um endpoint de HTTP(S) dedicado à sua função do Lambda.

Streaming de respostas

Configura seus URLs de funções do Lambda para fazer o streaming de cargas de resposta de volta aos clientes a partir das funções do Node.js, para melhorar o desempenho do tempo até o primeiro byte (TTFB) ou para retornar cargas maiores.

Controles de simultaneidade e escalabilidade

Aplicam controle detalhado sobre a escalabilidade e a capacidade de resposta das aplicações de produção.

Assinatura de código

Verifica se somente desenvolvedores aprovados publicam código confiável e inalterado em suas funções do Lambda

Redes privadas

Cria uma rede privada para recursos como bancos de dados, instâncias de cache ou serviços internos.

Acesso ao sistema de arquivos

Configura uma função para montar um Amazon Elastic File System (Amazon EFS) em um diretório local, de forma que o código da função possa acessar e modificar os recursos compartilhados com segurança e com alta simultaneidade.

Lambda SnapStart para Java

Melhora o desempenho de inicialização para ambientes de runtimes de Java em até dez vezes sem custos extras e, geralmente, sem alterações no código da função.

Conceitos básicos do Lambda

Para começar a usar o Lambda, use o console do Lambda para criar uma função. Em alguns minutos, você pode criar e implantar uma função e testá-la no console.

Ao executar o tutorial, você aprenderá alguns conceitos fundamentais do Lambda, como passar argumentos para sua função usando o objeto de evento do Lambda. Você também aprenderá a retornar as saídas de log da sua função e a visualizar os logs de invocação da função no CloudWatch Logs.

Para simplificar, você cria sua função usando o runtime do Python ou do Node.js. Com essas linguagens interpretadas, você pode editar o código da função diretamente no editor de código integrado do console. Com linguagens compiladas como Java e C#, você precisa criar um pacote de implantação em sua máquina de compilação local e carregá-lo no Lambda. Para saber mais sobre a implantação de funções no Lambda usando outros runtimes, consulte os links na seção the section called “Recursos adicionais e próximas etapas”.

> **Tip**
>
> Para saber como construir soluções com tecnologia sem servidor, confira o Guia do desenvolvedor com tecnologia sem servidor.

Pré-requisitos

Cadastre-se em uma Conta da AWS

Se você ainda não tem Conta da AWS, siga as etapas a seguir para criar uma.

Para se cadastrar em uma Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

   Quando você se cadastra em uma Conta da AWS, um Usuário raiz da conta da AWS é criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como uma prática

recomendada de segurança, atribua o acesso administrativo para um usuário e use somente o usuário-raiz para executar tarefas que requerem o acesso de usuário-raiz.

A AWS envia um e-mail de confirmação depois que o processo de cadastramento é concluído. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se cadastrar em uma Conta da AWS, proteja seu Usuário raiz da conta da AWS, habilite o AWS IAM Identity Center e crie um usuário administrativo para não usar o usuário raiz em tarefas cotidianas.

Proteger seu Usuário raiz da conta da AWS

1. Faça login no AWS Management Console como o proprietário da conta ao escolher a opção Usuário raiz e inserir o endereço de e-mail da Conta da AWS. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Signing in as the root user (Fazer login como usuário-raiz) no Guia do usuário do Início de Sessão da AWS.
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo MFA virtual para o usuário-raiz de sua conta da Conta da AWS (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center.
2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para obter um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso dos usuários com o Diretório do Centro de Identidade do IAM padrão no Guia do usuário do AWS IAM Identity Center.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use a URL de login que foi enviada ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

  Para obter ajuda com o login utilizando um usuário do Centro de Identidade do IAM, consulte Fazer login no portal de acesso da AWS, no Guia do usuário do Início de Sessão da AWS.

Atribuir acesso para usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center.
2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center.

Criar uma função do Lambda com o console

Neste exemplo, sua função usa um objeto JSON contendo dois valores inteiros rotulados `"length"` e `"width"`. A função multiplica esses valores para calcular uma área e a retorna como uma string JSON.

Sua função também imprime a área calculada, junto com o nome do grupo de logs do CloudWatch. Posteriormente, no tutorial, você aprenderá a usar o CloudWatch Logs para visualizar os registros da invocação de suas funções.

Para criar sua função, primeiro é necessário criar uma função básica Hello world. Na etapa a seguir, você então adiciona seu próprio código de função.

Criar uma função do Lambda hello world com o console

1. Abra a página Funções do console do Lambda.
2. Escolha a opção Criar função.
3. Selecione Criar do zero.
4. No painel Informações básicas, para Nome da função, insira **`myLambdaFunction`**.

5. Em Runtime, escolha Node.js 20.x ou Python 3.12
6. Deixe arquitetura definido como x86_64 e escolha Criar função.

O Lambda cria uma função que retorna a mensagem `Hello from Lambda!`. O Lambda também cria um perfil de execução para sua função. Um perfil de execução é um perfil do AWS Identity and Access Management (IAM) que concede a uma função do Lambda permissão para acessar recursos e Serviços da AWS. Para sua função, o perfil criado pelo Lambda concede permissões básicas para gravar no CloudWatch Logs.

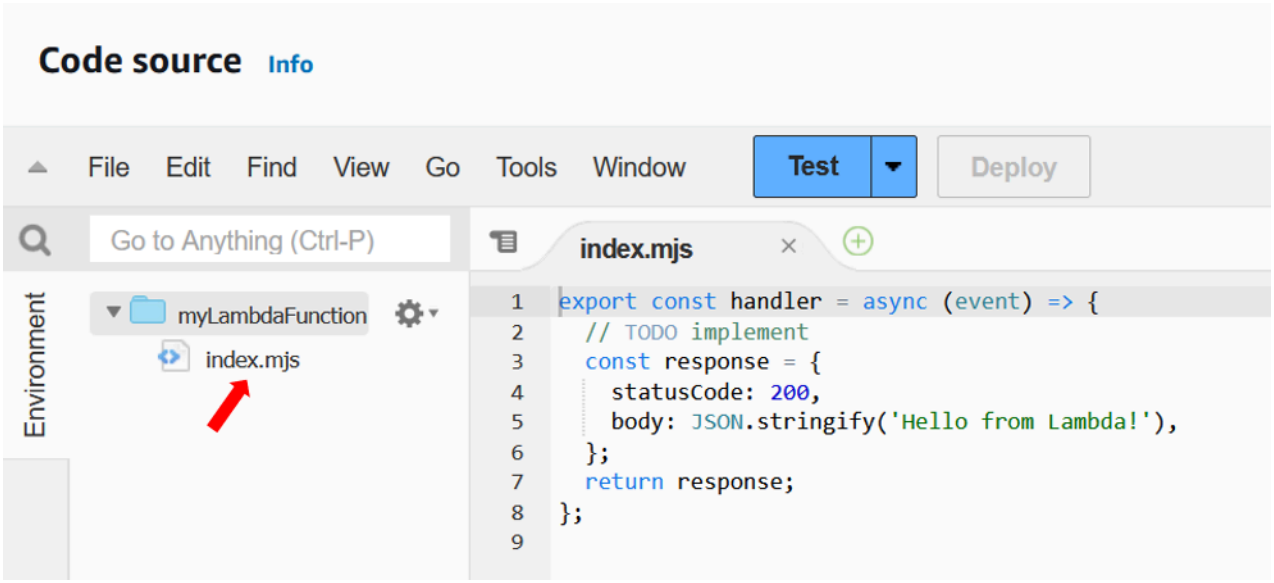
Agora você usa o editor de código integrado do console para substituir o código Hello world criado pelo Lambda com seu próprio código de função.

Node.js

Modificar o código no console

1. Escolha a guia Código.

   No editor de código integrado do console, você deve ver o código da função que o Lambda criou. Se você não vir a guia index.mjs no editor de código, selecione index.mjs no explorador de arquivos, conforme mostrado no diagrama a seguir.



2. Cole o código a seguir na guia index.mjs, substituindo o código que o Lambda criou.

```
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. Selecione Implantar para atualizar a função do seu código. Quando o Lambda implementa as alterações, o console exibe um banner informando que sua função foi atualizada com sucesso.

Entender seu código de função

Antes de passar para a próxima etapa, vamos analisar o código da função e entender alguns conceitos-chave do Lambda.

- O manipulador do Lambda:

  Sua função Lambda contém uma função Node.js chamada `handler`. Uma função do Lambda em Node.js pode conter mais de uma função Node.js, mas a função manipuladora é sempre o ponto de entrada para seu código. Quando sua função é invocada, o Lambda executa esse método.

  Quando você criou sua função Hello world usando o console, o Lambda definiu automaticamente o nome do método manipulador para sua função como `handler`. Certifique-se de não editar o nome dessa função Node.js. Caso faça isso, o Lambda não poderá executar seu código quando você invocar sua função.

  Para saber mais sobre o manipulador Lambda em Node.js, consulte the section called “Manipulador”.
- O objeto do evento Lambda:

A função `handler` recebe dois argumentos, `event` e `context`. Um evento no Lambda corresponde a um documento formatado em JSON que contém dados para a sua função processar.

Se sua função for invocada por outro AWS service (Serviço da AWS), o objeto de evento via conter informações sobre o evento que causou a invocação. Por exemplo, se um bucket do Amazon Simple Storage Service (Amazon S3) invocar sua função quando um objeto for carregado, o evento vai conter o nome do bucket do Amazon S3 e a chave do objeto.

Neste exemplo, você criará um evento no console inserindo um documento formatado em JSON com dois pares de valores-chave.

- O objeto de contexto Lambda:

  O segundo argumento que sua função assume é `context`. O Lambda passa o objeto de contexto para sua função automaticamente. O objeto de contexto contém informações sobre a invocação da função e o ambiente de execução.

  Você pode usar o objeto de contexto para gerar informações sobre a invocação da sua função para fins de monitoramento. Neste exemplo, sua função usa o parâmetro `logGroupName` para gerar o nome de seu grupo de logs do CloudWatch.

  Para saber mais sobre o objeto de contexto do Lambda em Node.js, consulte the section called “Contexto”.

- Registro em log no Lambda:

  Com o Node.js, você pode usar métodos de console, como `console.log` e `console.error`, para enviar informações para o log da sua função. O código de exemplo usa instruções `console.log` para gerar a área calculada e o nome do grupo de logs do CloudWatch da função. Você também pode usar qualquer biblioteca de registro em log que grava em `stdout` ou `stderr`.

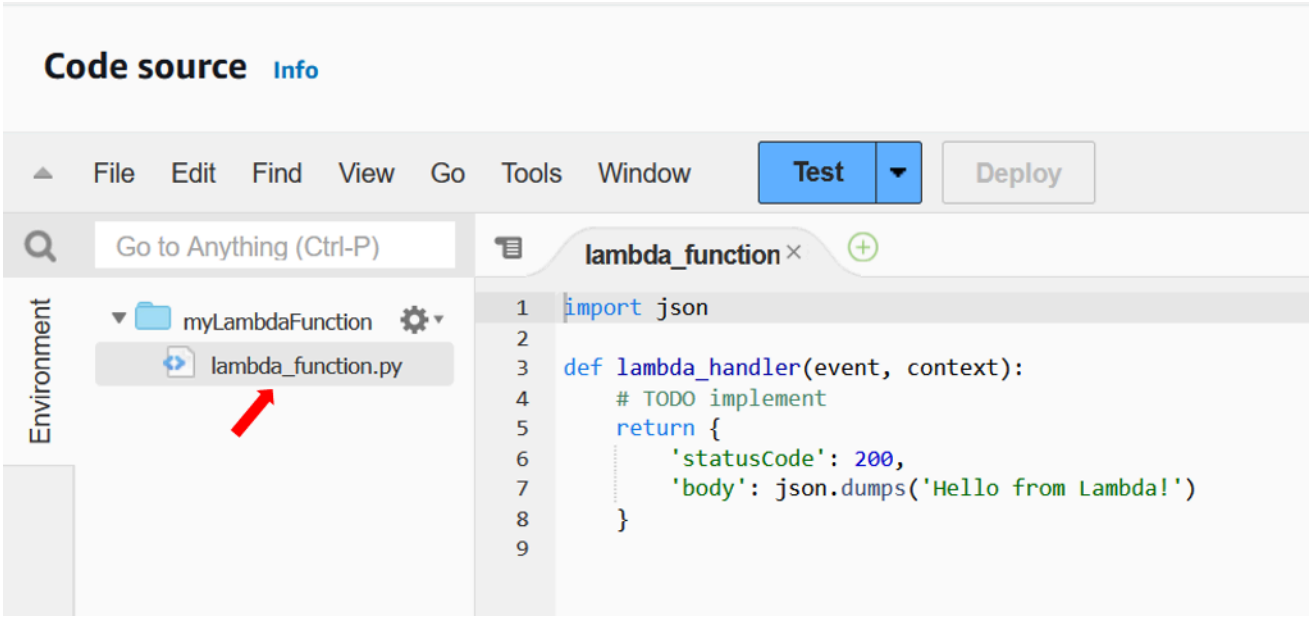
  Para saber mais, consulte the section called “Registro em log”. Para saber mais sobre registro em log em outros runtimes, consulte as páginas “Criar com” para ver os runtimes nos quais você está interessado.

Python

Modificar o código no console

1. Escolha a guia Código.

   No editor de código integrado do console, você deve ver o código da função que o Lambda criou. Se você não vir a guia lambda_function.py no editor de código, selecione lambda_function.py no explorador de arquivos, conforme mostrado no diagrama a seguir.



2. Cole o código a seguir na guia lambda_function.py, substituindo o código que o Lambda criou.

```
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. Selecione Implantar para atualizar a função do seu código. Quando o Lambda implementa as alterações, o console exibe um banner informando que sua função foi atualizada com sucesso.

Entender seu código de função

Antes de passar para a próxima etapa, vamos analisar o código da função e entender alguns conceitos-chave do Lambda.

- O manipulador do Lambda:

  Sua função do Lambda contém uma função Python chamada `lambda_handler`. Uma função do Lambda em Python pode conter mais de uma função Python, mas a função manipulador é sempre o ponto de entrada para seu código. Quando sua função é invocada, o Lambda executa esse método.

  Quando você criou sua função Hello world usando o console, o Lambda definiu automaticamente o nome do método manipulador para sua função como `lambda_handler`. Certifique-se de não editar o nome dessa função Python. Caso faça isso, o Lambda não poderá executar seu código quando você invocar sua função.

  Para saber mais sobre o manipulador Lambda em Python, consulte the section called “Manipulador”.

- O objeto do evento Lambda:

  A função `lambda_handler` recebe dois argumentos, `event` e `context`. Um evento no Lambda corresponde a um documento formatado em JSON que contém dados para a sua função processar.

Se sua função for invocada por outro AWS service (Serviço da AWS), o objeto de evento via conter informações sobre o evento que causou a invocação. Por exemplo, se um bucket do Amazon Simple Storage Service (Amazon S3) invocar sua função quando um objeto for carregado, o evento vai conter o nome do bucket do Amazon S3 e a chave do objeto.

Neste exemplo, você criará um evento no console inserindo um documento formatado em JSON com dois pares de valores-chave.

- O objeto de contexto Lambda:

  O segundo argumento que sua função assume é `context`. O Lambda passa o objeto de contexto para sua função automaticamente. O objeto de contexto contém informações sobre a invocação da função e o ambiente de execução.

  Você pode usar o objeto de contexto para gerar informações sobre a invocação da sua função para fins de monitoramento. Neste exemplo, sua função usa o parâmetro `log_group_name` para gerar o nome de seu grupo de logs do CloudWatch.

  Para saber mais sobre o objeto de contexto do Lambda em Python, consulte the section called “Contexto”.

- Registro em log no Lambda:

  Com o Python, você pode usar uma instrução `print` ou uma biblioteca de registro em log do Python para enviar informações ao log da sua função. Para ilustrar a diferença no que é capturado, o código de exemplo usa os dois métodos. Em uma aplicação de produção, recomendamos que você use uma biblioteca de registro em log.

  Para saber mais, consulte the section called “Registro em log”. Para saber mais sobre registro em log em outros runtimes, consulte as páginas “Criar com” para ver os runtimes nos quais você está interessado.

Invocar a função do Lambda usando o console

Para invocar sua função usando o console do Lambda, primeiro você cria um evento de teste para enviar à sua função. O evento é um documento formatado em JSON contendo dois pares de valores-chave com as chaves `"length"` e `"width"`.

Criar o evento de teste

1. No painel Origem do código, escolha Testar.
2. Selecione Criar novo evento.
3. Em Nome do evento, insira **`myTestEvent`**.
4. No painel Evento JSON, substitua os valores padrão colando o seguinte:

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

5. Escolha Salvar.

Agora você testa sua função e usa o console do Lambda e o CloudWatch Logs para visualizar registros da invocação da sua função.

Para testar sua função e visualizar os registros de invocação no console

- No painel Origem do código, escolha Testar. Quando sua função terminar de ser executada, você verá os logs de resposta e função exibidos na guia Resultados da execução. Você deve ver resultados semelhantes ao seguinte:

  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
```

Neste exemplo, você invocou seu código usando o atributo de teste do console. Isso significa que você pode ver os resultados da execução da função diretamente no console. Quando sua função é invocada fora do console, você precisa usar o CloudWatch Logs.

Para visualizar os registros de invocação da sua função no CloudWatch Logs

1. Abra a página Log groups (Grupos de log) do console do CloudWatch.
2. Escolha o nome do grupo de logs para sua função (`/aws/lambda/myLambdaFunction`). Esse é o nome do grupo de logs que sua função imprimiu no console.
3. Na guia Fluxos de log, escolha o fluxo de logs para a invocação da sua função.

   Você deve ver saída semelhante a:

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
```

Limpeza

Quando você tiver terminado de trabalhar com a função de exemplo, exclua-a. Você também pode excluir o grupo de logs que armazena os logs da função e a função de execução que o console criou.

Para excluir uma função do Lambda

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Ações, Excluir.

4. Na caixa de diálogo Delete function (Excluir função), digite delete (excluir) e escolha Delete (Excluir).

Para excluir o grupo de logs

1. Abra a página Log groups (Grupos de log) do console do CloudWatch.
2. Selecione o grupo de logs da função (`/aws/lambda/my-function`).
3. Selecione Actions (Ações), Delete log group(s) (Excluir grupo(s) de log).
4. Na caixa de diálogo Delete log group(s) (Excluir grupo(s) de logs), escolha Delete (Excluir).

Para excluir a função de execução

1. Abra a página Roles (Funções) no console do AWS Identity and Access Management (IAM).
2. Selecione o perfil de execução da função (por exemplo, `myLambdaFunction-role-31exxmpl`).
3. Escolha Excluir.
4. Na caixa de diálogo Delete role (Excluir perfil), insira o nome do perfil e, em seguida, escolha Delete (Excluir).

Você pode automatizar a criação e a limpeza de funções, papéis e grupos de log com o AWS CloudFormation e a AWS Command Line Interface (AWS CLI).

Recursos adicionais e próximas etapas

Agora que você criou e testou uma função do Lambda simples usando o console, siga estes próximos passos:

- Aprenda a adicionar dependências ao seu código e implantá-lo usando um pacote de implantação.zip. Escolha entre os links a seguir os idiomas nos quais você está interessado.

  Node.js

  Consulte the section called “Implantar arquivos .zip”

  Typescript

  Consulte the section called “Implantar arquivos .zip”

Python

Consulte the section called “Implantar arquivos .zip”

Ruby

Consulte the section called “Implantar arquivos .zip”

Java

Consulte the section called “Implantar arquivos .zip”

Go

Consulte the section called “Implantar arquivos .zip”

C#

Consulte the section called “Pacote de implantação”

- Execute o tutorial Usar um trigger do Amazon S3 para invocar uma função do Lambda para aprender a configurar uma função do Lambda para ser invocada por outro AWS service (Serviço da AWS).
- Escolha um dos tutoriais a seguir para obter um exemplo mais complexo do uso do Lambda com outros Serviços da AWS.
  - Usar o Lambda com o API Gateway: crie uma API REST do Amazon API Gateway que invoca uma função do Lambda.
  - Usar uma função do Lambda para acessar um banco de dados do Amazon RDS: use uma função do Lambda para gravar dados em um banco de dados do Amazon Relational Database Service (Amazon RDS) por meio do RDS Proxy.
  - Usar um trigger do Amazon S3 para criar imagens em miniatura: use uma função do Lambda para criar uma miniatura sempre que um arquivo de imagem for carregado em um bucket do Amazon S3.

Fundamentos AWS Lambda

A função do Lambda é o principal recurso do serviço do Lambda.

Você pode configurar suas funções usando o console do Lambda, a API do Lambda, AWS CloudFormation ou AWS SAM. Você cria código para a função e carrega o código usando um pacote de implantação. Quando o evento ocorre, o Lambda invoca a função. O Lambda executa várias instâncias de sua função em paralelo, regidas por simultaneidade e limites de escalabilidade.

Tópicos

Conceitos Lambda

O Lambda executa instâncias de sua função para processar eventos. É possível invocar a função diretamente usando a API do Lambda ou configurar um serviço da AWS ou um recurso para invocá-la.

Conceitos

- Função
- Trigger
- Evento
- Ambiente de execução
- arquiteturas de conjunto de instruções
- Pacote de implantação
- Runtime
- Camada
- Extensão
- Simultaneidade
- Qualifier
- Destino

Função

Uma função é um recurso que você pode invocar para executar o código no Lambda. Uma função tem código para processar os eventos que você transmite para a função ou que outros serviços da AWS enviam para a função.

Trigger

Um acionador é um recurso ou uma configuração que invoca uma função do Lambda. Acionadores incluem serviços da AWS que podem ser configurados para invocar uma função e mapeamentos da fonte do evento. Um mapeamento da fonte do evento é um recurso no Lambda que lê itens de um fluxo ou de uma fila e invoca uma função. Para obter mais informações, consulte Invocando o Lambda com eventos de outros serviços da AWS e Compreender os métodos de invocação de funções do Lambda.

Evento

Um evento é um documento no formato JSON que contém dados para uma função do Lambda processar. O tempo de execução converte o evento em um objeto e o transmite para o código da função. Ao invocar uma função, você determina a estrutura e o conteúdo do evento.

Example evento personalizado – dados climáticos

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

Quando um serviço da AWS invoca a função, ele define a forma do evento.

Example Evento de serviço — Notificação do 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!",
        ...
```

Para obter mais informações sobre eventos de serviços da AWS, consulte Invocando o Lambda com eventos de outros serviços da AWS.

Ambiente de execução

Um ambiente de execução fornece um ambiente de execução seguro e isolado para a função do Lambda. Um ambiente de execução gerencia os processos e recursos necessários para executar a função. O ambiente de execução fornece suporte ao ciclo de vida para a função e a qualquer extensão associada à função.

Para ter mais informações, consulte Ambiente de execução do Lambda.

arquiteturas de conjunto de instruções

A arquitetura do conjunto de instruções determina o tipo de processador de computador que o Lambda usa para executar a função. O Lambda fornece opções de arquiteturas de conjuntos de instruções:

- arm64: arquitetura ARM de 64 bits para o processador AWS Graviton2.
- x86_64: arquitetura x86 de 64 bits para processadores baseados em x86.

Para ter mais informações, consulte Configurar a arquitetura do conjunto de instruções para uma função do Lambda.

Pacote de implantação

Você implanta o código da função Lambda usando umpacote de implantação. O Lambda oferece suporte a dois tipos de pacote de implantação:

- Um arquivo .zip que contém o código da função e suas dependências. O Lambda fornece o sistema operacional e o tempo de execução para sua função.
- Uma imagem de contêiner compatível com a especificação Open Container Initiative (OCI). Você adiciona o código da função e as dependências à imagem. Também é necessário incluir o sistema operacional e um tempo de execução do Lambda.

Para ter mais informações, consulte Pacotes de implantação do Lambda.

Runtime

O tempo de execução fornece um ambiente específico de linguagem que é executado no ambiente de execução. O tempo de execução transmite eventos de invocação, informações de contexto e respostas entre o Lambda e a função. Você pode usar tempos de execução fornecidos pelo Lambda ou criar seus próprios. Se você empacotar seu código como um arquivo de arquivo.zip, você deve configurar sua função para usar um tempo de execução que corresponda à sua linguagem de programação. Para uma imagem de contêiner, você inclui o tempo de execução ao compilar a imagem.

Para ter mais informações, consulte Runtimes do Lambda.

Camada

Uma camada do Lambda é um arquivo .zip que pode conter código adicional ou outro conteúdo. Uma camada pode conter bibliotecas, um runtime personalizado, dados ou arquivos de configuração.

As camadas fornecem uma maneira conveniente de empacotar bibliotecas e outras dependências que você pode usar com suas funções do Lambda. O uso de camadas reduz o tamanho dos arquivos de implantação carregados e acelera a implantação do código. As camadas também promovem o compartilhamento de código e a separação de responsabilidades para que você possa iterar mais rapidamente na escrita da lógica de negócios.

Você pode incluir até cinco camadas por função. As camadas contam para as cotas de tamanho de implantação padrão do Lambda. Quando você inclui uma camada em uma função, o conteúdo é extraído para o diretório `/opt` no ambiente de execução.

Por padrão, as camadas que você cria são privadas na sua conta da AWS. Você pode optar por compartilhar uma camada com outras contas ou tornar a camada pública. Se suas funções consomem uma camada que uma conta diferente publicou, suas funções poderão continuar a usar a versão da camada depois que ela tiver sido excluída ou depois que sua permissão para acessar a camada for revogada. No entanto, você não pode criar uma nova função ou atualizar funções usando uma versão de camada excluída.

Funções implantadas como uma imagem de contêiner não usam camadas. Em vez disso, você empacota seu tempo de execução preferido, bibliotecas e outras dependências na imagem do contêiner ao criar a imagem.

Para ter mais informações, consulte Camadas do Lambda.

Extensão

As extensões do Lambda permitem que você aumente as funções. Por exemplo, você pode usar extensões para integrar as funções com suas ferramentas preferidas de monitoramento, observação, segurança e governança. Você pode escolher entre várias ferramentas que os parceiros do AWS Lambda fornecem, ou pode criar suas próprias extensões do Lambda.

Uma extensão interna é executada no processo de tempo de execução e compartilha o mesmo ciclo de vida que o tempo de execução. Uma extensão externa é executada como um processo separado no ambiente de execução. A extensão externa é inicializada antes que a função seja invocada, é executada em paralelo com o tempo de execução da função e continua a ser executada após a conclusão da invocação da função.

Para ter mais informações, consulte Ampliar funções do Lambda usando extensões do Lambda.

Simultaneidade

Simultaneidade é o número de solicitações que a função atende a cada momento. Quando a função é invocada, o Lambda provisiona uma instância se ela processar o evento. Quando a execução do código da função terminar, ela poderá processar outra solicitação. Se a função for invocada novamente enquanto uma solicitação ainda estiver sendo processada, outra instância será provisionada, aumentando a simultaneidade da função.

A simultaneidade está sujeita a cotas no nível da região da AWS. É possível configurar funções individuais limitando sua simultaneidade ou permitindo que elas alcancem um nível específico de simultaneidade. Para ter mais informações, consulte Configurar a simultaneidade reservada para uma função.

Qualifier

Ao chamar ou visualizar uma função, é possível incluir um qualificador para especificar uma versão ou um alias. Uma versão é um snapshot imutável da configuração e do código de uma função que tem um qualificador numérico. Por exemplo, `my-function:1`. Um alias é um ponteiro para uma versão que pode ser atualizado para mapear para uma versão diferente ou dividir o tráfego entre duas versões. Por exemplo, `my-function:BLUE`. É possível usar versões e aliases em conjunto para fornecer uma interface estável para os clientes chamarem sua função.

Para ter mais informações, consulte Versões da função do Lambda.

Destino

Adestinationé umAWSonde o Lambda pode enviar eventos de uma invocação assíncrona. É possível configurar um destino para eventos que tiveram falha no processamento. Alguns serviços também oferecem suporte a um destino para eventos que são processados com êxito.

Para ter mais informações, consulte Configurar destinos para invocação assíncrona.

Modelo de programação do Lambda

O Lambda oferece um modelo de programação comum a todos os runtimes. O modelo de programação define a interface entre seu código e o sistema Lambda. Você diz ao Lambda o ponto de entrada para sua função definindo um manipulador na configuração da função. O runtime transmite ao handler os objetos que contêm o evento de invocação e o contexto, como o nome da função e o ID da solicitação.

Quando o handler termina de processar o primeiro evento, o runtime o envia outro. A classe da função permanece na memória, de forma que é possível reutilizar clientes e variáveis que são declarados fora do método do handler no código de inicialização. Para economizar tempo de processamento em eventos subsequentes, crie recursos reutilizáveis como clientes do AWS SDK durante a inicialização. Uma vez inicializada, cada instância da função pode processar milhares de solicitações.

A função também tem acesso ao armazenamento local no diretório `/tmp`. O conteúdo do diretório permanece quando o ambiente de execução é congelado, fornecendo cache transitório que pode ser usado para várias invocações. Para obter mais informações, consulte Ambiente de execução do Lambda.

Quando o rastreamento do AWS X-Ray está habilitado, o runtime registra subsegmentos separados para inicialização e execução.

O tempo de execução captura a saída de registro da sua função e a envia para o Amazon CloudWatch Logs. Além de registrar em log a saída da função, o runtime também registra em log as entradas quando a invocação da função é iniciada e termina. Isso inclui um log de relatório com o ID da solicitação, a duração faturada, a duração da inicialização e outros detalhes. Se a função lançar um erro, o runtime retornará esse erro para o chamador.

> **Note**
>
> O registro está sujeito às cotas de CloudWatch registros. Os dados de log podem ser perdidos devido a limitação ou, em certos casos, quando uma instância da sua função é interrompida.

O Lambda dimensiona a função executando instâncias adicionais dela à medida que a demanda aumenta, e interrompendo instâncias à medida que a demanda diminui. Esse modelo leva a variações na arquitetura da aplicação, como:

- Salvo indicação em contrário, as solicitações de entrada poderiam ser processadas fora de ordem ou simultaneamente.
- Não dependa de que instâncias da sua função sejam de longa duração. Em vez disso, armazene o estado da sua aplicação em algum outro local.
- Use o armazenamento local e os objetos do nível de classe para aumentar a performance, mas mantenha no mínimo o tamanho do pacote de implantação e da quantidade de dados transferidos para o ambiente de execução.

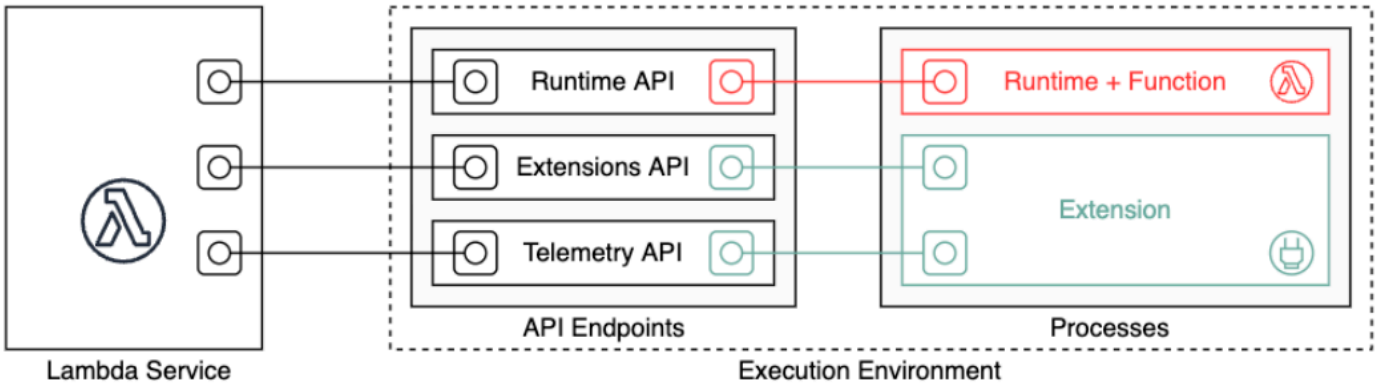
Para obter uma introdução prática do modelo de programação na linguagem de programação de sua preferência, consulte os capítulos a seguir.

- Criar funções do Lambda com Node.js
- Criar funções do Lambda com Python
- Construir funções do Lambda com Ruby
- Construir funções do Lambda com Java
- Criar funções do Lambda com Go
- Construir funções do Lambda com C#
- Construir funções do Lambda com o PowerShell

Ambiente de execução do Lambda

O Lambda invoca a função em um ambiente de execução, que fornece um ambiente do runtime seguro e isolado. O ambiente de execução gerencia os recursos necessários para executar a função. O ambiente de execução também fornece suporte ao ciclo de vida para o runtime da função e qualquer extensão externa associada à função.

O runtime da função se comunica com o Lambda usando a API de runtime. As extensões se comunicam com o Lambda usando a API Extensions. As extensões também podem receber mensagens de log e outras telemetrias da função ao usar a API de telemetria.



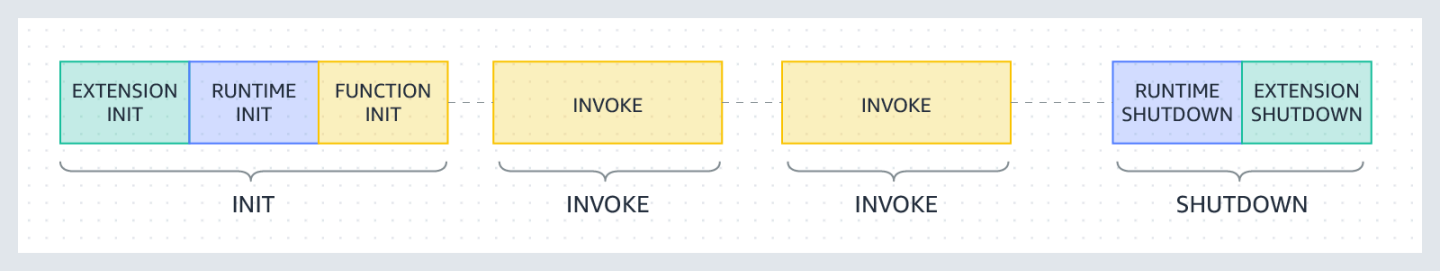
Quando você cria uma função do Lambda, você especifica informações de configuração, como a quantidade de memória disponível e o tempo máximo de execução permitido para sua função. O Lambda usa essas informações para configurar o ambiente de execução.

O runtime da função e cada extensão externa são processos executados dentro do ambiente de execução. Permissões, recursos, credenciais e variáveis de ambiente são compartilhados entre a função e as extensões.

Tópicos

- Ciclo de vida do ambiente de execução do Lambda
- Como implementar a ausência de estado em funções

Ciclo de vida do ambiente de execução do Lambda



Cada fase começa com um evento que o Lambda envia para o runtime e para todas as extensões registradas. O runtime e cada extensão indicam a conclusão enviando uma solicitação de API `Next`. O Lambda congela o ambiente de execução quando o runtime e cada extensão tiverem sido concluídos e não houver eventos pendentes.

Tópicos

Fase de inicialização

Na fase `Init`, o Lambda executa três tarefas:

- Iniciar todas as extensões (`Extension init`)
- Realizar bootstrap no runtime (`Runtime init`)
- Executar o código estático da função (`Function init`)
- Execute qualquer hook de runtime `beforeCheckpoint` (somente para Lambda SnapStart)

A fase `Init` termina quando o runtime e todas as extensões sinalizam que estão prontas enviando uma solicitação de API `Next`. A fase `Init` é limitada a 10 segundos. Se todas as três tarefas não forem concluídas em até dez segundos, o Lambda repetirá a fase `Init` no momento da primeira invocação de função com o tempo-limite de função configurado.

Quando o Lambda SnapStart está ativado, a fase `Init` ocorre ao você publicar uma versão da função. O Lambda salva um snapshot do estado da memória e do disco do ambiente de execução inicializado, mantém o snapshot criptografado e o armazena em cache para acesso de baixa latência. Se você tiver um hook de runtime `beforeCheckpoint`, então, o código será executado ao final da fase `Init`.

> **Note**
>
> O tempo limite de dez segundos não se aplica às funções que estão usando concorrência provisionada ou SnapStart. Para funções de concorrência provisionada e SnapStart, seu código de inicialização pode ser executado por até 15 minutos. O limite de tempo é de 130 segundos ou o tempo limite da função configurada (máximo de 900 segundos), o que for maior.

Quando você usa simultaneidade provisionada, o Lambda inicializa o ambiente de execução quando você define as configurações do PC para uma função. O Lambda também garante que os ambientes de execução inicializados estejam sempre disponíveis antes das invocações. Você pode observar lacunas entre a invocação e as fases de inicialização da sua função. Dependendo do runtime e da configuração de memória da sua função, é possível observar uma latência variável na primeira invocação em um ambiente de execução inicializado.

Para funções que usam simultaneidade sob demanda, o Lambda pode ocasionalmente inicializar ambientes de execução antes das solicitações de invocação. Quando isso acontece, você também pode observar uma lacuna de tempo entre as fases de inicialização e invocação da função. Recomendamos que você não dependa desse comportamento.

Falhas durante a fase de inicialização

Se uma função falhar ou atingir o tempo limite durante a fase `Init`, o Lambda emite informações de erro no log `INIT_REPORT`.

Example — log INIT_REPORT para tempo limite

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

Example — log INIT_REPORT para falha de extensão

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

Se a fase `Init` for bem-sucedida, o Lambda não emitirá o log `INIT_REPORT`, a menos que o SnapStart seja ativado. As funções do SnapStart sempre emitem `INIT_REPORT`. Para ter mais informações, consulte Monitoramento para Lambda SnapStart.

Fase de restauração (somente para Lambda SnapStart)

Quando você invoca uma função do SnapStart pela primeira vez e à medida que aumenta a escala verticalmente da função, o Lambda retoma novos ambientes de execução a partir do snapshot retido, em vez de inicializar a função do zero. Se você tiver um hook de runtime `afterRestore()`, o código será executado ao final da fase `Restore`. Você terá cobranças pela duração dos hooks de runtime `afterRestore()`. O runtime (JVM) deve ser carregado e os hooks de runtime `afterRestore()` devem ser concluídos dentro do limite de tempo limite (dez segundos). Caso contrário, você obterá uma SnapStartTimeoutException. Quando a fase `Restore` é concluída, o Lambda invoca o manipulador de função (a Fase de invocação).

Falhas durante a fase de restauração

Se a fase `Restore` falhar, o Lambda emite informações de erro no log `RESTORE_REPORT`.

Example — log RESTORE_REPORT para tempo limite

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

Example — log RESTORE_REPORT para falha no hook de runtime

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

Para obter mais informações sobre o log `RESTORE_REPORT`, consulte Monitoramento para Lambda SnapStart.

Fase de invocação

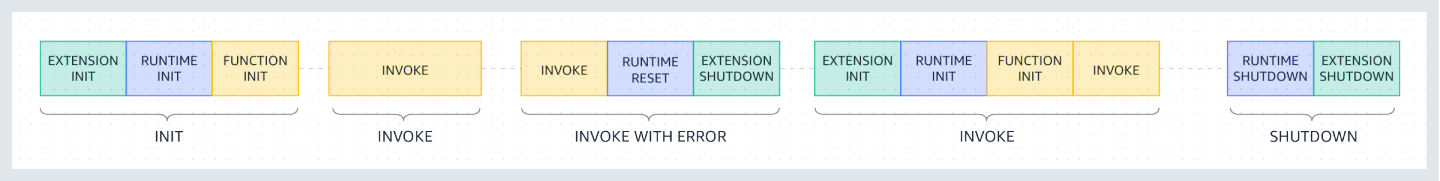
Quando uma função do Lambda é invocada em resposta a uma solicitação de API `Next`, o Lambda envia um evento `Invoke` para o runtime e para cada extensão.

A configuração de tempo limite da função limita a duração de toda a fase `Invoke`. Por exemplo, se você definir o tempo limite da função como 360 segundos, a função e todas as extensões precisam ser concluídas em até 360 segundos. Observe que não há fase de pós-invocação independente. A duração é a soma de todo o tempo de invocação (runtime + extensões) e não é calculada até que a função e todas as extensões tenham terminado a execução.

A fase de invocação termina após o runtime e todas as extensões sinalizam que elas foram concluídas enviando uma solicitação de API `Next`.

Falhas durante a fase de invocação

Se a função do Lambda falhar ou expirar durante o`Invoke`fase, o Lambda redefine o ambiente de execução. O diagrama a seguir ilustra o comportamento do ambiente de execução do Lambda quando há falha de invocação:



No diagrama anterior:

- A primeira fase é a fase INIT, que é executada sem erros.
- A segunda fase é a fase INVOKE, que é executada sem erros.
- Em algum momento, suponha que a função tenha uma falha de invocação durante a execução (como erro de tempo limite ou de runtime da função). A terceira fase, chamada INVOKE WITH ERROR, ilustra esse caso. Quando isso ocorre, o serviço do Lambda executa uma redefinição. A redefinição se comporta como um evento `Shutdown`. Primeiro, o Lambda encerra o runtime; depois, envia um evento `Shutdown` para cada extensão externa registrada. O evento inclui o motivo do desligamento. Se esse ambiente for usado para uma nova invocação, o Lambda reinicializará a extensão e o runtime com a próxima invocação.

  > **Note**
  >
  > Redefinir o Lambda não limpa o conteúdo do diretório `/tmp` antes da próxima fase de inicialização. Esse comportamento é consistente com a fase de desligamento regular.

- A quarta fase representa a fase INVOKE imediatamente após uma falha de invocação. Aqui, o Lambda reinicializa o ambiente executando a fase INIT. Isso se chama inicialização suprimida.

Quando ocorrem inicializações suprimidas, o Lambda não relata explicitamente uma fase adicional de INIT no CloudWatch Logs. Em vez disso, você perceberá que a duração na linha REPORT inclui uma duração adicional de INIT mais a duração de INVOKE. Por exemplo, digamos que você veja os seguintes logs no 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
```

Neste exemplo, a diferença entre os carimbos de data e hora REPORT e START é de 2,5 segundos. Isso não corresponde à duração relatada de 3022,91 milissegundos, pois não leva em conta o INIT adicional (inicialização suprimida) que o Lambda executou. Neste exemplo, podemos inferir que a fase INVOKE exata durou 2,5 segundos.

Para obter mais insights sobre esse comportamento, você pode usar API de Telemetria do Lambda. A API Telemetry emite eventos `INIT_START`, `INIT_RUNTIME_DONE` e `INIT_REPORT` com `phase=invoke` sempre que ocorrem inicializações suprimidas entre as fases de invocação.

- A quinta fase representa a fase SHUTDOWN, que é executada sem erros.

Fase de desligamento

Quando o Lambda estiver prestes a encerrar o runtime, ele enviará um evento `Shutdown` a cada extensão externa registrada. As extensões podem usar esse tempo para tarefas de limpeza finais. O evento `Shutdown` é uma resposta a uma solicitação de API `Next`.

Duração: toda a fase `Shutdown` é limitada a 2 segundos. Se o runtime ou qualquer extensão não responder, o Lambda o encerrará por meio de um sinal (`SIGKILL`).

Após a conclusão da função e de todas as extensões, o Lambda mantém o ambiente de execução por algum tempo à espera de uma outra invocação de função. No entanto, o Lambda encerra os ambientes de execução a cada poucas horas para permitir atualizações e manutenção do runtime, mesmo para funções que são invocadas continuamente. Você não deve presumir que o ambiente de execução persistirá indefinidamente. Para ter mais informações, consulte Como implementar a ausência de estado em funções.

Quando a função é invocada novamente, o Lambda descongela o ambiente para reutilização. Reutilizar o ambiente de execução tem as seguintes implicações:

- Os objetos declarados fora do método do manipulador da função permanecem inicializados, fornecendo otimização adicional quando a função é invocada novamente. Por exemplo, se sua função do Lambda estabelecer uma conexão com o banco de dados, em vez de restabelecer a conexão, a conexão original é usada em invocações subsequentes. Recomendamos que você adicione lógica em seu código para verificar se uma há conexão existente antes de criar outra.
- Cada ambiente de execução fornece entre 512 MB e 10.240 MB, em incrementos de 1 MB, de espaço em disco no diretório `/tmp`. O conteúdo do diretório permanece quando o ambiente de execução é congelado, fornecendo cache transitório que pode ser usado para várias invocações. Você pode adicionar um código extra para verificar se o cache tem os dados que você armazenou. Para obter mais informações sobre limites de tamanho de implantação, consulte Cotas Lambda.
- Processos em segundo plano ou retornos de chamada que foram iniciados pela função do Lambda e não foram concluídos quando a função terminou serão retomados se o Lambda reutilizar o ambiente de execução. Garanta que todos os processos em segundo plano ou retornos de chamadas no seu código sejam concluídos antes que o código seja encerrado.

Como implementar a ausência de estado em funções

Ao escrever seu código de função do Lambda, trate o ambiente de execução como sem estado, supondo que ele exista apenas para uma única invocação. O Lambda encerra os ambientes de execução a cada poucas horas para permitir atualizações e manutenção do runtime, mesmo para funções que são invocadas continuamente. Inicialize qualquer estado necessário (por exemplo, buscar um carrinho de compras em uma tabela do Amazon DynamoDB) quando sua função começar. Antes de sair, confirme as alterações permanentes de dados em armazenamentos duráveis, como Amazon Simple Storage Service (Amazon S3), DynamoDB ou Amazon Simple Queue Service (Amazon SQS). Evite depender de estruturas de dados existentes, arquivos temporários ou estados que abranjam invocações, como contadores ou agregados. Isso garante que sua função manipule cada invocação de forma independente.

Pacotes de implantação do Lambda

O código da função do AWS Lambda consiste em scripts ou programas compilados e as dependências deles. Você usa umpacote de implantaçãopara implantar seu código de função no Lambda. O Lambda é compatível com dois tipos de pacotes de implantação: imagens de contêiner e arquivos .zip.

Tópicos

Imagens de contêiner

Uma imagem de contêiner inclui o sistema operacional de base, o runtime, as extensões do Lambda, o código da aplicação e dependências dele. Você também pode adicionar dados estáticos, como modelos de machine learning, à imagem.

O Lambda oferece um conjunto de imagens de base de código aberto que você pode usar para criar a imagem de contêiner. Para criar e testar imagens de contêiner, você pode usar a interface da linha de comando AWS Serverless Application Model (AWS SAM) (CLI) ou ferramentas nativas de contêiner, como a CLI do Docker.

Faça upload de suas imagens de contêiner para o Amazon Elastic Container Registry (Amazon ECR), um serviço gerenciado de registro de imagens de contêiner da AWS. Para implantar a imagem em sua função, especifique o URL da imagem do Amazon ECR usando o console do Lambda, a API do Lambda, as ferramentas da linha de comando ou os AWS SDKs.

Para obter mais informações sobre imagens de contêiner do Lambda, consulte Criar uma função do Lambda usando uma imagem de contêiner.

Arquivos .zip

Um arquivo .zip inclui o código da aplicação e as dependências dele. Quando você cria funções usando o console do Lambda ou um toolkit, o Lambda cria automaticamente um arquivo .zip do seu código.

Quando você cria funções com a API do Lambda, ferramentas da linha de comando ou AWS SDKs, é necessário criar o pacote de implantação. Também será necessário criar um pacote de implantação se sua função usar uma linguagem compilada ou para adicionar dependências a ela. Para implantar o código da sua função, faça upload do pacote de implantação do Amazon Simple Storage Service (Amazon S3) ou de sua máquina local.

Você pode fazer o upload de um arquivo .zip como seu pacote de implantação usando o console do Lambda,AWS Command Line Interface(AWS CLI) ou para um bucket do Amazon Simple Storage Service (Amazon S3).

Usar o console do Lambda

As etapas a seguir demonstram como fazer o upload de um arquivo .zip como seu pacote de implantação usando o console do Lambda.

Para fazer o upload de um arquivo .zip no console do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione uma função.
3. NoCódigo-fonte, selecioneFazer upload dee, em seguida.zip.
4. Escolha Upload (Carregar) para escolher seu arquivo .zip local.
5. Escolha Salvar.

Usando a AWS CLI

Você pode fazer o upload de um arquivo .zip como seu pacote de implantação usando o AWS Command Line Interface (AWS CLI). Para obter instruções específicas de linguagem, consulte os tópicos a seguir.

Node.js

- Implantar funções do Lambda em Node.js com arquivos .zip

Python

- Trabalhar com arquivos .zip para funções do Lambda em Python

Ruby

- Como trabalhar com arquivos .zip para funções do Lambda em Ruby

Java

- Implantar funções do Lambda em Java com arquivos .zip ou JAR

Go

- Implantar funções do Lambda em Go com arquivos .zip

C#

- Criar e implantar funções do Lambda em C# com arquivos .zip

PowerShell

- Implemente funções PowerShell Lambda com arquivos de arquivos.zip

Uso do Amazon S3

Você pode fazer o upload de um arquivo .zip como seu pacote de implantação usando o Amazon Simple Storage Service (Amazon S3). Para ter mais informações, consulte .

Camadas

Se você implantar seu código de função usando um arquivo .zip, poderá usar camadas do Lambda como mecanismo de distribuição para bibliotecas, tempos de execução personalizados e outras dependências de função. As camadas permitem que você gerencie seu código de função no desenvolvimento de modo independente do código inalterável e dos recursos que ele usa. Você pode configurar a função para usar camadas que você criar, camadas fornecidas pela AWS ou camadas de outros clientes da AWS.

Não é possível usar camadas com imagens de contêiner. Em vez disso, empacote seu runtime, bibliotecas e outras dependências de sua preferência na imagem do contêiner ao compilar a imagem.

Para obter mais informações sobre camadas, consulte Camadas do Lambda.

Usando outros serviços do AWS para criar um pacote de implantação

A seção a seguir descreve outros serviços do AWS que você pode usar para empacotar dependências para a função do seu Lambda.

Pacotes de implantação com bibliotecas C ou C++

Se o pacote de implantação contiver bibliotecas nativas, você poderá criar o pacote de implantação com o AWS Serverless Application Model (AWS SAM). É possível usar o comando `sam build` da CLI do AWS SAM com `--use-container` para criar seu pacote de implantação. Essa opção cria um pacote de implantação dentro de uma imagem do Docker compatível com o ambiente de execução do Lambda.

Para obter mais informações, consulte sam build no Guia do desenvolvedor do AWS Serverless Application Model.

Pacotes de implantação com mais de 50 MB

Se o pacote de implantação for maior que 50 MB, carregue o código de função e as dependências para um bucket do Amazon S3.

Você pode criar um pacote de implantação e fazer upload do arquivo .zip para seu bucket do Amazon S3 na região da AWS em que você quer criar uma função do Lambda. Ao criar sua função do Lambda, especifique o nome do bucket do S3 e o nome da chave do objeto no console do Lambda ou usando oAWS CLI.

Para criar um bucket usando o console do Amazon S3, consulte Como criar um bucket no Guia do usuário do console do Amazon Simple Storage Service.

Usar o Lambda com a infraestrutura como código (IaC)

O Lambda oferece várias maneiras de implantar códigos e criar funções. Por exemplo, você pode usar o console do Lambda ou a AWS Command Line Interface (AWS CLI) para criar ou atualizar manualmente as funções do Lambda. Além dessas opções manuais, a AWS oferece várias soluções para implantar funções do Lambda e aplicações sem servidor usando a infraestrutura como código (IaC). Com a IaC, você pode provisionar e manter funções do Lambda e outros recursos da AWS usando código, em vez de processos e configurações manuais.

Na maioria das vezes, as funções do Lambda não são executadas isoladamente. Em vez disso, elas fazem parte de uma aplicação sem servidor com outros recursos, como bancos de dados, filas e armazenamento. Com a IaC, você pode automatizar os processos de implantação para implantar e atualizar de forma rápida e repetitiva aplicações inteiras sem servidor associadas a vários recursos da AWS separados. Essa abordagem agiliza o ciclo de desenvolvimento, torna o gerenciamento de configurações mais fácil e garante que seus recursos sejam implantados sempre da mesma forma.

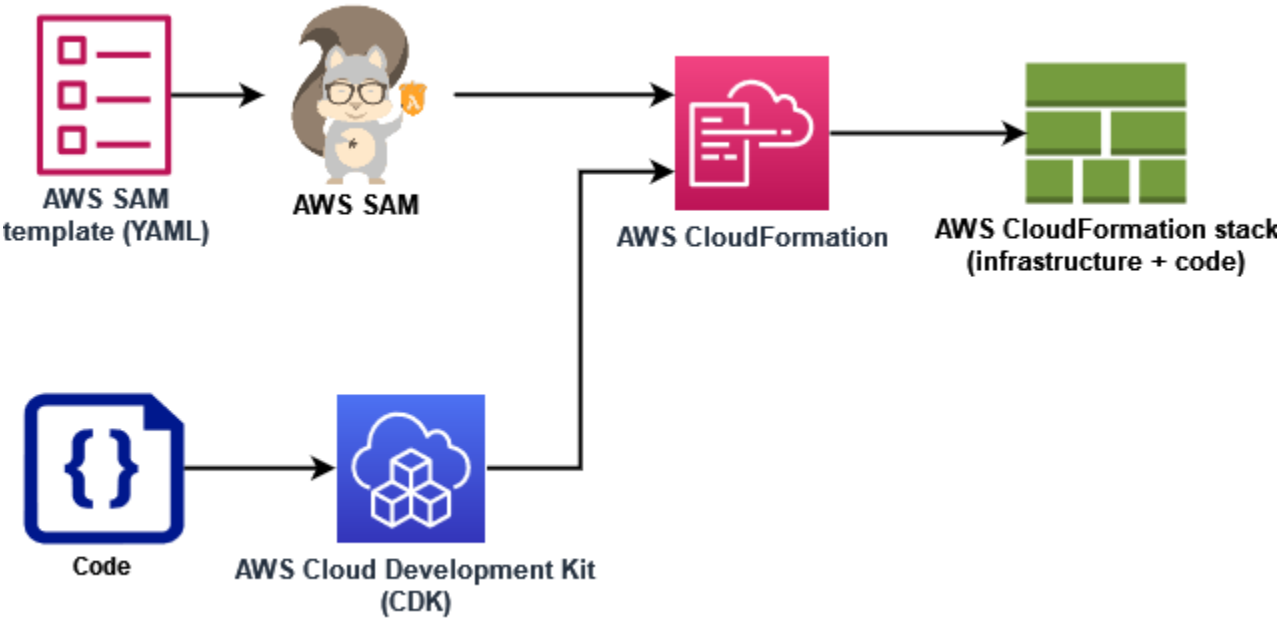
Tópicos

Ferramentas de IaC para o Lambda

Para implantar funções do Lambda e aplicações sem servidor usando a IaC, a AWS oferece várias outras ferramentas e serviços.

O AWS CloudFormation foi o primeiro serviço oferecido pela AWS para criar e configurar recursos de nuvem. Com o AWS CloudFormation, você cria modelos de texto para definir infraestruturas e códigos. Com a introdução de novos serviços pela AWS e o aumento da complexidade da criação de modelos para o AWS CloudFormation, outras duas ferramentas foram lançadas. O AWS SAM é outra estrutura baseada em modelos para definir aplicações sem servidor. O AWS Cloud Development Kit (AWS CDK) é uma abordagem que prioriza o código para definir e provisionar a infraestrutura usando estruturas de código em várias linguagens de programação populares.

Tanto com o AWS SAM quanto com o AWS CDK, o AWS CloudFormation opera em segundo plano para criar e implantar sua infraestrutura. O diagrama a seguir ilustra a relação entre essas ferramentas, e os parágrafos abaixo do diagrama explicam suas principais características.



- AWS CloudFormation- Com CloudFormation você modela e configura seus AWS recursos usando um modelo YAML ou JSON que descreve seus recursos e suas propriedades. CloudFormation provisiona seus recursos de forma segura e repetível, permitindo que você construa com frequência sua infraestrutura e aplicativos sem etapas manuais. Quando você altera a configuração, CloudFormation determina as operações corretas a serem executadas para atualizar sua pilha. CloudFormation pode até mesmo reverter as alterações.
- AWS Serverless Application Model (AWS SAM): o AWS SAM é uma estrutura de código aberto para a definição de aplicações sem servidor. Os modelos do AWS SAM usam uma sintaxe abreviada para definir funções, APIs, bancos de dados e mapeamentos da origem do evento com apenas algumas linhas de texto (YAML) por recurso. Durante a implantação, o AWS SAM transforma e expande a sintaxe do AWS SAM em sintaxe do AWS CloudFormation. Por esse motivo, qualquer CloudFormation sintaxe pode ser adicionada aos AWS SAM modelos. Isso oferece AWS SAM todo o poder do CloudFormation, mas com menos linhas de configuração.
- AWS Cloud Development Kit (AWS CDK)- Com oAWS CDK, você define sua infraestrutura usando construções de código e a provisiona por meio AWS CloudFormation dela. AWS CDKpermite modelar a infraestrutura de aplicativos com Python TypeScript, Java, .NET e Go (no Developer Preview) usando seu IDE, ferramentas de teste e padrões de fluxo de trabalho existentes. Você

tem acesso a todos os benefícios do AWS CloudFormation, incluindo implantação repetível, fácil reversão e detecção de desvios.

A AWS também fornece um serviço denominado AWS Application Composer para desenvolver modelos de IaC usando uma interface gráfica simples. Com o Application Composer, você projeta uma arquitetura de aplicações arrastando, agrupando e conectando os Serviços da AWS em uma tela visual. Em seguida, o Application Composer cria um modelo do AWS SAM ou do AWS CloudFormation com base no seu design que você pode usar para implantar aplicações.

Na seção the section called “Conceitos básicos de IaC para o Lambda” abaixo, o Application Composer é usado para desenvolver um modelo para uma aplicação sem servidor baseada em uma função do Lambda existente.

Conceitos básicos de IaC para o Lambda

Neste tutorial, você pode começar a usar a IaC com o Lambda elaborando um modelo do AWS SAM com base em uma função do Lambda existente e, em seguida, criando uma aplicação sem servidor no Application Composer adicionando outros recursos da AWS.

Se preferir começar com um tutorial do AWS CloudFormation ou do AWS SAM para aprender a trabalhar com modelos sem usar o Application Composer, você encontrará links para outros recursos na seção the section called “Próximas etapas” no final desta página.

Ao realizar este tutorial, você aprenderá alguns conceitos fundamentais, como a forma que os recursos da AWS são especificados no AWS SAM. Você também aprenderá a usar o Application Composer para criar uma aplicação sem servidor que você pode implantar usando o AWS SAM ou o AWS CloudFormation.

Para concluir o tutorial, você executará as seguintes tarefas:

- Criar exemplo de função do Lambda
- Usar o console do Lambda para visualizar o modelo do AWS SAM para a função
- Exportar a configuração da função para o AWS Application Composer e criar uma aplicação simples sem servidor com base na configuração da função
- Salvar um modelo do AWS SAM atualizado que você possa usar como base para implantar sua aplicação sem servidor

Na seção the section called “Próximas etapas”, você encontrará recursos que podem ser usados para aprender mais sobre o AWS SAM e o Application Composer. Esses recursos incluem links para tutoriais mais avançados que ensinam como implantar uma aplicação sem servidor usando o AWS SAM.

Pré-requisitos

Neste tutorial, o recurso de sincronização local do Application Composer é usado para salvar os arquivos de modelo e código na sua máquina de compilação local. Para usar esse recurso, você precisa de um navegador compatível com a API File System Access, que permite a uma aplicação da Web ler, gravar e salvar arquivos no sistema de arquivos local. Recomendamos usar o Google Chrome ou o Microsoft Edge. Para obter mais informações sobre a API File System Access, consulte What is the File System Access API?

Criar uma função do Lambda

Nessa primeira etapa, você criará uma função do Lambda que poderá ser usada para concluir o restante do tutorial. Para simplificar, use o console do Lambda para criar uma função básica “Hello world” com o runtime do Python 3.11.

Para criar uma função do Lambda “Hello world” com o console:

1. Abra o console do lambda.
2. Escolha a opção Criar função.
3. Deixe a opção Criar do zero selecionada e, em Informações básicas, insira **`LambdaIaCDemo`** em Nome da função.
4. Em Runtime, escolha Python 3.11.
5. Escolha a opção Criar função.

Ver o modelo do AWS SAM da sua função

Antes de exportar a configuração da função para o Application Composer, use o console do Lambda para ver a configuração atual da função como um modelo do AWS SAM. Seguindo as etapas desta seção, você aprenderá sobre a anatomia de um modelo do AWS SAM e como definir recursos, como funções do Lambda, para começar a especificar uma aplicação sem servidor.

Ver o modelo do AWS SAM da função

1. Abra a página Funções do console do Lambda.
2. Escolha a função que você acabou de criar (`LambdaIaCDemo`).
3. No painel Visão geral da função, escolha Modelo.

   No lugar do diagrama que representa a configuração da sua função, você verá um modelo do AWS SAM para sua função. O modelo será algo assim:

```
# 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:*
```

Vamos analisar o modelo YAML da sua função e entender alguns conceitos-chave.

O modelo começa com a declaração `Transform: AWS::Serverless-2016-10-31`. Ela é necessária porque os modelos do AWS SAM são implantados em segundo plano por meio do AWS CloudFormation. A declaração `Transform` identifica o modelo como um arquivo de modelo do AWS SAM.

Após a declaração `Transform`, temos a seção `Resources`. Nela, são definidos os recursos da AWS que você deseja implantar com o modelo do AWS SAM. Os modelos do AWS SAM podem conter uma combinação de recursos do AWS SAM e do AWS CloudFormation. Isso ocorre porque, durante a implantação, os modelos do AWS SAM se expandem para modelos do AWS CloudFormation; portanto, qualquer sintaxe válida do AWS CloudFormation pode ser adicionada a um modelo do AWS SAM.

No momento, só há um recurso definido na seção `Resources` do modelo, sua função do Lambda `LambdaIaCDemo`. Para adicionar uma função do Lambda a um modelo do AWS SAM, use o tipo de recurso `AWS::Serverless::Function`. O recurso `Properties` de uma função do Lambda define o runtime da função, o manipulador da função e outras opções de configuração. O caminho para o código-fonte da função que o AWS SAM deve usar para implantar a função também é definido aqui. Para saber mais sobre os recursos da função Lambda emAWS SAM, consulte o AWS::Serverless::FunctionGuia do AWS SAMdesenvolvedor.

Além das propriedades e configurações da função, o modelo também especifica uma política do AWS Identity and Access Management (IAM) para a função. Essa política dá permissão à sua função para gravar registros no Amazon CloudWatch Logs. Quando você cria uma função no console do Lambda, ele anexa automaticamente essa política à sua função. Para saber mais sobre como

especificar uma política do IAM para uma função em um AWS SAM modelo, consulte a `policies` propriedade na AWS::Serverless::Functionpágina do Guia do AWS SAM desenvolvedor.

Para saber mais sobre a estrutura de modelos do AWS SAM, consulte AWS SAM template anatomy.

Usar o AWS Application Composer para criar uma aplicação sem servidor

Para começar a criar uma aplicação simples sem servidor usando o modelo do AWS SAM da função como ponto de partida, exporte a configuração da função para o Application Composer e ative o modo de sincronização local dele. A sincronização local salva automaticamente o código da função e o modelo do AWS SAM na máquina de compilação local e mantém o modelo salvo sincronizado à medida que você adiciona outros recursos da AWS no Application Composer.

Exportar a função para o Application Composer

1. No painel Visão geral da função, escolha Exportar para o Application Composer.

   Para exportar a configuração e o código da função para o Application Composer, o Lambda cria um bucket do Amazon S3 na sua conta para armazenar esses dados temporariamente.
2. Na caixa de diálogo, escolha Confirmar e criar projeto para aceitar o nome padrão desse bucket e exportar a configuração e o código da função para o Application Composer.
3. (Opcional) Para escolher outro nome para o bucket do Amazon S3 criado pelo Lambda, insira um novo nome e escolha Confirmar e criar projeto. Os nomes de buckets do Amazon S3 devem ser exclusivos no mundo todo e seguir as regras de nomenclatura de buckets.

   Ao selecionar Confirmar e criar projeto, o console do Application Composer é aberto. Na tela, você verá sua função do Lambda.
4. No Menu suspenso, escolha Ativar sincronização local.
5. Na caixa de diálogo exibida, escolha Selecionar pasta e selecione uma pasta na sua máquina de compilação local.
6. Escolha Ativar para ativar a sincronização local.

Para exportar a função para o Application Composer, você precisa de permissão para usar determinadas ações de API. Se não conseguir exportar sua função, consulte the section called “Permissões obrigatórias” e verifique se você tem as permissões necessárias.

> **Note**
>
> O preço padrão do Amazon S3 se aplica ao bucket criado pelo Lambda quando você exporta uma função para o Application Composer. Os objetos que o Lambda coloca no bucket são excluídos automaticamente após dez dias, mas o Lambda não exclui o bucket em si. Para evitar que cobranças adicionais sejam adicionadas à sua Conta da AWS, siga as instruções em Excluir um bucket depois de exportar a função para o Application Composer. Para obter mais informações sobre o bucket do Amazon S3 criado pelo Lambda, consulte the section called “Application Composer”.

Projetar a aplicação sem servidor no Application Composer

Depois de ativar a sincronização local, as alterações feitas no Application Composer serão refletidas no modelo do AWS SAM salvo na sua máquina de compilação local. Agora você pode arrastar e soltar recursos adicionais da AWS na tela do Application Composer para criar sua aplicação. Neste exemplo, você adiciona uma fila simples do Amazon SQS como gatilho para sua função do Lambda e uma tabela do DynamoDB na qual a função gravará os dados.

1. Para adicionar um gatilho do Amazon SQS à função do Lambda, faça o seguinte:
   a. No campo de pesquisa na paleta Recursos, insira **`SQS`**.
   b. Arraste o recurso Fila do SQS para sua tela e posicione-o à esquerda da função do Lambda.
   c. Escolha Detalhes e, em ID lógico, insira **`LambdaIaCQueue`**.
   d. Escolha Salvar.
   e. Conecte seus recursos do Amazon SQS e do Lambda clicando na porta Assinatura no cartão de fila do SQS e arrastando-a para a porta à esquerda no cartão da função do Lambda. Se aparecer uma linha entre os dois recursos, é sinal de que a conexão teve êxito. O Application Composer também exibe uma mensagem na parte inferior da tela indicando que os dois recursos foram conectados com êxito.
2. Para adicionar uma tabela do Amazon DynamoDB na qual a função do Lambda gravará os dados, faça o seguinte:
   a. No campo de pesquisa na paleta Recursos, insira **`DynamoDB`**.
   b. Arraste o recurso Tabela do DynamoDB para sua tela e posicione-o à direita da função do Lambda.

c. Escolha Detalhes e, em ID lógico, insira **`LambdaIaCTable`**.

d. Escolha Salvar.

e. Conecte a tabela do DynamoDB à sua função do Lambda clicando na porta à direita do cartão da função do Lambda e arrastando-a até a porta à esquerda do cartão do DynamoDB.

Agora que esses recursos extras foram adicionados, vamos dar uma olhada no modelo do AWS SAM atualizado que foi criado pelo Application Composer.

Ver o modelo do AWS SAM atualizado

- Na tela do Application Composer, escolha Modelo para alternar da visualização da tela para a visualização do modelo.

Agora, seu modelo do AWS SAM deve conter os seguintes recursos e propriedades adicionais:

- Uma fila do Amazon SQS com o identificador `LambdaIaCQueue`

```
LambdaIaCQueue:
    Type: AWS::SQS::Queue
    Properties:
      MessageRetentionPeriod: 345600
```

  Quando você adiciona uma fila do Amazon SQS usando o Application Composer, ele define a propriedade `MessageRetentionPeriod`. Você também pode definir a propriedade `FifoQueue` selecionando Detalhes no cartão da fila do SQS e marcando ou desmarcando Fila FIFO.

  Para definir outras propriedades para sua fila, você pode adicioná-las manualmente ao modelo. Para saber mais sobre o recurso `AWS::SQS::Queue` e suas propriedades disponíveis, consulte AWS::SQS::Queue no Guia do usuário do AWS CloudFormation.

- Uma propriedade `Events` na definição da função do Lambda que especifica a fila do Amazon SQS usada como gatilho para a função.

```
Events:
  LambdaIaCQueue:
    Type: SQS
    Properties:
      Queue: !GetAtt LambdaIaCQueue.Arn
```

```
      BatchSize: 1
```

A propriedade `Events` consiste em um tipo de evento e um conjunto de propriedades que dependem desse tipo. Para saber mais sobre as diferentes configurações Serviços da AWS que você pode configurar para acionar uma função Lambda e as propriedades que você pode definir, consulte o EventSourceGuia do AWS SAMdesenvolvedor.

- Uma tabela do DynamoDB com o identificador `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
```

Ao adicionar uma tabela do DynamoDB usando o Application Composer, você pode definir as chaves da tabela escolhendo Detalhes no cartão da tabela do DynamoDB e editando os valores das chaves. O Application Composer também define valores padrão para várias outras propriedades, incluindo `BillingMode` e `StreamViewType`.

Para saber mais sobre essas propriedades e outras que você pode adicionar ao modelo do AWS SAM, consulte AWS::DynamoDB::Table no Guia do usuário do AWS CloudFormation.

- Uma nova política do IAM que dá permissão à sua função para realizar operações CRUD na tabela do DynamoDB que você adicionou.

```
Policies:
...
  - DynamoDBCrudPolicy:
    TableName: !Ref LambdaIaCTable
```

O modelo completo do AWS SAM resultante será algo assim:

```
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
```

Implantar a aplicação sem servidor usando o AWS SAM (opcional)

Se você quiser usar o AWS SAM para implantar uma aplicação sem servidor usando o modelo que acabou de criar no Application Composer, é necessário primeiro instalar a CLI do AWS SAM. Para isso, siga as instruções em Installing the AWS SAM CLI.

Antes de implantar a aplicação, você também precisa atualizar o código da função que o Application Composer salvou junto com o modelo. No momento, o arquivo `lambda_function.py` que o Application Composer salvou contém somente o código básico “Hello world” que o Lambda forneceu quando você criou a função.

Para atualizar o código da função, copie o código a seguir e cole-o no arquivo `lambda_function.py` que o Application Composer salvou na máquina de compilação local. Você especificou o diretório no qual o Application Composer deve salvar esse arquivo quando ativou o modo de sincronização local.

Esse código aceita um par de valores-chave em uma mensagem da fila do Amazon SQS que você criou no Application Composer. Se a chave e o valor forem strings, o código as usará para gravar um item na tabela do DynamoDB definida no modelo.

Código da função do Python atualizado

```
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
```

Implantar a aplicação sem servidor

Para implantar sua aplicação usando a CLI do AWS SAM, execute as etapas a seguir. Para que sua função seja compilada e implantada corretamente, a versão 3.11 do Python deve estar instalada na máquina de compilação e no `PATH`.

1. Execute o comando a seguir pelo diretório no qual o Application Composer salvou os arquivos `template.yaml` e `lambda_function.py`.

   ```
   sam build
   ```

   Esse comando reúne os artefatos de compilação da aplicação e os coloca no formato e no local adequados para implantá-los.
2. Para implantar a aplicação e criar os recursos do Lambda, do Amazon SQS e do DynamoDB especificados no modelo do AWS SAM, execute o comando a seguir.

   ```
   sam deploy --guided
   ```

   Usar o sinalizador `--guided` significa que o AWS SAM mostrará instruções para guiar você no processo de implantação. Para a implantação, aceite as opções padrão pressionando Enter.

Durante o processo de implantação, o AWS SAM cria os seguintes recursos na sua Conta da AWS:

- Uma pilha do AWS CloudFormation chamada `sam-app`
- Uma função do Lambda com o formato de nome `sam-app-LambdaIaCDemo-`*`99VXPpYQVv1M`*
- Uma fila do Amazon SQS com o formato do nome `sam-app-LambdaIaCQueue-`*`xL87VeKsGiIo`*
- Uma tabela do DynamoDB com o formato do nome `sam-app-LambdaIaCTable-`*`CN0S66C0VLNV`*

O AWS SAM também cria as funções e as políticas do IAM necessárias para que a função do Lambda possa ler mensagens da fila do Amazon SQS e realizar operações CRUD na tabela do DynamoDB.

Para saber mais sobre como usar o AWS SAM para implantar aplicações sem servidor, consulte os recursos na seção the section called “Próximas etapas”.

Testar a aplicação implantada (opcional)

Para confirmar que a aplicação sem servidor foi implantada corretamente, envie uma mensagem para a fila do Amazon SQS que contenha um par de valores-chave e verifique se o Lambda grava um item na tabela do DynamoDB usando esses valores.

Testar a aplicação sem servidor

1. Abra a página Filas do console do Amazon SQS e selecione a fila que o AWS SAM criou com base no modelo. O nome tem o formato `sam-app-LambdaIaCQueue-xL87VeKsGiIo`.
2. Selecione Enviar e receber mensagens e cole o JSON a seguir no Corpo da mensagem na seção Enviar mensagem.

   ```
   {
       "myKey": "myValue"
   }
   ```

3. Escolha Send Message (Enviar mensagem).

   Enviar a mensagem para a fila fará com que o Lambda invoque sua função por meio do mapeamento da origem do evento definido no modelo do AWS SAM. Para confirmar que o Lambda invocou sua função conforme o esperado, verifique se um item foi adicionado à tabela do DynamoDB.

4. Abra a página Tabelas do console do DynamoDB e selecione sua tabela. O nome tem o formato `sam-app-LambdaIaCTable-CN0S66C0VLNV`.
5. Escolha Explore table items (Explorar itens da tabela). No painel Itens retornados, você verá um item com o id `myKey` e o Valor `myValue`.

Próximas etapas

Para saber mais sobre como usar o Application Composer com o AWS SAM e o AWS CloudFormation, comece conferindo Using Application Composer with AWS CloudFormation and AWS SAM.

Para ver um tutorial guiado que usa o AWS SAM para implantar uma aplicação sem servidor projetada no Application Composer, também recomendamos que você realize o tutorial do AWS Application Composer em AWS Serverless Patterns Workshop.

O AWS SAM tem uma interface de linha de comando (CLI) que você pode usar com modelos do AWS SAM e integrações de terceiros compatíveis para criar e executar aplicações sem servidor. Com a CLI do AWS SAM, você pode criar e implantar a aplicação, realizar testes e depuração no local, configurar pipelines de CI/CD e muito mais. Para saber mais sobre como usar a CLI do AWS SAM, consulte Getting started with AWS SAM no Guia do desenvolvedor do AWS Serverless Application Model.

Para saber como implantar uma aplicação sem servidor com um modelo do AWS SAM usando o console do AWS CloudFormation, confira primeiro Usar o console do AWS CloudFormation no Guia do usuário do AWS CloudFormation.

Regiões com suporte para integração do Lambda com o Application Composer

As seguintes Regiões da AWS têm suporte para a integração do Lambda com o Application Composer:

- Leste dos EUA (Norte da Virgínia)
- Leste dos EUA (Ohio)
- Oeste dos EUA (N. da Califórnia)
- Oeste dos EUA (Oregon)
- África (Cidade do Cabo)
- Ásia-Pacífico (Hong Kong)
- Ásia-Pacífico (Hyderabad)
- Ásia-Pacífico (Jacarta)
- Ásia-Pacífico (Melbourne)
- Ásia-Pacífico (Mumbai)
- Ásia-Pacífico (Osaka)
- Ásia-Pacífico (Seul)
- Ásia-Pacífico (Singapura)
- Ásia-Pacífico (Sydney)

- Ásia-Pacífico (Tóquio)
- Canadá (Central)
- Europa (Frankfurt)
- Europa (Zurique)
- Europa (Irlanda)
- Europe (London)
- Europa (Estocolmo)
- Oriente Médio (Emirados Árabes Unidos)

Redes privadas com a VPC

A Amazon Virtual Private Cloud (Amazon VPC) é uma rede virtual na nuvem da AWS, dedicada à sua conta da AWS. Você pode usar uma Amazon VPC para criar uma rede privada para recursos como bancos de dados, instâncias de cache ou serviços internos. Para obter mais informações sobre a Amazon VPC, consulte O que é a Amazon VPC?

Uma função do Lambda sempre é executada dentro de uma VPC de propriedade do serviço Lambda. O Lambda aplica regras de segurança e de acesso à rede a essa VPC, e mantém e monitora a VPC automaticamente. Se a função do Lambda precisar acessar os recursos na VPC da sua conta, configure a função para acessar a VPC. O Lambda fornece recursos gerenciados denominados ENIs de hiperplano que a função do Lambda usa para conectar a VPC do Lambda a uma ENI (interface de rede elástica) na VPC da sua conta.

Não há custo adicional pelo uso da VPC ou de uma ENI de hiperplano. Há cobranças para alguns componentes da VPC, como gateways NAT. Para obter mais informações, consulte Definição de preço da Amazon VPC.

Tópicos

Elementos da rede VPC

As redes Amazon VPC incluem os seguintes elementos de rede:

- Interface de rede elástica: uma interface de rede elástica é um componente lógico de rede em uma VPC que representa uma placa de rede virtual.

- Sub-rede: um intervalo de endereços IP na VPC. Você pode adicionar recursos da AWS a uma sub-rede especificada. Use uma sub-rede pública para os recursos que deverão ser conectados à Internet e uma sub-rede privada para os recursos que não serão conectados à Internet.
- Grupo de segurança: use grupos de segurança para controlar o acesso aos recursos da AWS em cada sub-rede.
- Lista de controle de acesso (ACL): use uma ACL de rede para fornecer segurança adicional em uma sub-rede. A ACL de sub-rede padrão permite todo o tráfego de entrada e saída.
- Tabela de rotas: contém um conjunto de rotas que a AWS usa para direcionar o tráfego da rede para a VPC. Você pode associar explicitamente uma sub-rede a uma tabela de rotas específica. Por padrão, a sub-rede é associada à tabela de rotas principal.
- Rota: cada rota em uma tabela de rotas especifica um intervalo de endereços IP e o destino para o qual o Lambda envia o tráfego para esse intervalo. A rota também especifica um destino, que é o gateway, a interface de rede ou a conexão por meio da qual o tráfego será enviado.
- Gateway NAT: um serviço de conversão de endereços de rede (NAT) da AWS que controla o acesso de uma sub-rede privada da VPC privada à Internet.
- Endpoints de VPC: é possível usar um endpoint da VPC da Amazon para criar conectividade privada com serviços hospedados na AWS sem necessidade de acesso pela Internet ou por meio de um dispositivo NAT, uma conexão VPN ou uma conexão da AWS Direct Connect. Para obter mais informações, consulte AWS PrivateLink e endpoints da VPC.

> **Tip**
>
> Para configurar a função do Lambda para acessar uma VPC e uma sub-rede, é possível usar o console ou a API do Lambda.
> Consulte a seção `VpcConfig` em CreateFunction para configurar a função. Para detalhes das etapas, consulte Como anexar funções do Lambda a uma Amazon VPC em sua Conta da AWS.

Para obter mais informações sobre definições de rede da Amazon VPC, consulte Como funciona a Amazon VPC no Guia do desenvolvedor da Amazon VPC e Perguntas frequentes da Amazon VPC.

Conectar funções do Lambda à VPC

Uma função do Lambda sempre é executada dentro de uma VPC de propriedade do serviço Lambda. Por padrão, uma função do Lambda não é conectada às VPCs da sua conta. Quando você

conecta uma função a uma VPC em sua conta, a função não consegue acessar a Internet, a menos que a VPC conceda acesso.

O Lambda acessa os recursos na VPC usando uma ENI de hiperplano. As ENIs de hiperplano fornecem recursos de NAT da VPC do Lambda à VPC da conta usando NAT de VPC a VPC (V2N). O V2N fornece conectividade da VPC do Lambda à VPC da conta, mas não no sentido inverso.

Quando você cria uma função do Lambda (ou atualiza as configurações da VPC), o Lambda aloca uma ENI de hiperplano para cada sub-rede na configuração da VPC da sua função. Várias funções do Lambda poderão compartilhar uma interface de rede se as funções compartilharem a mesma sub-rede e o mesmo grupo de segurança.

Para se conectar a outro serviço da AWS, você pode usar endpoints da VPC para comunicações privadas entre a VPC e os serviços da AWS compatíveis. Uma abordagem alternativa é usar um gateway NAT para rotear o tráfego de saída para um outro serviço da AWS.

Para conceder acesso à Internet para a função, roteie o tráfego de saída para um gateway NAT em uma sub-rede pública. O gateway NAT tem um endereço IP público e pode se conectar à Internet pelo gateway da Internet da VPC. Para ter mais informações, consulte Habilitar o acesso à Internet para funções do Lambda conectadas à VPC.

Sub-redes compartilhadas

O compartilhamento de VPC permite que várias contas da AWS criem os recursos de aplicações, como instâncias do Amazon EC2 e funções do Lambda, em nuvens privadas virtuais (VPCs) compartilhadas e gerenciadas de forma centralizada. Nesse modelo, a conta que possui a VPC (proprietário) compartilha uma ou mais sub-redes com outras contas (participantes) que pertencem ao mesmo AWS Organization.

Para acessar recursos privados, conecte a função a uma sub-rede privada compartilhada em sua VPC. O proprietário da sub-rede deve compartilhar uma sub-rede com você antes que você possa conectar uma função a ela. O proprietário da sub-rede também pode cancelar o compartilhamento da sub-rede posteriormente, removendo assim a conectividade. Para obter detalhes sobre como compartilhar, cancelar o compartilhamento e gerenciar recursos da VPC em sub-redes compartilhadas, consulte Compartilhar sua VPC com outras contas no guia Amazon VPC.

ENIs de hiperplano do Lambda

A ENI de hiperplano é um recurso de rede gerenciado que o serviço Lambda cria e gerencia. Vários ambientes de execução na VPC do Lambda podem usar uma ENI de hiperplano para acessar, em

segurança, recursos dentro das VPCs da sua conta. As ENIs de hiperplano fornecem recursos NAT da VPC do Lambda para a VPC da sua conta.

Para cada sub-rede, o Lambda cria uma interface de rede para cada conjunto exclusivo de grupos de segurança. As funções da conta que compartilham a mesma combinação de sub-rede e grupo de segurança usarão as mesmas interfaces de rede. As conexões feitas por meio da camada de hiperplano são monitoradas automaticamente, mesmo que a configuração do grupo de segurança não exija monitoramento. Os pacotes de entrada da VPC que não correspondem às conexões estabelecidas são descartados na camada de hiperplano. Para obter mais informações, consulte Rastreamento de conexão do grupo de segurança no Guia do usuário do Amazon EC2.

Como as funções da sua conta compartilham os recursos de ENI, o ciclo de vida da ENI é mais complexo que o de outros recursos do Lambda. As seções a seguir descrevem o ciclo de vida da ENI.

Ciclo de vida da ENI

- Criar ENIs
- Gerenciar ENIs
- Excluir ENIs

Criar ENIs

O Lambda pode criar recursos de ENI de hiperplano para uma função habilitada para VPC recém-criada ou para uma alteração da configuração de VPC de uma função existente. A função permanece no estado pendente enquanto o Lambda cria os recursos necessários. Quando a ENI de hiperplano estiver pronta, a função fará a transição para o estado ativo e a ENI ficará disponível para uso. O Lambda pode precisar de vários minutos para criar uma ENI de hiperplano.

Para uma função habilitada para VPC recém-criada, quaisquer invocações ou outras ações de API que operem na função falharão até que o estado da função passe para ativo.

Para uma alteração da configuração de VPC de uma função existente, todas as invocações de função continuam a usar a ENI de hiperplano associada à antiga configuração de sub-rede e grupo de segurança até que o estado da função passe para ativo.

Se uma função do Lambda permanecer ociosa por 30 dias, o Lambda recuperará as ENIs de hiperplano não utilizadas e definirá o estado da função como ocioso. A próxima invocação fará com

que o Lambda reative a função ociosa. A invocação falha e a função entra no estado pendente até que o Lambda conclua a criação ou alocação de uma ENI de hiperplano.

Para obter mais informações sobre estados de funções, consulte Estados da função do Lambda.

Gerenciar ENIs

O Lambda usa as permissões da função de execução da sua função para criar e gerenciar interfaces de rede. O Lambda cria uma ENI de hiperplano quando você define uma combinação exclusiva de sub-rede e grupo de segurança para uma função habilitada para VPC em uma conta. O Lambda reutiliza a ENI de hiperplano para outras funções habilitadas para VPC da sua conta que usam a mesma combinação de sub-rede e grupo de segurança.

Não há cotas para o número de funções do Lambda que podem usar a mesma ENI de hiperplano. Porém, cada ENI de hiperplano comporta até 65.000 conexões/portas. Se o número de conexões exceder a 65.000, o Lambda criará uma nova ENI de hiperplano para fornecer conexões adicionais.

Quando você atualiza a configuração da sua função para acessar uma VPC diferente, o Lambda encerra a conectividade da ENI do hiperplano na VPC anterior. O processo de atualização da conectividade para uma nova VPC pode levar vários minutos. Durante esse tempo, as invocações da função continuam a usar a VPC anterior. Após a conclusão da atualização, novas invocações começarão a usar a ENI de hiperplano na nova VPC. Neste ponto, a função do Lambda não estará mais conectada à VPC anterior.

Excluir ENIs

Quando você atualiza uma função para remover sua configuração de VPC, o Lambda leva até 20 minutos para excluir a ENI de hiperplano anexada. O Lambda só exclui a ENI se nenhuma outra função (ou versão de função publicada) estiver usando essa ENI de hiperplano.

O Lambda depende de permissões na função de execução da função para excluir a ENI de hiperplano. Se você excluir a função de execução antes que o Lambda exclua a ENI de hiperplano, o Lambda não poderá excluir a ENI de hiperplano. Você pode executar a exclusão manualmente.

O Lambda não exclui interfaces de rede que estejam sendo usadas por funções ou versões de função da sua conta. Você pode usar o Localizador de ENI do Lambda para identificar as funções ou versões de função que estão usando uma ENI de hiperplano. Para as funções ou versões de função de que você não precisa mais, é possível remover a configuração de VPC para que o Lambda exclua a ENI de hiperplano.

Conexões

O Lambda é compatível com dois tipos de conexões: TCP (Transmission Control Protocol) e UDP (User Datagram Protocol).

Quando você cria uma VPC, o Lambda cria automaticamente um conjunto de opções de DHCP e associa-as à VPC. Você pode configurar suas próprias opções de DHCP definidas para a VPC. Para obter mais detalhes, consulte Opções de DHCP da Amazon VPC.

A Amazon fornece um servidor DNS (o Amazon Route 53 Resolver) à VPC. Para obter mais informações, consulte Suporte de DNS para sua VPC.

Suporte a IPv6

O Lambda oferece suporte a conexões de entrada aos endpoints de pilha dupla públicos do Lambda e a conexões de saída para sub-redes VPC de pilha dupla via IPv6.

Entrada

Para invocar sua função pelo IPv6, use os endpoints de pilha dupla públicos do Lambda. Os endpoints de pilha dupla são compatíveis com IPv4 e IPv6. Os endpoints de pilha dupla do Lambda usam a seguinte sintaxe:

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

Você também pode usar URLs de função do Lambda para invocar funções via IPv6. Os endpoints de URLs de função têm o seguinte formato:

```
https://url-id.lambda-url.us-east-1.on.aws
```

Saída

Sua função pode se conectar a recursos em sub-redes de VPC de pilha dupla via IPv6. Esta opção é desativada por padrão. Para permitir o tráfego IPv6 de saída, use o console ou a opção `--vpc-config Ipv6AllowedForDualStack=true` com o comando create-function ou update-function-configuration.

> **Note**
>
> Para permitir o tráfego IPv6 de saída em uma VPC, todas as sub-redes conectadas à função devem ser sub-redes de pilha dupla. O Lambda não oferece suporte a conexões

IPv6 de saída para sub-redes somente IPv6 em uma VPC, conexões IPv6 de saída para funções que não estão conectadas a uma VPC ou conexões IPv6 de entrada usando endpoints da VPC (AWS PrivateLink).

Você pode atualizar seu código de função para se conectar explicitamente a recursos da sub-rede via IPv6. O exemplo em Python a seguir abre um soquete e se conecta a um servidor IPv6.

Example — Conectar ao servidor 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()
```

Segurança

A AWS fornece grupos de segurança e ACLs de rede para aumentar a segurança da VPC. Os grupos de segurança controlam o tráfego de entrada e de saída para seus recursos e as Network ACLs controlam o tráfego de entrada e de saída de suas sub-redes. Os grupos de segurança fornecem controle de acesso suficiente para a maioria das sub-redes. Você pode usar as ACLs de

rede se desejar uma camada adicional de segurança para a VPC. Para obter mais informações, consulte Privacidade do tráfego entre redes na Amazon VPC. Toda sub-rede que você cria está automaticamente associada ao Network ACL padrão da VPC. Você pode alterar a associação e o conteúdo do Network ACL padrão.

Para obter as práticas recomendadas de segurança em geral, consulte Práticas recomendadas de segurança para a VPC. Para obter detalhes sobre como usar o IAM para gerenciar o acesso à API e aos recursos do Lambda, consulte Permissões do AWS Lambda.

É possível usar chaves de condição específicas do Lambda para configurações de VPC a fim de fornecer controles de permissão adicionais para as funções do Lambda. Para obter mais informações sobre chaves de condição da VPC, consulte Using IAM condition keys for VPC settings (Usar chaves de condição do IAM para configurações da VPC).

> **Note**
>
> As funções do Lambda podem ser invocadas da Internet pública ou dos endpoints do AWS PrivateLink. É possível acessar os URLs da função somente por meio da Internet pública. Embora as funções do Lambda sejam compatíveis com o AWS PrivateLink, os URLs da função não são.

Observabilidade

Você pode usar os logs de fluxo da VPC para capturar informações sobre tráfego IP de entrada e de saída nas interfaces de rede da VPC. Você pode publicar os dados do log de fluxo no Amazon CloudWatch Logs ou no Amazon S3. Após criar um log de fluxo, você poderá recuperar e visualizar seus dados no destino selecionado.

Observação: quando você anexa uma função a uma VPC, as mensagens do log do CloudWatch não usam as rotas da VPC. O Lambda as envia usando o roteamento regular para logs.

Configurar a arquitetura do conjunto de instruções para uma função do Lambda

A arquitetura do conjunto de instruções de uma função do Lambda determina o tipo de processador de computador que o Lambda usa para executar a função. O Lambda fornece opções de arquiteturas de conjuntos de instruções:

- arm64: arquitetura ARM de 64 bits para o processador AWS Graviton2.
- x86_64: arquitetura x86 de 64 bits para processadores baseados em x86.

> **Note**
>
> A arquitetura arm64 está disponível na maioria das Regiões da AWS. Para obter mais informações, consulte Preços do AWS Lambda. Na tabela de preços de memória, escolha a guia Arm Price (Preço do ARM) e, em seguida, abra a lista suspensa Region (Regiões) para visualizar quais Regiões da AWS oferecem suporte ao arm64 com o Lambda.
> Para obter um exemplo de como criar uma função com arquitetura arm64, consulte AWS Lambda Functions Powered by AWS Graviton2 Processor.

Tópicos

Vantagens do uso da arquitetura arm64

As funções do Lambda que usam arquitetura arm64 (processador AWS Graviton2) podem alcançar preços e performance significativamente melhores do que a função equivalente executada na arquitetura x86_64. Considere usar o arm64 para aplicações com uso intenso de computação, como computação de alta performance, codificação de vídeo e workloads de simulação.

A CPU Graviton2 usa o núcleo Neoverse N1 e oferece suporte a Armv8.2 (incluindo extensões CRC e cripto), além de várias outras extensões de arquitetura.

O Graviton2 reduz o tempo de leitura da memória fornecendo um cache L2 maior por vCPU, o que melhora a performance de latência de backends da Web e móveis, microsserviços e sistemas de processamento de dados. O Graviton2 também oferece melhor performance de criptografia e oferece suporte a conjuntos de instruções que melhoram a latência da inferência de machine learning com base em CPU.

Para obter mais informações sobre o AWS Graviton2, consulte Processador AWS Graviton.

Requisitos para a migração para a arquitetura arm64

Ao selecionar uma função do Lambda para migrar para a arquitetura arm64, a fim de garantir uma migração suave, certifique-se de que sua função atenda aos seguintes requisitos:

- No momento, a função usa um runtime Amazon Linux 2 do Lambda.
- O pacote de implantação contém apenas componentes de código aberto e código-fonte controlados por você para que você possa fazer as atualizações necessárias para a migração.
- Se o código da função incluir dependências de terceiros, cada biblioteca ou pacote fornecerá uma versão arm64.

Compatibilidade do código da função com a arquitetura arm64

O código da sua função do Lambda deve ser compatível com a arquitetura do conjunto de instruções da função. Antes de migrar uma função para a arquitetura arm64, observe os seguintes pontos sobre o código da função atual:

- Se você adicionou o código da função usando o editor de código integrado, seu código provavelmente será executado em qualquer arquitetura sem precisar de modificações.
- Se você carregou o código da função, deverá carregar um novo código compatível com a arquitetura de destino.
- Se sua função usa camadas, será necessário verificar cada camada para garantir que ela seja compatível com a nova arquitetura. Se uma camada não for compatível, edite a função para substituir a versão da camada atual por uma versão de camada compatível.
- Se sua função usa extensões do Lambda, será necessário verificar cada camada para garantir que ela seja compatível com a nova arquitetura.

- Se sua função usar um tipo de pacote de implantação de imagem de contêiner, você deverá criar uma nova imagem de contêiner compatível com a arquitetura da função.

Como migrar para a arquitetura arm64

Para migrar uma função do Lambda para a arquitetura arm64, recomendamos seguir estas etapas:

1. Crie a lista de dependências para sua aplicação ou workload. As dependências comuns incluem:
   - Todas as bibliotecas e pacotes usados pela função.
   - As ferramentas que você usa para criar, implantar e testar a função, como compiladores, pacotes de testes, pipelines de integração contínua e entrega contínua (CI/CD), ferramentas de provisionamento e scripts.
   - As extensões do Lambda e ferramentas de terceiros que você usa para monitorar a função na produção.
2. Para cada uma das dependências, verifique a versão e verifique se há versões arm64 disponíveis.
3. Crie um ambiente para migrar sua aplicação.
4. Faça o bootstrap da aplicação.
5. Teste e depure a aplicação.
6. Teste a performance da função arm64. Compare a performance com a versão x86_64.
7. Atualize o pipeline da infraestrutura para oferecer suporte às funções do Lambda em arm64.
8. Envie a implantação para a produção.

   Por exemplo, use a configuração de roteamento de alias para dividir o tráfego entre as versões x86 e arm64 da função e comparar a performance e a latência.

Para obter mais informações sobre como criar um ambiente de código para a arquitetura arm64, incluindo informações específicas de linguagem para Java, Go, .NET e Python, consulte Conceitos básicos do AWS Graviton no GitHub.

Configuração da arquitetura do conjunto de instruções

Você pode configurar a arquitetura do conjunto de instruções para funções do Lambda novas e existentes usando o console do Lambda, a AWS, SDKs, a AWS Command Line Interface (AWS CLI) ou o AWS CloudFormation. Siga estas etapas para alterar a arquitetura do conjunto de instruções para uma função existente do Lambda no console.

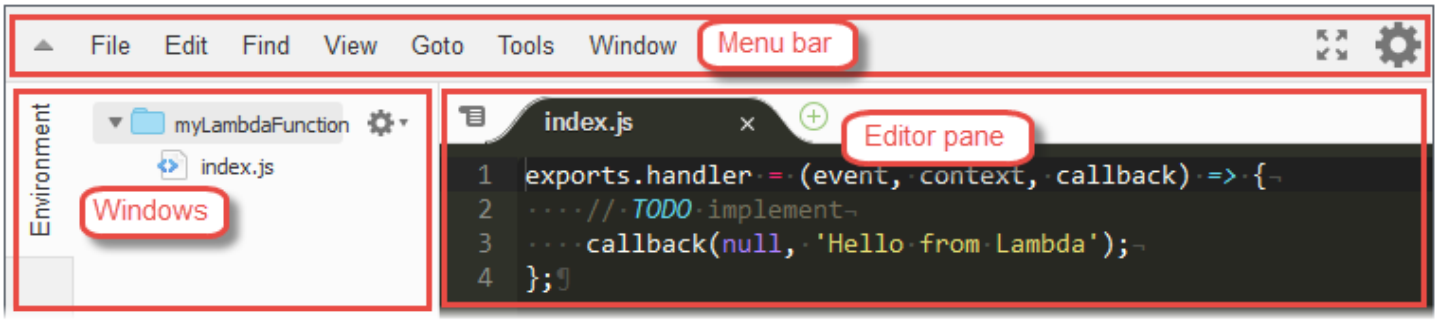
1. Abra a página Funções do console do Lambda.
2. Escolha o nome da função para a qual você deseja configurar a arquitetura do conjunto de instruções.
3. Na guia principal Código, na seção Configurações de runtime, escolha Editar.
4. Em Arquitetura, escolha a arquitetura do conjunto de instruções a ser usado pela sua função.
5. Escolha Salvar.

> **Note**
>
> Todos os runtimes do Amazon Linux 2 são compatíveis com as arquiteturas x86_64 e ARM CPU.
> Os runtimes que não usam o Amazon Linux 2, como o Go 1.x, não são compatíveis com a arquitetura arm64. Para usar a arquitetura arm64 com o Go 1.x, você pode executar sua função em um runtime provided.al2. Para obter mais informações, consulte as instruções de implantação para pacotes .zip e imagens de contêiner.

Editar código usando o editor do console do Lambda

É possível usar o editor de código no console do Lambda para escrever, testar e visualizar os resultados da execução do código de função do Lambda. O editor de código é compatível com linguagens que não exigem compilação, como Node.js e Python. O editor de código é compatível somente com pacotes de implantação de arquivos em .zip e o tamanho do pacote de implantação deve ser inferior a 3 MB.

O editor de código inclui a barra de menu, as janelas e o painel do editor.



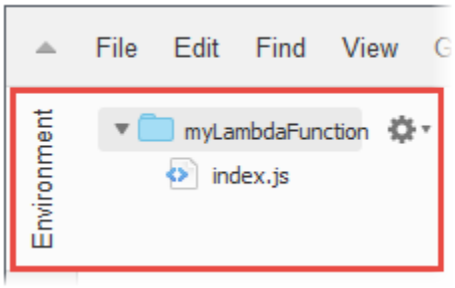
Para obter uma lista do que os comandos fazem, consulte a referência de comandos da barra de menus no Guia do usuário do AWS Cloud9. Alguns dos comandos listados nessa referência não estão disponíveis no editor de código.

Tópicos

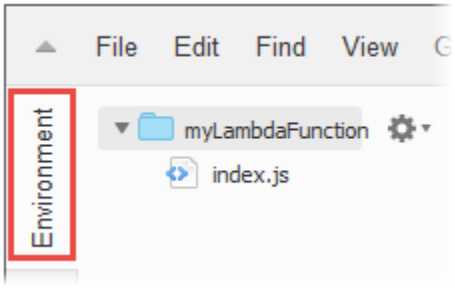
- Trabalhar com arquivos e pastas
- Trabalhar com códigos
- Trabalhar no modo de tela cheia
- Trabalhar com preferências

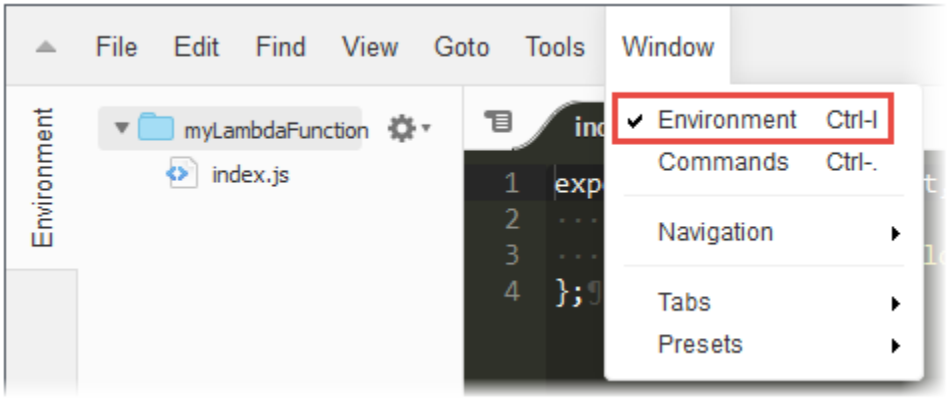
Trabalhar com arquivos e pastas

Use a janela Environment (Ambiente) no editor de código para criar, abrir e gerenciar arquivos para sua função.



Para exibir ou ocultar a janela Environment, escolha o botão Environment (Ambiente). Se o botão Environment (Ambiente) não estiver visível, escolha Window, Environment (Janela, ambiente) na barra de menus.



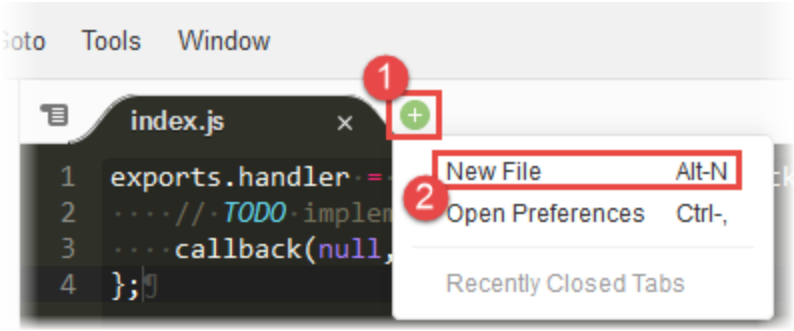


Para abrir um único arquivo e mostrar seu conteúdo no painel do editor, clique duas vezes no arquivo na janela Environment (Ambiente).

Para abrir vários arquivos e mostrar seus conteúdos no painel do editor, escolha os arquivos na janela Environment (Ambiente). Clique com o botão direito do mouse na seleção e escolha Open (Abrir).

Para criar um novo arquivo, faça o seguinte:

- Na janela Environment (Ambiente), clique com o botão direito do mouse na pasta para a qual você deseja enviar o novo arquivo e escolha New File (Novo arquivo). Insira o nome e a extensão do arquivo e, em seguida, pressione Enter.
- Escolha File, New File (Arquivo, novo arquivo) na barra de menus. Quando estiver pronto para salvar o arquivo, escolha File, Save ou File, Save As na barra de menus. Use a caixa de diálogo Save As exibida para nomear o arquivo e escolha onde salvá-lo.
- Na barra de botões da guia no painel do editor, escolha o botão + e escolha New File (Novo arquivo). Quando estiver pronto para salvar o arquivo, escolha File, Save ou File, Save As na barra de menus. Use a caixa de diálogo Save As exibida para nomear o arquivo e escolha onde salvá-lo.



Para criar uma nova pasta, clique com o botão direito do mouse na pasta na janela Environment onde você deseja que a nova pasta seja criada e escolha New Folder. Insira o nome da pasta e, em seguida, pressione Enter.

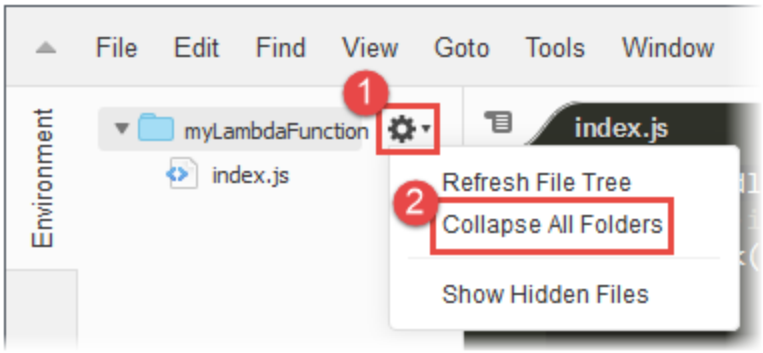
Para salvar um arquivo, com o arquivo aberto e seu conteúdo visível no painel do editor, escolha File, Save na barra de menus.

Para renomear um arquivo ou uma pasta, clique com o botão direito do mouse no arquivo ou na pasta na janela Environment. Insira o nome para substituição e, em seguida, pressione Enter.

Para excluir arquivos ou pastas, selecione os arquivos ou as pastas na janela Environment. Clique com o botão direito do mouse na seleção e escolha Delete. Confirme a exclusão escolhendo Yes (para seleção única) ou Yes to All.

Para recortar, copiar, colar ou duplicar arquivos ou pastas, selecione os arquivos ou as pastas na janela Environment. Clique com o botão direito do mouse na seleção e escolha Recortar, Copiar, Colar ou Duplicar, respectivamente.

Para recolher pastas, escolha o ícone de engrenagem na janela Environment e, em seguida, Collapse All Folders.



Para exibir ou ocultar arquivos, escolha o ícone de engrenagem na janela Environment e, em seguida, Show Hidden Files.

Para visualizar as variáveis de ambiente configuradas para a função, faça o seguinte:

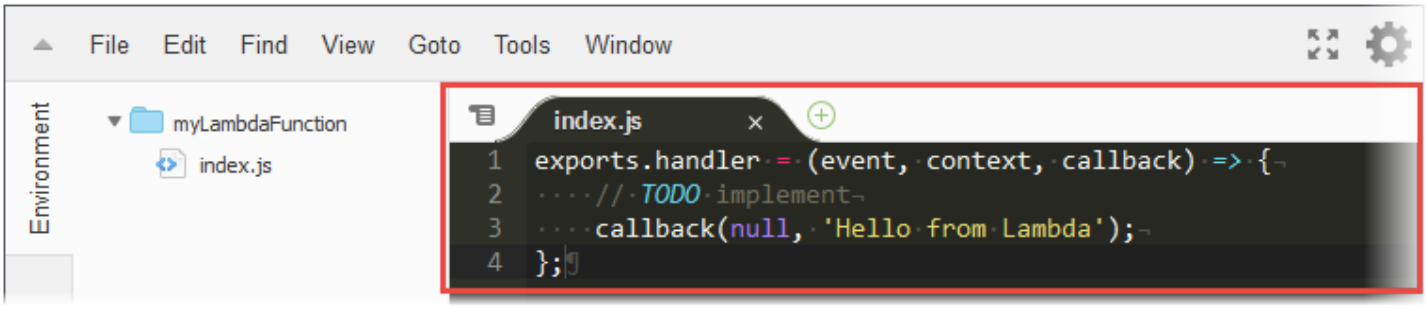
1. Escolha a guia Código.
2. Escolha a guia Variáveis de ambiente.
3. Escolha Ferramentas, Mostrar variáveis de ambiente.

As variáveis de ambiente permanecem criptografadas quando estão listadas no editor de código do console. Se você habilitou os auxiliares de criptografia para a criptografia em trânsito, essas configurações permanecerão inalteradas. Para ter mais informações, consulte Proteger variáveis de ambiente no Lambda.

A lista de variáveis de ambiente é somente leitura e está disponível somente no console do Lambda. Este arquivo não está incluso quando você realiza download do arquivo .zip da função e não é possível adicionar variáveis de ambiente ao fazer upload deste arquivo.

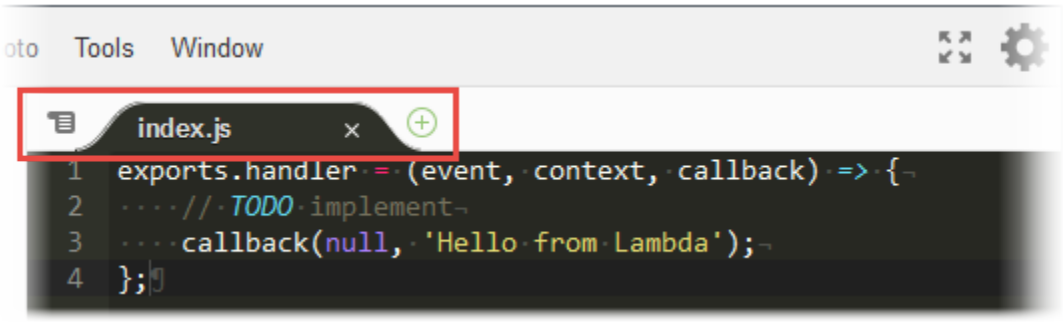
Trabalhar com códigos

Use o painel no editor de código para visualizar e escrever o código.



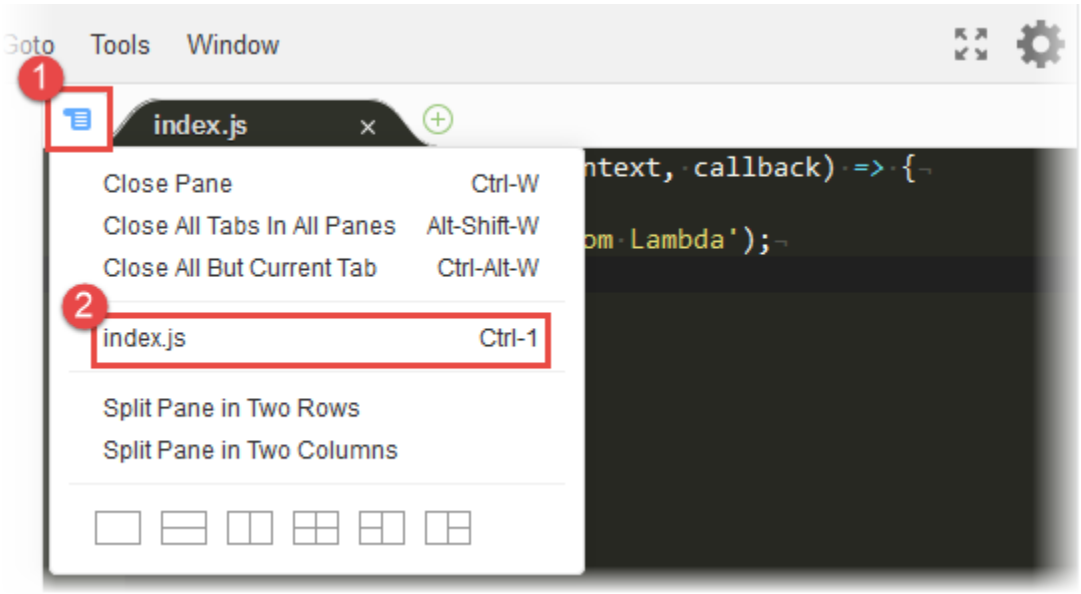
Trabalhar com botões da guia

Use a barra de botões da guia para selecionar, visualizar e criar arquivos.



Para exibir o conteúdo de um arquivo aberto, faça o seguinte:

- Escolha a guia do arquivo.
- Escolha o botão de menu suspenso na barra de botões da guia e, em seguida, escolha o nome do arquivo.



Para fechar um arquivo aberto, faça o seguinte:

- Escolha o ícone X na guia do arquivo.
- Escolha a guia do arquivo. Escolha o botão de menu suspenso na barra de botões da guia e, em seguida, escolha Close Pane.

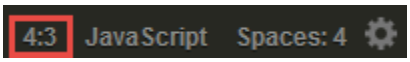
Para fechar vários arquivos abertos, escolha o menu suspenso na barra de botões da guia e escolha Close All Tabs in All Panes ou Close All But Current Tab, conforme necessário.

Para criar um novo arquivo, escolha o botão + na barra de botões da guia e, em seguida, escolha New File. Quando estiver pronto para salvar o arquivo, escolha File, Save ou File, Save As na barra de menus. Use a caixa de diálogo Save As exibida para nomear o arquivo e escolha onde salvá-lo.

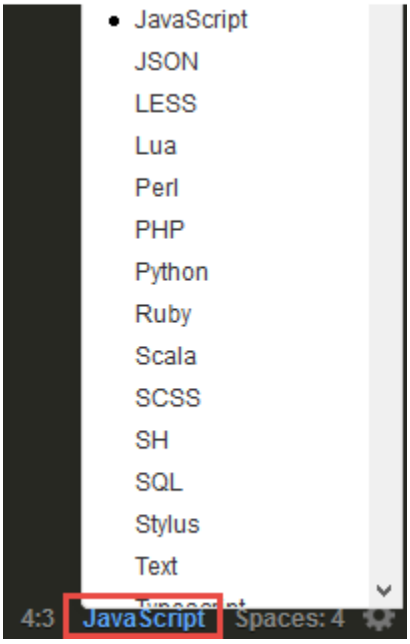
Trabalhar com a barra de status

Use a barra de status para mudar rapidamente para uma linha no arquivo ativo e alterar a forma como o código é exibido.

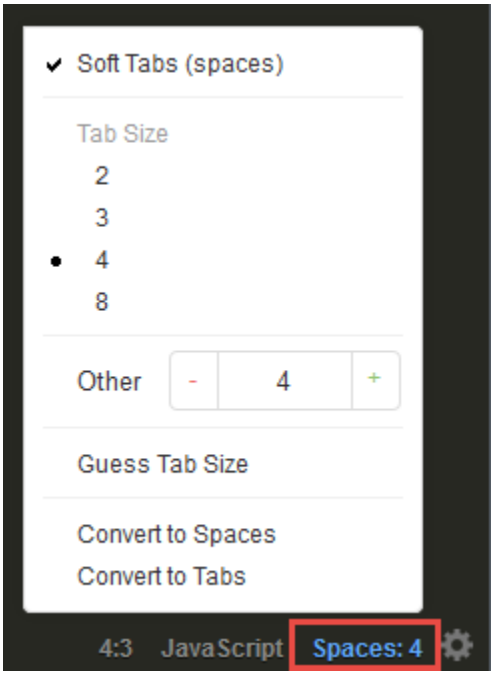
Para mover rapidamente para uma linha no arquivo ativo, escolha o seletor de linha, insira o número da linha para a qual deseja ir e, em seguida, pressione Enter.



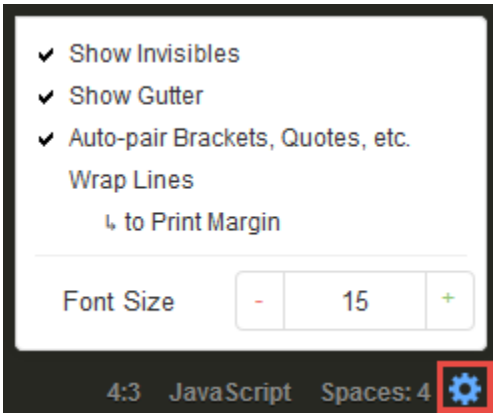
Para alterar o esquema de cores do código no arquivo ativo, escolha o seletor do esquema de cores do código e selecione o novo esquema de cores do código.



Para escolher se tabulações suaves ou espaços serão usados no arquivo ativo, além do tamanho da tabulação ou se deve haver conversão de espaços ou tabulações, escolha o seletor de espaços e tabulações e selecione as novas configurações.



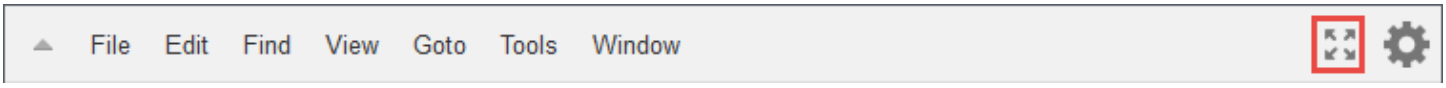
Para definir se todos os arquivos devem mostrar ou ocultar caracteres invisíveis ou gutter, inserir automaticamente pares de colchetes ou aspas, fazer a quebra de linhas ou alterar o tamanho da fonte, escolha o ícone de engrenagem e selecione as novas configurações.



Trabalhar no modo de tela cheia

Você pode expandir o editor de código e ter mais espaço para trabalhar com seu código.

Para expandir o editor de código nas margens da janela do navegador da web, escolha o botão Toggle fullscreen na barra de menus.



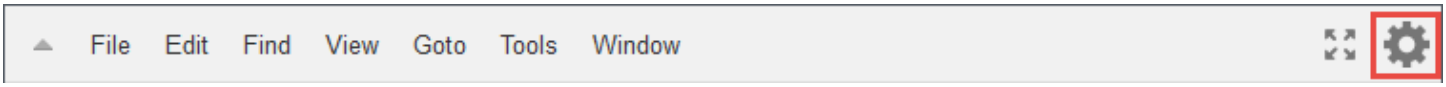
Para diminuir o editor de código até o tamanho original, escolha novamente o botão Toggle fullscreen.

No modo de tela cheia, opções adicionais são exibidas na barra de menus: Save e Test. Save salva o código de função. Test ou Configure Events permitem que você crie ou edite os eventos de teste da função.

Trabalhar com preferências

Você pode alterar várias configurações do editor de código, por exemplo, quais dicas de codificação e avisos serão exibidos, comportamentos de code folding, comportamentos de autopreenchimento de código e muito mais.

Para alterar as configurações do editor de código, escolha o ícone de engrenagem Preferences na barra de menus.



Para obter uma lista das configurações, consulte as referências a seguir no Guia do usuário do AWS Cloud9.

- Project settings that you can change
- Alterações que podem ser feitas nas configurações do usuário

Algumas das configurações listadas nessas referências não estão disponíveis no editor de código.

Recursos adicionais do Lambda

O Lambda fornece um console de gerenciamento e uma API para gerenciar e invocar funções. Proporciona runtimes que oferecem suporte a um conjunto padrão de recursos para que você possa alternar facilmente entre linguagens e frameworks, dependendo do que precisar. Além das funções, também é possível criar versões, aliases, camadas e runtimes personalizados.

Recursos avançados

- Escalabilidade
- Controles de simultaneidade
- URLs de função
- Invocação assíncrona
- Mapeamentos de origem do evento
- Destinos
- Esquemas de funções
- Ferramentas de teste e implantação
- Modelos de aplicativos

Escalabilidade

O Lambda gerencia a infraestrutura que executa o código, e a dimensiona automaticamente em resposta às solicitações recebidas. Quando a função é invocada mais rapidamente do que a capacidade de uma única instância da função processar eventos, o Lambda faz o dimensionamento executando instâncias adicionais. Quando o tráfego diminui, as instâncias inativas são congeladas ou interrompidas. Você só paga pelo tempo que a função gasta inicializando ou processando eventos.

Para ter mais informações, consulte Como entender a escalabilidade da função do Lambda.

Controles de simultaneidade

Use configurações de simultaneidade para garantir que seus aplicativos de produção estejam altamente disponíveis e altamente responsivos.

Para evitar que uma função use muita simultaneidade e reservar uma parte da simultaneidade disponível da sua conta para uma função, use a simultaneidade reservada. A simultaneidade

reservada divide o pool de simultaneidade disponível em subconjuntos. Uma função com simultaneidade reservada usa apenas a simultaneidade do subconjunto dedicado dela.

Para permitir que as funções sejam dimensionadas sem flutuações na latência, use simultaneidade provisionada. Em caso de funções que levam muito tempo para serem inicializadas ou que exigem latência extremamente baixa para todas as chamadas, a simultaneidade provisionada permite pré-inicializar instâncias de sua função e mantê-las em execução o tempo todo. O Lambda integra-se ao Application Auto Scaling para oferecer suporte ao ajuste de escala automático para simultaneidade provisionada com base na utilização.

Para ter mais informações, consulte Configurar a simultaneidade reservada para uma função.

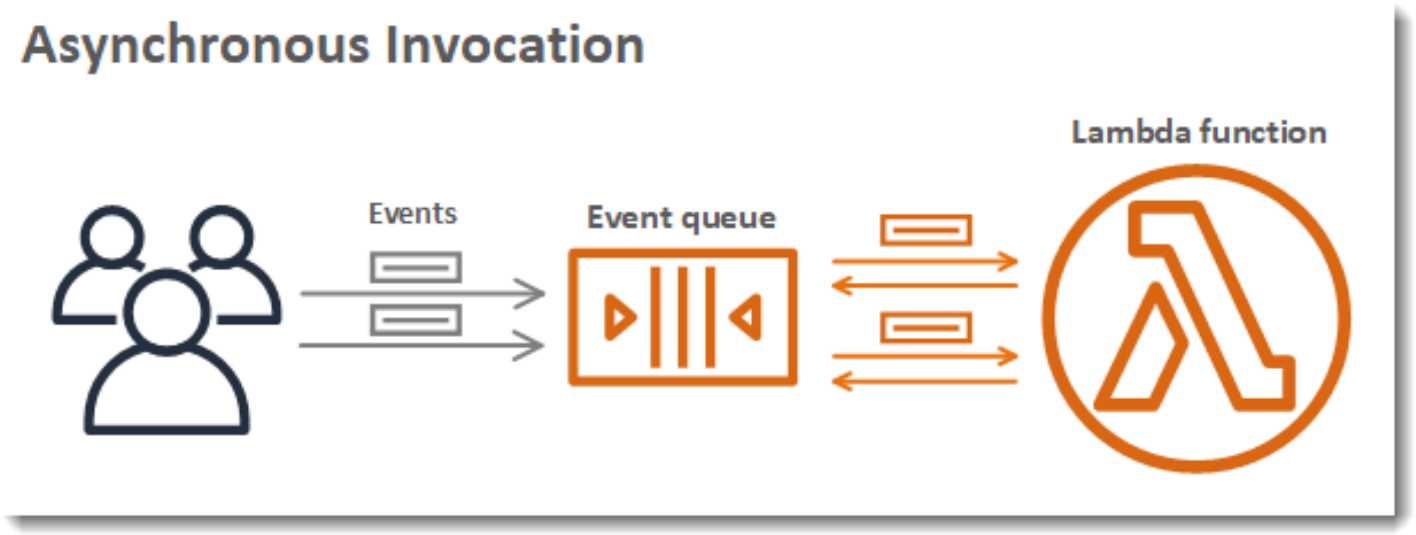
URLs de função

O Lambda é compatível com endpoints HTTP(S) por meio de URLs de função. Com URLs de função, você pode atribuir um endpoint HTTP dedicado à sua função do Lambda. Quando o URL de função está configurado, você pode usá-lo para invocar a função por meio de um navegador da Web, curl, Postman ou qualquer cliente HTTP.

Você pode adicionar um URL de função a uma função existente ou criar uma nova função com um URL de função. Para ter mais informações, consulte Invocar URLs de função do Lambda.

Invocação assíncrona

Quando você invocar uma função, poderá optar por invocá-la de forma síncrona ou assíncrona. Com a invocação síncrona, você aguarda a função processar o evento e retornar uma resposta. Com a invocação assíncrona, o Lambda coloca o evento na fila para processamento e retorna uma resposta imediatamente.

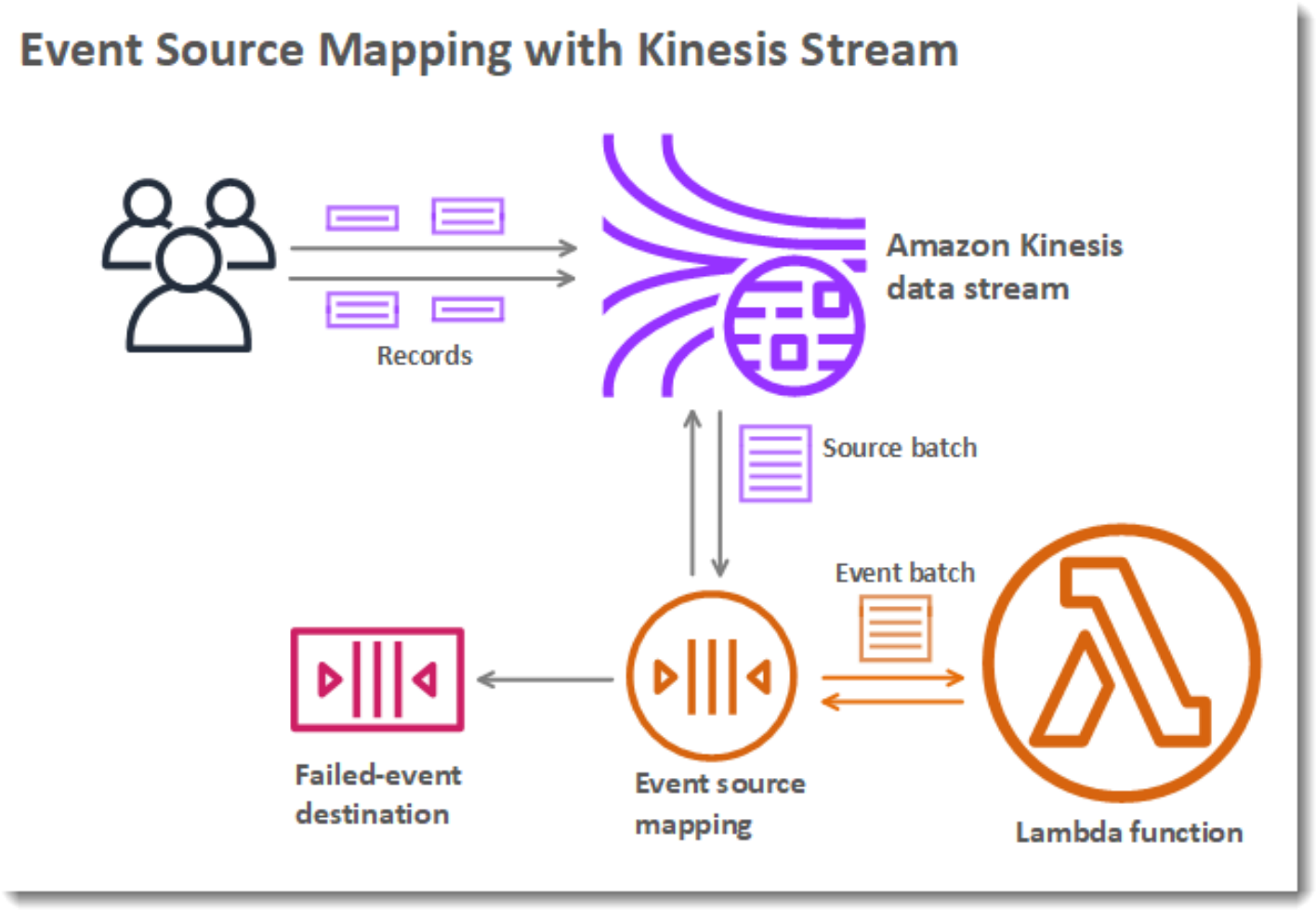


Para chamadas assíncronas, o Lambda manipula novas tentativas se a função retornar um erro ou for limitada. Para personalizar esse comportamento, você pode configurar parâmetros de tratamento de erros em uma função, versão ou alias. Também é possível configurar o Lambda para enviar eventos que falharam no processamento para uma fila de mensagens mortas ou para enviar um registro de qualquer chamada para um destino.

Para ter mais informações, consulte Invocação assíncrona.

Mapeamentos de origem do evento

Para processar itens de um stream ou fila, você pode criar um mapeamento de fonte de eventos. O mapeamento de uma fonte de eventos é um recurso no Lambda que lê itens de uma fila do Amazon Simple Queue Service (Amazon SQS), um Amazon Kinesis Stream ou Amazon DynamoDB Stream e os envia para sua função em lotes. Cada evento que seus processos de função pode conter centenas ou milhares de itens.

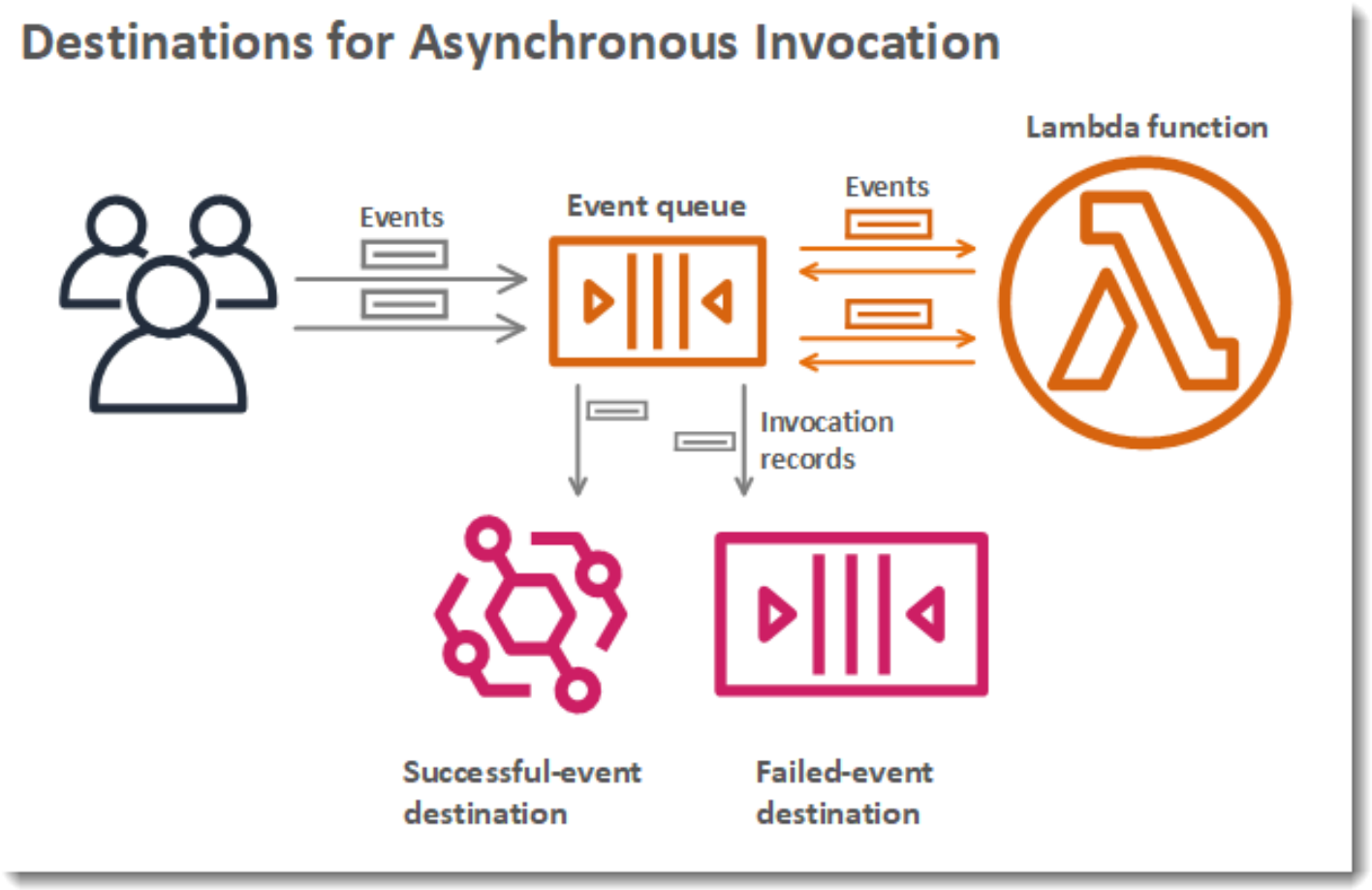


Os mapeamentos de fontes de eventos mantêm uma fila local de itens não processados e manipulam novas tentativas se a função retornar um erro ou for limitada. Você pode configurar um mapeamento de origem de evento para personalizar o tratamento de erros ou de comportamento de lote, e para enviar um registro de itens que falham no processamento para um destino.

Para ter mais informações, consulte Como o Lambda processa registros de origens de eventos baseadas em fluxos e filas.

Destinos

Um destino é um recurso da AWS que recebe registros de chamada para uma função. Para invocação assíncrona, você pode configurar o Lambda para enviar registros de chamada para uma fila, tópico, função ou barramento de eventos. Você pode configurar destinos separados para chamadas bem-sucedidas e eventos que falharam no processamento. O registro de chamada contém detalhes sobre o evento, a resposta da função e o motivo pelo qual o registro foi enviado.



Para mapeamentos de fontes de eventos que são lidos de transmissões, você pode configurar o Lambda para enviar para uma fila ou tópico um registro de lotes que falharam no processamento. Um registro de falhas de um mapeamento de origem de evento contém metadados sobre o lote e aponta para os itens no fluxo.

Para obter mais informações, consulte Configurar destinos para invocação assíncrona e as seções de tratamento de erros de Usar o AWS Lambda com o Amazon DynamoDB e Como o Lambda processa registros do Amazon Kinesis Data Streams.

Esquemas de funções

Ao criar uma função no console do Lambda, é possível optar por começar do zero, usar um esquema ou usar uma imagem de contêiner. Um esquema fornece um código de exemplo que mostra como usar o Lambda com um serviço da AWS ou uma aplicação de terceiros bem conhecida. Os esquemas incluem predefinições de código de exemplo e configuração de funções para runtimes de Node.js e Python.

Os esquemas são fornecidos para uso sob a Licença de Software da Amazon. Eles estão disponíveis apenas no console do Lambda.

Ferramentas de teste e implantação

O Lambda é compatível com a implantação de código no estado em que encontra ou como imagens de contêiner. Você pode usar os serviços e as conhecidas ferramentas da comunidade da AWS, como a interface de linha de comando (CLI) do Docker, para criar e implantar as funções do Lambda. Para configurar a CLI do Docker, consulte Obter Docker no site do Docker Docs. Para obter uma introdução ao uso do Docker com oAWS, consulteConceitos básicos do Amazon ECR usando oAWS CLInoAmazon Elastic Container Registry Guide.

A AWS CLI e a CLI do AWS SAM são ferramentas de linha de comando para gerenciamento de pilhas de aplicações do Lambda. Além de comandos para gerenciar pilhas de aplicações com a API do AWS CloudFormation, o suporte do AWS CLI a comandos de nível superior que simplificam tarefas como o upload de pacotes de implantação e a atualização de modelos. A CLI do AWS SAM fornece outras funcionalidades, como validação de modelos, testes localmente e integração com sistemas de CI/CD.

- Instalar a CLI do AWS SAM
- Testar e depurar aplicações de tecnologia sem servidor com o AWS SAM
- Implantar aplicações com tecnologia sem servidor usando sistemas de CI/CD com o AWS SAM

Modelos de aplicativos

O console do Lambda pode ser usado para criar uma aplicação com um pipeline de entrega contínua. Os modelos de aplicações no console do Lambda incluem código para uma ou mais funções, um modelo de aplicação que define funções e recursos de suporte da AWS e um modelo de infraestrutura que define um pipeline do AWS CodePipeline. O pipeline tem estágios de compilação e implantação que são executados sempre que você enviar alterações para o repositório Git incluído.

Os modelos de aplicativos são fornecidos para serem utilizados sob a licença de MIT No Attribution (MIT sem atribuição) . Eles estão disponíveis apenas no console do Lambda.

Para ter mais informações, consulte Gerenciar aplicativos no console do AWS Lambda.

Saiba como construir soluções com tecnologia sem servidor

> **Tip**
>
> Para saber como construir soluções com tecnologia sem servidor, confira o Guia do desenvolvedor com tecnologia sem servidor.

Runtimes do Lambda

O Lambda oferece suporte a vários idiomas por meio do uso de runtimes. Um runtime fornece um ambiente específico da linguagem que retransmite eventos de invocação, informações de contexto e respostas entre o Lambda e a função. Você pode usar runtimes fornecidos pelo Lambda ou criar seus próprios.

Cada versão principal da linguagem de programação lançada tem um runtime separado, com um identificador de runtime exclusivo, como `nodejs20.x` ou `python3.12`. Para alterar uma função com a finalidade de usar uma nova versão principal da linguagem, é necessário alterar o identificador de runtime. Como o AWS Lambda não pode garantir a compatibilidade com versões anteriores entre as versões principais, essa é uma operação direcionada ao cliente.

Em uma função definida como uma imagem de contêiner, você escolhe um runtime e a distribuição Linux ao criar a imagem de contêiner. Para alterar o runtime, crie uma nova imagem de contêiner.

Quando você usa um arquivo .zip para o pacote de implantação, escolhe um runtime ao criar a função. Para alterar o runtime, você pode atualizar a configuração da sua função. O runtime é emparelhado com uma das distribuições do Amazon Linux. O ambiente de execução subjacente fornece bibliotecas e variáveis de ambiente adicionais que podem ser acessadas no código de sua função.

O Lambda invoca sua função em um ambiente de execução. Um ambiente de execução fornece um ambiente de runtime isolado e seguro, que gerencia os recursos necessários para executar a função. O Lambda reutiliza o ambiente de execução de uma invocação anterior, caso haja alguma disponível, ou pode criar um novo ambiente de execução.

Para usar outras linguagens no Lambda, como Go ou Rust, use um runtime somente para sistema operacional. O ambiente de execução do Lambda fornece uma interface de runtime para obter eventos de invocação e enviar respostas. Você pode implantar outras linguagens implementando um runtime personalizado junto com o código da função ou em uma camada.

Tempos de execução compatíveis

A tabela a seguir lista os runtimes do Lambda compatíveis e as datas de descontinuação projetadas. Depois que um runtime for descontinuado, você ainda poderá criar e atualizar funções por um período limitado. Para ter mais informações, consulte the section called “Uso do runtime após a

descontinuação". A tabela fornece as datas atualmente previstas para a descontinuação do runtime. Essas datas são fornecidas para fins de planejamento e estão sujeitas a alterações.

Runtimes compatíveis

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
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 de junho de 2024	28 de fevereiro de 2025	31 de março de 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 de outubro de 2024	28 de fevereiro de 2025	31 de março de 2025
Java 21	`java21`	Amazon Linux 2023			
Java 17	`java17`	Amazon Linux 2			

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
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 de novembro de 2024	28 de fevereiro de 2025	31 de março de 2025
Ruby 3.3	`ruby3.3`	Amazon Linux 2023			
Ruby 3.2	`ruby3.2`	Amazon Linux 2			
Runtime somente para sistema operacional	`provided.al2023`	Amazon Linux 2023			
Runtime somente para sistema operacional	`provided.al2`	Amazon Linux 2			

> **Note**
>
> Para novas regiões, o Lambda não oferecerá suporte a runtimes que deverão ser descontinuados nos próximos seis meses.

O Lambda mantém os runtimes gerenciados e suas imagens base de contêiner correspondentes atualizados com patches e suporte para lançamentos de versões menores. Para obter mais informações, consulte Atualizações de runtime do Lambda.

O Lambda continua oferecendo suporte à linguagem de programação Go após a descontinuação do runtime Go 1.x. Para obter mais informações, consulte Migrating AWS Lambda functions from the Go1.x runtime to the custom runtime on Amazon Linux 2 no AWS Compute Blog.

Todos os runtimes do Lambda compatíveis oferecem suporte às arquiteturas x86_64 e arm64.

Novas versões de runtime

O Lambda só fornece runtimes gerenciados para novas versões de linguagem quando a versão atinge a fase de suporte de longo prazo (LTS) do ciclo de lançamento da linguagem. Por exemplo, para o ciclo de lançamento do Node.js, quando a versão atinge a fase de LTS ativo.

Antes de a versão atingir a fase de suporte de longo prazo, permanece em desenvolvimento e ainda pode estar sujeita a alterações significativas. O Lambda aplica atualizações de runtime automaticamente por padrão. Portanto, alterações significativas na versão de um runtime podem impedir que as funções funcionem conforme o esperado.

O Lambda não fornece runtimes gerenciados para versões de linguagem que não estão programadas para a fase de LTS.

A lista a seguir mostra o mês de lançamento previsto para os próximos runtimes do Lambda. Essas datas são apenas indicativas e estão sujeitas a alterações.

- Python 3.13: novembro de 2024
- Node.js 22: novembro de 2024

Política de descontinuação de runtime

Os Runtimes do Lambda para arquivos .zip se baseiam em uma combinação de sistema operacional, linguagem de programação e bibliotecas de software que estão sujeitos a manutenção e atualizações de segurança. A política de descontinuação padrão do Lambda é descontinuar um runtime quando qualquer um de seus componentes importantes chegar ao fim do suporte de longo prazo (LTS) da comunidade e as atualizações de segurança não estiverem mais disponíveis. Geralmente, isso ocorre para runtimes de linguagens, mas, em alguns casos, um runtime pode ser descontinuado porque o sistema operacional (SO) chegou ao fim do LTS.

Depois que um runtime é descontinuado, o AWS pode não aplicar mais patches de segurança a ele, e as funções que o utilizam não são mais elegíveis para suporte técnico. Esses runtimes descontinuados são fornecidos “como estão”, sem nenhuma garantia, e podem conter bugs, erros, defeitos ou outras vulnerabilidades.

Para saber mais sobre como gerenciar atualizações e descontinuações de runtime, consulte as seções a seguir e Managing AWS Lambda runtime upgrades on the AWS Compute Blog.

> **Important**
>
> Ocasionalmente, o Lambda atrasa a descontinuação de um runtime do Lambda por um período limitado além da data de término do suporte da versão da linguagem compatível com o runtime. Durante esse período, o Lambda aplica apenas patches de segurança ao sistema operacional do runtime. O Lambda não aplica patches de segurança aos runtimes de linguagem de programação depois que eles atingem a data de término do suporte.

Descontinuação do runtime do Node.js 16

Em resposta ao feedback dos clientes, a AWS está adiando a descontinuação do runtime do Node.js 16 para até 9 meses depois do fim do LTS da comunidade. O runtime do Node.js 16 será descontinuado na data mostrada na tabela Runtimes compatíveis. Conforme declarado na nota anterior, entre o final do LTS, em 11 de setembro de 2023, e a data de descontinuação, o Lambda aplicará apenas patches do sistema operacional ao runtime. Nenhum patch de segurança para o runtime da linguagem será aplicado durante esse período.

Adiar a descontinuação do Node.js 16 dá aos clientes que usam esse runtime a oportunidade de migrar suas funções diretamente para o Node.js 20, pulando o Node.js 18.

Modelo de responsabilidade compartilhada

O Lambda é responsável por selecionar e publicar atualizações de segurança para todos os runtimes gerenciados e imagens de base de contêiner compatíveis. Por padrão, o Lambda aplicará essas atualizações automaticamente às funções que usam runtimes gerenciados. Se a configuração padrão de atualização automática de runtime foi alterada, consulte o modelo de responsabilidade compartilhada dos controles de gerenciamento de runtime. Em funções implantadas usando imagens de contêiner, você é responsável por recompilar a imagem de contêiner da função a partir da imagem de base mais recente e por implantá-la novamente.

Quando um runtime é descontinuado, cessa a responsabilidade do Lambda de atualizar o runtime gerenciado e as imagens de base de contêiner. Você é responsável por atualizar suas funções para usar um runtime ou imagem de base compatível.

Em todos os casos, você é responsável por aplicar as atualizações ao código de função, incluindo suas dependências. Suas responsabilidades no modelo de responsabilidade compartilhada estão resumidas na tabela a seguir.

Fase do ciclo de vida do runtime	Responsabilidades do Lambda	Suas responsabilidades
Runtime gerenciado compatível	Fornecer atualizações de runtime periódicas com patches de segurança e outras atualizações. Aplicar automaticamente as atualizações de runtime por padrão, (consulte the section called “Controles de gerenciamento de runtime” para ver os comportamentos não padrão).	Atualizar seu código de função, incluindo dependênc ias, para resolver qualquer vulnerabilidade de segurança.
Imagem de contêiner compatível	Fornecer atualizações regulares de imagens de base de contêiner com patches de segurança e outras atualizaç ões,.	Atualizar seu código de função, incluindo dependênc ias, para resolver qualquer vulnerabilidade de segurança. Recompilar e reimplantar regularmente sua imagem de contêiner usando a imagem de base mais recente.
Runtime gerenciado prestes a ser descontinuado	Notifique os clientes por meio da documentação, AWS Health Dashboard, e-mail	Monitore a documentação do Lambda, AWS Health Dashboard, e-mail ou Trusted Advisor para obter informaçõ

Fase do ciclo de vida do runtime	Responsabilidades do Lambda	Suas responsabilidades
	e Trusted Advisor antes de descontinuar um runtime. A responsabilidade por atualizações do runtime termina quando ele é descontinuado.	es sobre a descontinuação de runtimes. Atualize as funções para um runtime compatível antes que o runtime anterior seja descontinuado.
Imagem de contêiner prestes a ser descontinuada	Notificações de descontinuação não disponíveis para funções que usam imagens de contêiner. A responsabilidade pelas atualizações de imagens de base de contêiner termina quando elas são descontinuadas.	Preste atenção aos cronogramas de descontinuação e atualize as funções para uma imagem de base compatível antes que a imagem anterior seja descontinuada.

Uso do runtime após a descontinuação

Depois que um runtime é descontinuado, o AWS pode não aplicar mais patches de segurança a ele, e as funções que o utilizam não são mais elegíveis para suporte técnico. Esses runtimes descontinuados são fornecidos “como estão”, sem nenhuma garantia, e podem conter bugs, erros, defeitos ou outras vulnerabilidades. As funções que usam um runtime descontinuado também podem apresentar degradação de performance ou outros problemas, como a expiração de um certificado, que podem fazer com que elas deixem de funcionar bem.

Por pelo menos 30 dias após a descontinuação de um runtime, você ainda poderá criar novas funções do Lambda usando esse runtime. A partir de 30 dias após a descontinuação, o Lambda começa a bloquear a criação de novas funções.

Por pelo menos 60 dias após a descontinuação de um runtime, você ainda pode atualizar o código de função e a configuração das funções existentes. A partir de 60 dias após a descontinuação, o

Lambda começa a bloquear a atualização de código de função e configuração para as funções existentes.

> **Note**
>
> Para alguns runtimes, a AWS está atrasando as datas de block-function-create e block-function-update além dos habituais 30 e 60 dias após a descontinuação. A AWS fez essa alteração em resposta ao feedback dos clientes, a fim de proporcionar mais tempo para você atualizar suas funções. Consulte as tabelas em the section called “Tempos de execução compatíveis” e the section called “Runtimes defasados” para ver as datas do seu runtime.

Você pode atualizar uma função para usar uma versão de runtime compatível mais recente indefinidamente após um runtime ser descontinuado. Você deve testar se a função funciona com o novo runtime antes de aplicar uma alteração do runtime em ambientes de produção, pois não é possível reverter para o runtime descontinuado após o período de 60 dias. Recomendamos o uso de versões e aliases de função para permitir uma implantação segura com reversão.

Observe que o período exato pelo qual você pode continuar criando e atualizando funções não é fixo. Esse período pode variar para cada descontinuação e para diferentes Regiões da AWS. As datas nominais para o bloqueio de criações e atualizações de funções são fornecidas na tabela Runtimes compatíveis na primeira seção desta página. O Lambda não iniciará o bloqueio de criações ou atualizações de funções antes das datas indicadas nessa tabela.

Você poderá continuar a invocar suas funções indefinidamente depois que o runtime for descontinuado. No entanto, a AWS recomenda enfaticamente que você migre as funções para um runtime compatível para que as funções continuem a receber patches de segurança e permaneçam elegíveis para receber suporte técnico.

Receber notificações de descontinuação de runtime

Quando um runtime se aproxima da data de descontinuação, o Lambda envia um alerta por e-mail se alguma função da sua Conta da AWS usa esse runtime. As notificações também são exibidas no AWS Health Dashboard e no AWS Trusted Advisor.

- Receber notificações por e-mail:

  O Lambda envia um alerta por e-mail pelo menos 180 dias antes da descontinuação de um runtime. Este e-mail lista as versões $LATEST de todas as funções que usam o runtime. Para

ver uma lista completa das versões de funções afetadas, use o Trusted Advisor ou consulte the section called “Listagem de funções que usam um runtime descontinuado”.

O Lambda envia uma notificação por e-mail ao contato principal da sua Conta da AWS. Para obter informações sobre como visualizar ou atualizar os endereços de e-mail em sua conta, consulte Updating contact information na Referência geral da AWS.

- Receber notificações por meio do AWS Health Dashboard:

  O AWS Health Dashboard exibe uma notificação pelo menos 180 dias antes da descontinuação do runtime. As notificações aparecem na página Integridade da sua conta, em Outras notificações. A guia Recursos afetados da notificação lista as versões $LATEST de todas as funções que usam o runtime.

  > **Note**
  >
  > Para ver uma lista completa e atualizada das versões de funções afetadas, use o Trusted Advisor ou consulte the section called “Listagem de funções que usam um runtime descontinuado”.

  As notificações do AWS Health Dashboard expiram 90 dias após a descontinuação do runtime.

- Usar o AWS Trusted Advisor

  O Trusted Advisor exibe uma notificação pelo menos 180 dias antes da descontinuação do runtime. As notificações são exibidas na página Segurança. Uma lista das funções afetadas é exibida em AWS Lambda Functions Using Deprecated Runtimes. Essa lista de funções mostra versões $LATEST e publicadas, e é atualizada automaticamente para refletir o status atual das funções.

  Você pode ativar notificações semanais por e-mail no Trusted Advisor na página Preferências do console do Trusted Advisor.

Listagem de funções que usam um runtime descontinuado

Além de usar o Trusted Advisor para ver uma lista ativa das funções afetadas pelas descontinuações programadas de runtime, você também pode usar a AWS Command Line Interface (AWS CLI) ou um dos SDKs da AWS para listar todas as versões de função que usam um runtime específico.

Para gerar essa lista usando a AWS CLI, execute o comando a seguir. Substitua `RUNTIME_IDENTIFIER` pelo nome do runtime que está sendo descontinuado e escolha sua própria Região da AWS. Para listar somente as versões da função $LATEST, omita `--function-version ALL` do comando.

```
aws lambda list-functions --function-version ALL --region us-east-1 --output text --
query "Functions[?Runtime=='RUNTIME_IDENTIFIER'].FunctionArn"
```

> **Tip**
>
> O exemplo de comando lista as funções na região `us-east-1` para uma determinada Conta da AWS. Você precisará repetir esse comando para cada região na qual sua conta tiver funções e para cada uma das suas Contas da AWS.

Para saber mais sobre como usar um AWS SDK para listar suas funções usando a ação ListFunctions, consulte a documentação do SDK para sua linguagem de programação preferida. Você também pode usar um dos SDKs da AWS para coletar estatísticas sobre suas funções mais invocadas e invocadas mais recentemente usando as ações da API DescribelogStreams e getMetricStatistics.

Você também pode usar o recurso Consultas avançadas do AWS Config para listar todas as suas funções que usam um runtime afetado. Essa consulta retorna apenas as versões da função $LATEST, mas você pode agregar consultas para listar funções em todas as regiões e em várias Contas da AWS com um único comando. Para saber mais, consulte Querying the Current Configuration State of AWS Auto Scaling Resources no AWS Config Developer Guide.

Runtimes defasados

Os seguintes runtimes atingiram o fim do suporte:

Runtimes defasados

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
.NET 7 (somente contêiner)	`dotnet7`	Amazon Linux 2	14 de maio de 2024		
Java 8	`java8`	Amazon Linux	8 de janeiro de 2024	8 de fevereiro de 2024	28 de fevereiro de 2025
Go 1.x	`go1.x`	Amazon Linux	8 de janeiro de 2024	8 de fevereiro de 2024	28 de fevereiro de 2025
Runtime somente para sistema operacional	`provided`	Amazon Linux	8 de janeiro de 2024	8 de fevereiro de 2024	28 de fevereiro de 2025
Ruby 2.7	`ruby2.7`	Amazon Linux 2	7 de dezembro de 2023	9 de janeiro de 2024	28 de fevereiro de 2025
Node.js 14	`nodejs14.x`	Amazon Linux 2	4 de dezembro de 2023	9 de janeiro de 2024	28 de fevereiro de 2025
Python 3.7	`python3.7`	Amazon Linux	4 de dezembro de 2023	9 de janeiro de 2024	28 de fevereiro de 2025
.NET Core 3.1	`dotnetcore3.1`	Amazon Linux 2	3 de abril de 2023	3 de abril de 2023	3 de maio de 2023

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
Node.js 12	`nodejs12.x`	Amazon Linux 2	31 de março de 2023	31 de março de 2023	30 de abril de 2023
Python 3.6	`python3.6`	Amazon Linux	18 de julho de 2022	18 de julho de 2022	29 de agosto de 2022
.NET 5 (somente contêiner)	`dotnet5.0`	Amazon Linux 2	10 de maio de 2022		
.NET Core 2.1	`dotnetcore2.1`	Amazon Linux	5 de janeiro de 2022	5 de janeiro de 2022	13 de abril de 2022
Node.js 10	`nodejs10.x`	Amazon Linux 2	30 de julho de 2021	30 de julho de 2021	14 de fevereiro de 2022
Ruby 2.5	`ruby2.5`	Amazon Linux	30 de julho de 2021	30 de julho de 2021	31 de março de 2022
Python 2.7	`python2.7`	Amazon Linux	15 de julho de 2021	15 de julho de 2021	30 de maio de 2022
Node.js 8.10	`nodejs8.10`	Amazon Linux	6 de março de 2020		6 de março de 2020
Node.js 4.3	`nodejs4.3`	Amazon Linux	5 de março de 2020		5 de março de 2020
Borda do Node.js 4.3	`nodejs4.3-edge`	Amazon Linux	5 de março de 2020		30 de abril de 2019
Node.js 6.10	`nodejs6.10`	Amazon Linux	12 de agosto de 2019	12 de agosto de 2019	

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
.NET Core 1.0	`dotnetcore1.0`	Amazon Linux	27 de junho de 2019		30 de julho de 2019
.NET Core 2.0	`dotnetcore2.0`	Amazon Linux	30 de maio de 2019		30 de maio de 2019
Node.js 0.10	`nodejs`	Amazon Linux			31 de outubro de 2016

Em quase todos os casos, a data do fim da vida útil de uma versão da linguagem ou de um sistema operacional é conhecida com antecedência. Os links a seguir fornecem programações relacionadas com o fim da vida útil para cada linguagem compatível com o Lambda como um runtime gerenciado.

Políticas de suporte de framework e linguagem

- Node.js: github.com
- Python: devguide.python.org
- Ruby: www.ruby-lang.org
- Java: www.oracle.com Perguntas frequentes sobre o Corretto
- Go: golang.org
- .NET: dotnet.microsoft.com

Atualizações de runtime do Lambda

O Lambda mantém cada runtime gerenciado atualizado com atualizações de segurança, correções de bugs, novos recursos, aprimoramentos de performance e suporte para lançamentos de versões secundárias. Essas atualizações de runtime são publicadas como versões de runtime. O Lambda aplica atualizações de runtime às funções ao migrar a função de uma versão de runtime anterior para uma nova versão de runtime.

Por padrão, para funções que usam runtimes gerenciados, o Lambda aplica atualizações de runtime automaticamente. Com as atualizações automáticas de runtime, o Lambda assume a responsabilidade operacional de corrigir as versões de runtime. Para a maioria dos clientes, as atualizações automáticas são a escolha mais adequada. Para ter mais informações, consulte Controles de gerenciamento de runtime.

O Lambda também publica cada nova versão de runtime como uma imagem de contêiner. Para atualizar as versões de runtime para as funções baseadas em contêiner, é necessário criar uma nova imagem de contêiner a partir da imagem base atualizada e implantar novamente a função.

Cada versão de runtime está associada a um número de versão e a um ARN (nome do recurso da Amazon). Os números de versão de runtime usam um esquema de numeração definido pelo Lambda, independentemente dos números de versão usados pela linguagem de programação. O ARN da versão de runtime corresponde a um identificador exclusivo para cada versão de runtime.

É possível visualizar o ARN da versão de runtime atual da função na linha `INIT_START` dos logs da função e no console do Lambda.

As versões de runtime não devem ser confundidas com os identificadores de runtime. Cada runtime tem um identificador de runtime exclusivo, como `python3.9` ou `nodejs18.x`. Eles correspondem a cada lançamento da versão principal da linguagem de programação. As versões de runtime descrevem a versão de patch de um runtime individual.

> **Note**
>
> O ARN para o mesmo número de versão de runtime pode variar entre as Regiões da AWS e as arquiteturas de CPU.

Tópicos

Controles de gerenciamento de runtime

O Lambda se empenha para fornecer atualizações de runtime compatíveis com as funções existentes. No entanto, como acontece com a aplicação de patches de software, há casos raros em que uma atualização de runtime pode afetar negativamente uma função existente. Por exemplo, os patches de segurança podem expor um problema subjacente em uma função existente que depende do comportamento inseguro anterior. Os controles de gerenciamento de runtime do Lambda ajudam a reduzir o risco de impacto em suas workloads no caso raro de uma incompatibilidade de versão de runtime. Para cada versão de função (`$LATEST` ou a versão publicada), é possível escolher um dos seguintes modos de atualização de runtime:

- Auto (Automático) (padrão): atualize automaticamente para a versão de runtime mais recente e segura usando Lançamento da versão de runtime em duas fases. Recomendamos este modo para a maioria dos clientes, para que você sempre se beneficie das atualizações de runtime.
- Atualização da função: atualize para a versão de runtime mais recente e segura ao atualizar a função. Quando você atualiza a função, o Lambda atualiza o runtime da função para a versão de runtime mais recente e segura. Essa abordagem sincroniza as atualizações de runtime com as implantações de função, oferecendo a você controle sobre quando o Lambda aplicará as atualizações de runtime. Com esse modo, é possível detectar e mitigar incompatibilidades raras de atualização de runtime com antecedência. Ao usar esse modo, é necessário atualizar regularmente as funções para manter o runtime atualizado.
- Manual: atualize manualmente sua versão do runtime. Especifique uma versão de runtime em sua configuração de função. A função usa essa versão de runtime indefinidamente. No caso raro em que uma nova versão de runtime é incompatível com uma função existente, você pode usar este modo para reverter a função para uma versão de runtime anterior. Não recomendamos o uso do modo Manual para tentar obter a consistência de runtime entre as implantações. Para ter mais informações, consulte Reverter uma versão de runtime.

A responsabilidade pela aplicação de atualizações de runtime às funções varia de acordo com o modo de atualização de runtime escolhido. Para ter mais informações, consulte Modelo de responsabilidade compartilhada.

Lançamento da versão de runtime em duas fases

O Lambda apresenta novas versões de runtime nesta ordem:

1. Na primeira fase, o Lambda aplica a nova versão de runtime sempre que você cria ou atualiza uma função. Uma função é atualizada quando você chama as operações da UpdateFunctionConfigurationAPI UpdateFunctionCodeou.
2. Na segunda fase, o Lambda atualiza qualquer função que usa o modo de atualização de runtime Auto (Automático) e que ainda não tenha sido atualizada para a nova versão de runtime.

A duração geral do processo de implantação varia de acordo com diversos fatores, incluindo a gravidade de quaisquer patches de segurança inclusos na atualização de runtime.

Se você estiver desenvolvendo e implantando ativamente as funções, provavelmente adquirirá novas versões de runtime durante a primeira fase. Isso sincroniza as atualizações de runtime com as atualizações de função. No caso raro de a versão de runtime mais recente impactar negativamente a aplicação, essa abordagem permitirá que você tome medidas corretivas imediatas. As funções que não estão em desenvolvimento ativo ainda recebem o benefício operacional de atualizações automáticas de runtime durante a segunda fase.

Essa abordagem não afeta as funções definidas como os modos Function update (Atualização da função) ou Manual. As funções que usam o modo Function update (Atualização da função) recebem as atualizações de runtime mais recentes somente quando você as cria ou atualiza. As funções que usam o modo Manual não recebem as atualizações de runtime.

O Lambda publica novas versões de runtime de maneira gradual e contínua nas Regiões da AWS. Se as funções estiverem definidas para os modos Auto (Automático) ou Function update (Atualização da função), é possível que as funções implantadas ao mesmo tempo em regiões diferentes ou em momentos diferentes na mesma região selecionem diferentes versões de runtime. Os clientes que requerem consistência de versão de runtime garantida em seus ambientes devem usar imagens de contêiner para implantar as funções do Lambda. O modo Manual foi criado como mitigação temporária para permitir a reversão de versão runtime no caso raro de uma versão de runtime ser incompatível com a função.

Reverter uma versão de runtime

No caso raro de uma nova versão de runtime ser incompatível com a função existente, é possível reverter a versão de runtime para uma anterior. Isso mantém sua aplicação funcionando e minimiza as interrupções, fornecendo tempo para corrigir a incompatibilidade antes de retornar à versão de runtime mais recente.

O Lambda não impõe um limite de tempo para o uso de qualquer versão de runtime específica. No entanto, recomendamos fortemente atualizar para a versão de runtime mais recente o mais rápido possível para se beneficiar dos patches de segurança, melhorias de performance e recursos mais recentes. O Lambda oferece a opção de reverter para uma versão de runtime anterior somente como uma mitigação temporária no caso raro de um problema de compatibilidade de atualização de runtime. As funções que usam uma versão de runtime anterior por um período prolongado podem, futuramente, apresentar performance degradada ou problemas, como a expiração de um certificado, que podem fazer com que parem de funcionar corretamente.

É possível reverter uma versão de runtime das seguintes formas:

- Usando o modo de atualização de runtime Manual
- Usando versões de funções publicadas

Para obter mais informações, consulte Introdução aos controles de gerenciamento de runtime do AWS Lambda no Blog de computação da AWS.

Reverter uma versão de runtime usando o modo de atualização de runtime Manual

Se você estiver usando o modo de atualização de versão de runtime Auto (Automático) ou estiver usando a versão de runtime `$LATEST`, poderá reverter a versão de runtime usando o modo Manual. Para a versão de função que você deseja reverter, altere o modo de atualização de versão de runtime para Manual e especifique o ARN da versão de runtime anterior. Para obter mais informações sobre como localizar o ARN da versão de runtime anterior, consulte Identificação de alterações de versão de runtime.

> **Note**
>
> Se a versão `$LATEST` de sua função estiver configurada para usar o modo Manual, não será possível alterar a arquitetura da CPU ou a versão de runtime que sua função usa. Para

realizar essas alterações, você deve alterar para os modos Auto (Automático) ou Function update (Atualização da função).

Reverter uma versão de runtime usando versões de funções publicadas

As versões de função publicadas correspondem a um snapshot imutável do código da função `$LATEST` e da configuração no momento em que você os criou. No modo Auto (Automático), o Lambda atualiza automaticamente a versão de runtime das versões de funções publicadas durante a segunda fase da implantação da versão de runtime. No modo Function update (Atualização da função), o Lambda não atualiza a versão de runtime das versões de função publicadas.

Portanto, as versões de função publicadas usando o modo Function update (Atualização da função) criam um snapshot estático do código da função, da configuração e da versão de runtime. Ao usar o modo Function update (Atualização da função) com versões de função, é possível sincronizar as atualizações de runtime com as implantações. Também é possível coordenar a reversão do código, da configuração e das versões de runtime ao redirecionar o tráfego para uma versão de função publicada anteriormente. É possível integrar essa abordagem na integração e entrega contínuas (CI/CD) para reversão totalmente automática no caso raro de incompatibilidade de atualização de runtime. Ao usar essa abordagem, é necessário atualizar a função regularmente e publicar novas versões de função para obter as atualizações de runtime mais recentes. Para ter mais informações, consulte Modelo de responsabilidade compartilhada.

Identificação de alterações de versão de runtime

O número da versão do tempo de execução e o ARN são registrados na linha de `INIT_START` registro, que o Lambda emite para o CloudWatch Logs toda vez que cria um novo ambiente de execução. Como o ambiente de execução usa uma versão de runtime semelhante para todas as invocações de função, o Lambda emite a linha de log `INIT_START` somente quando o Lamba executa a fase de inicialização. O Lambda não emite essa linha de log para cada invocação de função. O Lambda emite a linha de registro para CloudWatch Logs, mas ela não é visível no console.

Example Exemplo de linha de log “INIT_START”

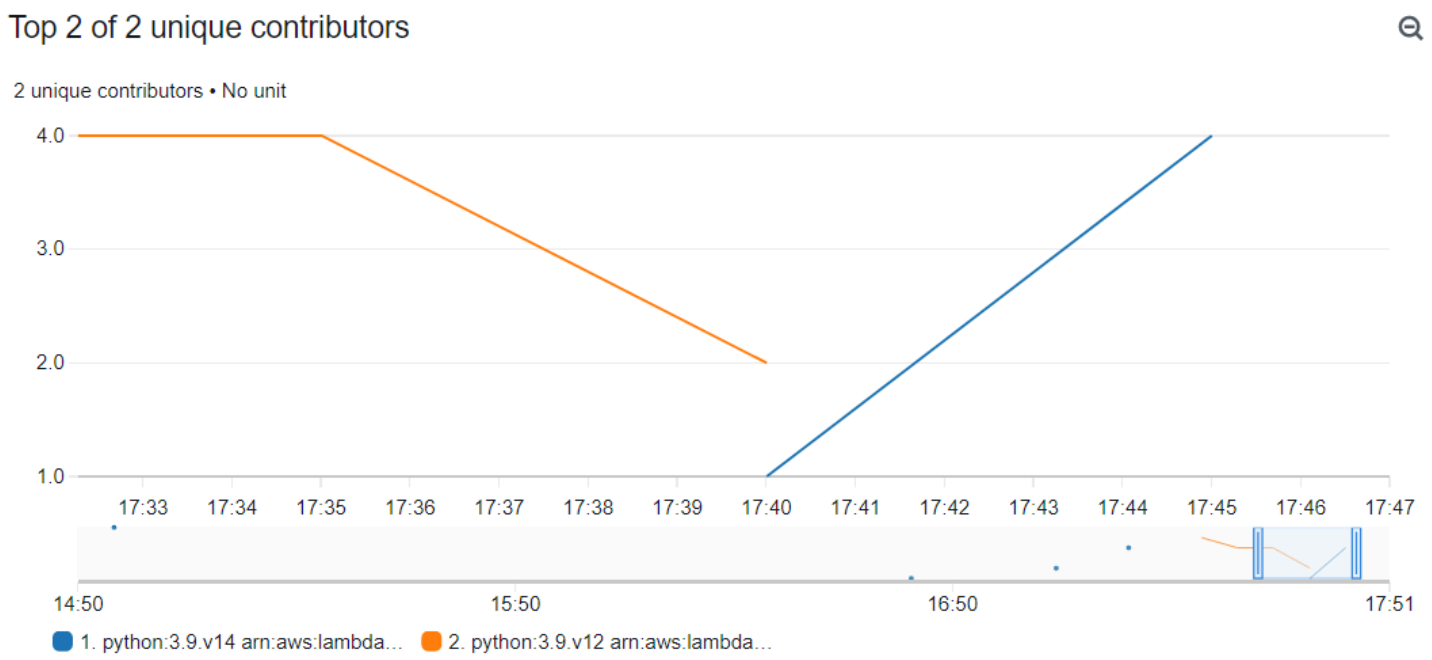
```
INIT_START Runtime Version: python:3.9.v14    Runtime Version ARN: arn:aws:lambda:eu-
south-1::runtime:7b620fc2e66107a1046b140b9d320295811af3ad5d4c6a011fad1fa65127e9e6I
```

Em vez de trabalhar diretamente com os registros, você pode usar o Amazon CloudWatch Contributor Insights para identificar transições entre as versões de tempo de execução. A regra a

seguir contabiliza as versões de runtime distintas de cada linha de log `INIT_START`. Para usar a regra, substitua o nome do grupo de logs de exemplo `/aws/lambda/*` pelo prefixo apropriado para sua função ou grupo de funções.

```
{
  "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"
  }
}
```

O relatório do CloudWatch Contributor Insights a seguir mostra um exemplo de uma transição de versão em tempo de execução, conforme capturado pela regra anterior. A linha laranja mostra a inicialização do ambiente de execução para a versão de runtime anterior (python:3.9.v12) e a linha azul mostra a inicialização do ambiente de execução para a nova versão de runtime (python:3.9.v14).



Definição das configurações de gerenciamento de runtime

É possível definir as configurações de gerenciamento de runtime usando o console do Lambda ou a AWS Command Line Interface (AWS CLI).

> **Note**
>
> Você pode definir as configurações de gerenciamento de runtime separadamente para cada versão de função.

Para configurar como o Lambda atualiza a versão de runtime (console)

1. Abra a página Funções do console do Lambda.
2. Escolha o nome de uma função.
3. Na guia Code (Código), em Runtime settings (Configurações de runtime), escolha Edit runtime management configuration (Editar configuração de gerenciamento de runtime).
4. Em Configuração de gerenciamento de runtime, escolha uma das seguintes opções:
   - Para que a função seja atualizada automaticamente para a versão de runtime mais recente, escolha Auto (Automático).

- Para que a função seja atualizada para a versão de runtime mais recente ao alterar a função, escolha Function update (Atualização da função).
- Para que a função seja atualizada para a versão de runtime mais recente somente ao alterar o ARN da versão de runtime, escolha Manual.

> **Note**
>
> É possível encontrar o ARN da versão de runtime em Runtime management configuration (Configuração de gerenciamento de runtime). Também é possível encontrar o ARN na linha `INIT_START` de seus logs de função.

5. Escolha Salvar.

Para configurar como o Lambda atualiza a versão de runtime (AWS CLI)

Para configurar o gerenciamento de runtime para uma função, é possível usar o comando `put-runtime-management-config` da AWS CLI juntamente com o modo de atualização de runtime. Ao usar o modo `Manual`, você também deve fornecer o ARN da versão de 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
```

Você deve ver saída semelhante a:

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

Modelo de responsabilidade compartilhada

O Lambda é responsável por selecionar e publicar atualizações de segurança para todos os runtimes gerenciados e imagens de contêiner com suporte. A responsabilidade pela atualização das funções existentes para usar a versão de runtime mais recente varia de acordo com o modo de atualização de runtime usado.

O Lambda é responsável por aplicar atualizações de runtime a todas as funções configuradas para usar o modo de atualização de runtime Auto (Automático).

Para funções configuradas com o modo de atualização de runtime Function update (Atualização da função), você é responsável por atualizar regularmente a função. O Lambda é responsável por aplicar as atualizações de runtime quando você faz essas atualizações. Se você não atualizar sua função, o Lambda não atualizará o runtime. Se você não atualiza regularmente sua função, recomendamos fortemente configurá-la para atualizações automáticas de runtime a fim de que continue recebendo atualizações de segurança.

Para funções configuradas para usar o modo de atualização de runtime Manual, você é responsável por atualizar a função para usar a versão de runtime mais recente. É altamente recomendável que você use esse modo somente para reverter a versão de runtime como uma mitigação temporária no caso raro de incompatibilidade de atualização de runtime. Também é recomendado que você altere para o modo Auto (Automático) o mais rápido possível para minimizar o tempo em que suas funções não são corrigidas.

Se você estiver usando imagens de contêiner para implantar as funções, o Lambda será responsável por publicar imagens de base atualizadas. Nesse caso, você é responsável por recriar a imagem de contêiner da função a partir da imagem de base mais recente e implantar novamente a imagem de contêiner.

Isso está resumido na seguinte tabela:

Modo de implantação	Responsabilidade do Lambda	Responsabilidade do cliente
Runtime gerenciado, modo Auto (Automático)	Publicar novas versões de runtime contendo os patches mais recentes. Aplicar os patches de runtime para as funções existentes.	Reverter para uma versão de runtime anterior no caso raro de um problema de compatibilidade de atualização de runtime.
Runtime gerenciado, modo Function update	Publicar novas versões de runtime contendo os patches mais recentes.	Atualizar as funções regularmente para obter a versão de runtime mais recente.

Modo de implantação	Responsabilidade do Lambda	Responsabilidade do cliente
(Atualização da função)		Alterar uma função para o modo Auto (Automático) quando não estiver atualizando a função regularmente. Reverter para uma versão de runtime anterior no caso raro de um problema de compatibi lidade de atualização de runtime.
Runtime gerenciado, modo Manual	Publicar novas versões de runtime contendo os patches mais recentes.	Usar esse modo somente para reversão temporária de runtime no caso raro de um problema de compatibilidade de atualização de runtime. Alterar as funções para os modos Auto (Automático) ou Function update (Atualização da função) e para a versão de runtime mais recente o mais rápido possível.
Imagem de contêiner	Publicar novas imagens de contêiner contendo os patches mais recentes.	Reimplantar as funções regularmente usando a imagem de contêiner de base mais recente para obter os patches mais recentes.

Para obter mais informações sobre a responsabilidade compartilhada com a AWS, consulte Modelo de responsabilidade compartilhada no site de Segurança na Nuvem AWS.

Aplicações de alta conformidade

Para atender aos requisitos de aplicação de patches, os clientes do Lambda normalmente contam com atualizações automáticas de runtime. Se sua aplicação estiver sujeita a requisitos estritos de atualização de patches, convém limitar o uso de versões de runtime anteriores. Você pode restringir os controles de gerenciamento de tempo de execução do Lambda usando AWS Identity and Access Management (IAM) para negar aos usuários da sua AWS conta o acesso à operação da PutRuntimeManagementConfigAPI. Esta operação é usada para escolher o modo de atualização de runtime para uma função. Negar o acesso a esta operação faz com que todas as funções sejam padronizadas para o modo Auto (Automático). É possível aplicar essa restrição em toda a

organização usando políticas de controle de serviços (SCP). Caso você precise reverter uma função para uma versão anterior do tempo de execução, você pode conceder uma exceção de política case-by-case com base nisso.

Modificar o ambiente de runtime

É possível usar extensões internas para modificar o processo de runtime. As extensões internas não são processos separados. Elas são executadas como parte do processo de runtime.

O Lambda fornece específicos do idiomaVariáveis de ambiente que você pode definir para adicionar opções e ferramentas ao runtime. O Lambda também fornece scripts wrapper, que permitem ao Lambda delegar o startup do runtime ao seu script. Você pode criar um script wrapper para personalizar o comportamento de startup do runtime.

Variáveis de ambiente específicas de linguagem

O Lambda é compatível com formas somente de configuração para permitir que o código seja pré-carregado durante a inicialização da função por meio das seguintes variáveis de ambiente específicas do idioma:

- `JAVA_TOOL_OPTIONS`: no Java, o Lambda é compatível com essa variável de ambiente para definir variáveis da linha de comando adicionais no Lambda. Essa variável de ambiente permite especificar a inicialização de ferramentas, especificamente o lançamento de agentes de linguagem de programação nativa ou Java usando as opções `agentlib` ou `javaagent`. Para obter mais informações, consulte Variável de ambiente `JAVA_TOOL_OPTIONS`.
- `NODE_OPTIONS`: disponível em runtimes do Node.js.
- `DOTNET_STARTUP_HOOKS`: no .NET Core 3.1 e superior, essa variável de ambiente especifica um caminho para um assembly (dll) que o Lambda pode usar.

Usar variáveis de ambiente específicas de linguagem é a maneira preferida de definir propriedades de startup.

Scripts wrapper

Você pode criar um script wrapper para personalizar o comportamento de startup do runtime da função do Lambda. Um script wrapper permite que você defina parâmetros de configuração que não podem ser definidos por meio de variáveis de ambiente específicas de linguagem.

> **Note**
>
> As chamadas podem falhar se o script wrapper não iniciar com êxito o processo de runtime.

Os scripts wrapper são compatíveis com todos os runtimes nativos do Lambda. Os scripts wrapper não são compatíveis com Runtimes somente para sistema operacional (a família de runtime `provided`).

Quando você usa um script wrapper para sua função, o Lambda inicia o runtime usando seu script. O Lambda envia ao script o caminho para o intérprete e todos os argumentos originais para o startup padrão do runtime. O script pode estender ou transformar o comportamento de startup do programa. Por exemplo, o script pode injetar e alterar argumentos, definir variáveis de ambiente ou capturar métricas, erros e outras informações de diagnóstico.

Você especifica o script definindo o valor da variável de ambiente `AWS_LAMBDA_EXEC_WRAPPER` como o caminho do sistema de arquivos de um binário ou script executável.

Exemplo: criar e usar um script wrapper com o Python 3.8

No exemplo a seguir, você cria um script wrapper para iniciar o interpretador Python com a opção `-X importtime`. Quando você executa a função, o Lambda gera uma entrada de log para mostrar a duração de cada importação.

Como criar e usar um script wrapper com o Python 3.8

1. Para criar o script wrapper, cole o seguinte código em um arquivo chamado `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. Para conceder permissões executáveis ao script, insira `chmod +x importtime_wrapper` pela linha de comando.

3. Implante o script como uma camada do Lambda.
4. Crie uma função usando o console do Lambda.
   a. Abra o console do lambda.
   b. Escolha a opção Criar função.
   c. Em Basic information (Informações básicas), em Function name (Nome da função), insira **`wrapper-test-function`**.
   d. ParaRuntime, escolha Python 3.8.
   e. Escolha Criar Função.
5. Adicione a camada à função.
   a. Escolha sua função e, em seguida, escolha Código se ela ainda não estiver selecionada.
   b. Escolha Add a layer.
   c. Em Choose a layer (Escolher uma camada), selecione o Name (Nome) e a Version (Versão) da camada compatível criada anteriormente.
   d. Escolha Add.
6. Adicione o código e a variável de ambiente à função.
   a. No editor de código da função, cole o seguinte código de função:

```
import json

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

   b. Escolha Salvar.
   c. Em Environment variables (Variáveis de ambiente), selecione Edit (Editar).
   d. Escolha Add environment variable (Adicionar variável de ambiente).
   e. Em Chave, digite `AWS_LAMBDA_EXEC_WRAPPER`.
   f. Em Valor, insira `/opt/importtime_wrapper`.
   g. Escolha Salvar.

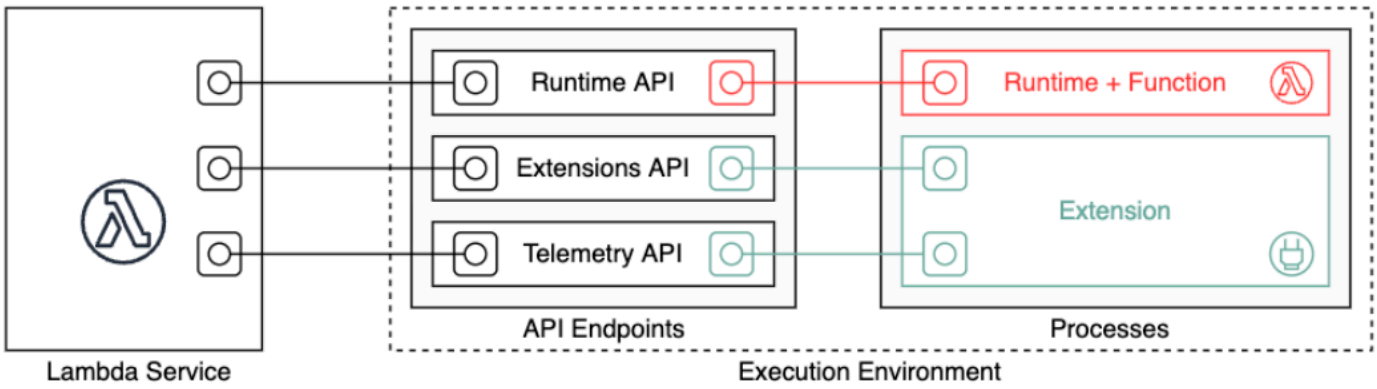
7. Para executar a função, selecione Test (Testar).

   Como o script wrapper iniciou o interpretador Python com a opção `-X importtime`, os logs mostram o tempo necessário para cada importação. Por exemplo:

```
...
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 de tempo de execução do Lambda

O AWS Lambda fornece uma API HTTP para tempos de execução personalizados para receber eventos de invocação do Lambda e enviar dados de resposta de volta para o ambiente de execução do Lambda.



A especificação OpenAPI para a versão da API de tempo de execução 2018-06-01 está disponível aqui: runtime-api.zip

Para criar um URL de solicitação de API, os tempos de execução obtêm o endpoint da API da variável de ambiente do `AWS_LAMBDA_RUNTIME_API`, adiciona a versão da API e o caminho de recurso desejado.

Example Solicitação

```
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next"
```

Métodos de API

- Próxima invocação
- Resposta de invocação
- Erro de inicialização
- Erro de invocação

Próxima invocação

Caminho: `/runtime/invocation/next`

Método – GET

O tempo de execução envia essa mensagem ao Lambda para solicitar um evento de invocação. O corpo da resposta contém a carga útil da invocação, que é um documento JSON que contém os dados do evento do acionador da função. Os cabeçalhos de resposta contêm dados adicionais sobre a invocação.

Cabeçalhos de resposta

- `Lambda-Runtime-Aws-Request-Id`: o ID da solicitação, que identifica a solicitação que acionou a invocação da função.

  Por exemplo, `8476a536-e9f4-11e8-9739-2dfe598c3fcd`.
- `Lambda-Runtime-Deadline-Ms`: a data em que a função expira tempo em milissegundos do Unix.

  Por exemplo, `1542409706888`.
- `Lambda-Runtime-Invoked-Function-Arn`: o ARN da função do Lambda, versão ou alias especificado na invocação.

  Por exemplo, `arn:aws:lambda:us-east-2:123456789012:function:custom-runtime`.
- `Lambda-Runtime-Trace-Id`: o cabeçalho de rastreamento do AWS X-Ray.

  Por exemplo, `Root=1-5bef4de7-ad49b0e87f6ef6c87fc2e700;Parent=9a9197af755a6419;Sampled=1`.
- `Lambda-Runtime-Client-Context`: para invocações do AWS Mobile SDK, os dados sobre a aplicação cliente e o dispositivo.
- `Lambda-Runtime-Cognito-Identity`: para invocações do AWS Mobile SDK, os dados sobre o provedor de identidade do Amazon Cognito.

Não defina um tempo limite na solicitação `GET`, pois a resposta poderá estar atrasada. Entre o momento em que o Lambda inicializa o tempo de execução e o momento em que o tempo de execução tem um evento para retornar, o processo do tempo de execução pode ficar congelado por vários segundos.

O ID da solicitação rastreia a invocação dentro do Lambda. Use-o para especificar a invocação ao enviar a resposta.

O cabeçalho de rastreamento contém o ID de rastreamento, o ID pai e a decisão de amostragem. Se a solicitação for de amostra, a amostra da solicitação foi feita pelo Lambda ou um serviço upstream.

O tempo de execução deve definir o `_X_AMZN_TRACE_ID` com o valor do cabeçalho. O X-Ray SDK lê isso para obter os IDs e determinar se deve rastrear a solicitação.

Resposta de invocação

Caminho: `/runtime/invocation/AwsRequestId/response`

Método – POST

Depois que a função for executada até a conclusão, o tempo de execução envia uma resposta de invocação para o Lambda. Para invocações síncronas, o Lambda envia a resposta de volta para o cliente.

Example solicitação com êxito

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

Erro de inicialização

Se a função retornar um erro ou o tempo de execução encontrar um erro durante a inicialização, o tempo de execução usará esse método para relatar o erro ao Lambda.

Caminho: `/runtime/init/error`

Método – POST

Cabeçalhos

`Lambda-Runtime-Function-Error-Type`: o tipo de erro encontrado pelo tempo de execução. Obrigatório: não

Este cabeçalho consiste em um valor de string. Lambda aceita qualquer string, mas recomendamos o formato <category.reason>. Por exemplo: .

- Tempo de execução. NoSuchHandler
- Tempo de execução.API KeyNotFound
- Tempo de execução. ConfigInvalid

- Tempo de execução. UnknownReason

Body parameters (Parâmetros do corpo)

`ErrorRequest`: informações adicionais sobre o erro. Obrigatório: não

Este campo é um objeto JSON com a seguinte estrutura:

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

Observe que o Lambda aceita qualquer valor para `errorType`.

O exemplo a seguir mostra uma mensagem de erro de função do Lambda na qual a função não pôde analisar os dados do evento fornecidos na chamada.

Example Erro de função

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Parâmetros do corpo da resposta

- `StatusResponse` – String. Informações de status, enviadas com 202 códigos de resposta.
- `ErrorResponse`: informações adicionais de erro, enviadas com os códigos de resposta de erro. ErrorResponse contém um tipo de erro e uma mensagem de erro.

Códigos de resposta

- 202: aceito
- 403: proibido
- 500: erro de contêiner. Estado não recuperável. O tempo de execução deve sair imediatamente.

Example solicitação com erro de inicialização

```
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"
```

Erro de invocação

Se a função retornar um erro ou o tempo de execução encontrar um erro, o tempo de execução usará esse método para relatar o erro ao Lambda.

Caminho: `/runtime/invocation/`*`AwsRequestId`*`/error`

Método – POST

Cabeçalhos

`Lambda-Runtime-Function-Error-Type`: o tipo de erro encontrado pelo tempo de execução. Obrigatório: não

Este cabeçalho consiste em um valor de string. Lambda aceita qualquer string, mas recomendamos o formato <category.reason>. Por exemplo: .

- Tempo de execução. NoSuchHandler
- Tempo de execução.API KeyNotFound
- Tempo de execução. ConfigInvalid
- Tempo de execução. UnknownReason

Body parameters (Parâmetros do corpo)

`ErrorRequest`: informações adicionais sobre o erro. Obrigatório: não

Este campo é um objeto JSON com a seguinte estrutura:

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

Observe que o Lambda aceita qualquer valor para `errorType`.

O exemplo a seguir mostra uma mensagem de erro de função do Lambda na qual a função não pôde analisar os dados do evento fornecidos na chamada.

Example Erro de função

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Parâmetros do corpo da resposta

- `StatusResponse` – String. Informações de status, enviadas com 202 códigos de resposta.
- `ErrorResponse`: informações adicionais de erro, enviadas com os códigos de resposta de erro. ErrorResponse contém um tipo de erro e uma mensagem de erro.

Códigos de resposta

- 202: aceito
- 400: solicitação inválida
- 403: proibido
- 500: erro de contêiner. Estado não recuperável. O tempo de execução deve sair imediatamente.

Example solicitação com erro

```
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 usar runtimes somente para sistema operacional do Lambda

O Lambda fornece runtimes gerenciados para Java, Python, Node.js, .NET e Ruby. Para criar funções do Lambda em uma linguagem de programação que não esteja disponível como um runtime gerenciado, use um runtime somente para sistema operacional (a família de runtime `provided`). Há três casos de uso principais para runtimes somente para sistema operacional:

- Compilação nativa antecipada (AOT): linguagens, como Go, Rust e C++ compilam nativamente em um binário executável, o que não requer um runtime de linguagem dedicado. Essas linguagens precisam apenas de um ambiente de sistema operacional no qual o binário compilado possa ser executado. Você também pode usar runtimes somente para sistema operacional do Lambda para implantar binários compilados com .NET Native AOT e Java GraalVM Native.

  Você deve incluir um cliente de interface de runtime no binário. O cliente de interface de runtime chama o API de tempo de execução do Lambda para recuperar as invocações da função e, em seguida, chama o manipulador da função. O Lambda fornece clientes de interface de runtime para Go, .NET Native AOT, C++ e Rust (experimental).

  Você deve compilar o binário para um ambiente Linux e para a mesma arquitetura de conjunto de instruções que planeja usar para a função (x86_64 ou arm64).
- Runtimes de terceiros: você pode executar funções do Lambda usando runtimes prontos para uso, como Bref para PHP ou o Swift AWS Lambda Runtime para Swift..
- Runtimes personalizados: você pode criar seu próprio runtime para uma linguagem ou versão de linguagem para a qual o Lambda não fornece um runtime gerenciado, como Node.js 19. Para ter mais informações, consulte Criar um runtime personalizado para AWS Lambda. Esse é o caso de uso menos comum para runtimes somente para sistema operacional.

O Lambda oferece suporte aos seguintes runtimes somente para sistema operacional:

Somente SO

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
Runtime somente para sistema operacional	`provided.al2023`	Amazon Linux 2023			
Runtime somente para sistema operacional	`provided.al2`	Amazon Linux 2			

O runtime do Amazon Linux 2023 (`provided.al2023`) oferece várias vantagens em relação ao Amazon Linux 2, incluindo uma área de implantação menor e versões atualizadas de bibliotecas, como `glibc`.

O runtime `provided.al2023` usa `dnf` como gerenciador de pacotes em vez de `yum`, que é o gerenciador de pacotes padrão no Amazon Linux 2. Para obter mais informações sobre as diferenças entre `provided.al2023` e`provided.al2`, consulte Introducing the Amazon Linux 2023 runtime for AWS Lambda no Blog AWS Compute.

Criar um runtime personalizado para AWS Lambda

Você pode implementar um runtime do AWS Lambda em qualquer linguagem de programação. O runtime é um programa que executa um método de manipulador da função do Lambda quando a função é invocada. Você pode ser incluir o runtime no pacote de implantação da função ou distribuí-lo em uma camada. Ao criar a função do Lambda, escolha um runtime somente do sistema operacional (a família de runtime `provided`).

> **Note**
>
> Criar um runtime personalizado é um caso de uso avançado. Se você estiver procurando informações sobre como compilar em um binário nativo ou usar um runtime pronto para uso de terceiros, consulte Quando usar runtimes somente para sistema operacional do Lambda.

Para obter uma explicação passo a passo do processo de implantação de runtime personalizado, consulte Tutorial: criar um runtime personalizado. Você também pode explorar um runtime personalizado implementado em C++ em awslabs/aws-lambda-cpp no GitHub.

Tópicos

- Requisitos
- Implementação do streaming de resposta em um runtime personalizado

Requisitos

Os runtimes personalizados devem realizar determinadas tarefas de inicialização e processamento. Um runtime executa o código de configuração da função, lê o nome do manipulador de uma variável de ambiente e lê eventos de invocação da API de runtime do Lambda. O runtime transmite os dados de eventos para o manipulador de funções e publica a resposta do manipulador de volta no Lambda.

Tarefas de inicialização

As tarefas de inicialização são executadas uma vez por instância da função a fim de preparar o ambiente para processar invocações.

- Recuperar configurações: leia variáveis de ambiente para obter detalhes sobre a função e o ambiente.
  - `_HANDLER`: a localização do manipulador, da configuração da função. O formato padrão é *`file`*`.`*`method`*, em que `file` é o nome do arquivo sem uma extensão, e `method` é o nome de um método ou da função que é definido no arquivo.
  - `LAMBDA_TASK_ROOT`: o diretório que contém o código da função.
  - `AWS_LAMBDA_RUNTIME_API`: o host e a porta da API do runtime.

  Para obter uma lista completa das variáveis disponíveis, consulte Variáveis de ambiente com runtime definido.
- Inicializar a função: carregar o arquivo do manipulador e executar qualquer código global ou estático que ele contenha. As funções devem criar recursos estáticos como clientes SDK e conexões de banco de dados uma vez e reutilizá-los para várias invocações.
- Lidar com erros: se ocorrer um erro, chame a API erro de inicialização e saia imediatamente.

A inicialização é considerada no runtime e tempo limite cobrados. Quando uma execução aciona a inicialização de uma nova instância da função, é possível ver o tempo de inicialização nos logs e no rastreamento do 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
```

Tarefas de processamento

Enquanto ela é executada, um usa a interface de runtime do Lambda para gerenciar eventos de entrada e relatar erros. Depois de executar as tarefas de inicialização, o runtime processa eventos de entrada em um loop. No código do seu runtime, execute as seguintes etapas em ordem.

- Obter um evento: chame a API próxima invocação para obter o próximo evento. O corpo da resposta contém os dados do evento. Os cabeçalhos da resposta contêm o ID da solicitação e outras informações.
- Propagar o cabeçalho de rastreamento: obtenha o cabeçalho de rastreamento do X-Ray do cabeçalho `Lambda-Runtime-Trace-Id` na resposta da API. Defina a variável de ambiente `_X_AMZN_TRACE_ID` localmente com o mesmo valor. O X-Ray SDK usa esse valor para conectar dados de rastreamento entre serviços.
- Criar um contexto de objeto: crie um objeto com informações de contexto de variáveis do ambiente e cabeçalhos na resposta da API.
- Invocar o manipulador de funções: transmita o evento e o objeto de contexto para o manipulador.
- Processar a resposta: chame a API de resposta de invocação para publicar a resposta do manipulador.
- Processar erros: se ocorrer um erro, chame a API erro de invocação.
- Limpeza: libere recursos não utilizados, envie dados para outros serviços ou execute tarefas adicionais antes de receber o próximo evento.

Entrypoint

O ponto de entrada de um runtime personalizado é um arquivo executável denominado `bootstrap`. O arquivo de bootstrap pode ser o runtime ou ele pode invocar outro arquivo que criará o runtime. Se a raiz do pacote de implantação não contiver um arquivo denominado `bootstrap`, o Lambda

procurará o arquivo nas camadas da função. Se o arquivo `bootstrap` não existir ou não for executável, a função retornará um erro de `Runtime.InvalidEntrypoint` na invocação.

Veja um exemplo em que o arquivo `bootstrap` usa um pacote de versão do Node.js para executar um runtime do JavaScript em um arquivo separado denominado `runtime.js`.

Example bootstrap

```
#!/bin/sh
    cd $LAMBDA_TASK_ROOT
    ./node-v11.1.0-linux-x64/bin/node runtime.js
```

Implementação do streaming de resposta em um runtime personalizado

Para funções de streaming de resposta, os endpoints `response` e `error` têm um comportamento ligeiramente modificado que possibilita que o runtime faça o streaming de respostas parciais para o cliente e retorne cargas úteis em blocos. Para obter mais informações sobre o comportamento específico, consulte o seguinte:

- `/runtime/invocation/AwsRequestId/response`: propaga o cabeçalho `Content-Type` do runtime para enviar ao cliente. O Lambda retorna a carga de resposta em blocos por meio de codificação de transferência em blocos do HTTP/1.1. O stream de resposta pode ter um tamanho máximo de 20 MiB. Para fazer o streaming da resposta para o Lambda, o runtime deve:
  - Definir o cabeçalho HTTP `Lambda-Runtime-Function-Response-Mode` como `streaming`.
  - Defina o cabeçalho `Transfer-Encoding` como `chunked`.
  - Gravar a resposta em conformidade com a especificação de codificação de transferência em blocos HTTP/1.1.
  - Encerrar a conexão subjacente após escrever a resposta com êxito.
- `/runtime/invocation/AwsRequestId/error`: o runtime pode usar esse endpoint para relatar erros de função ou de runtime ao Lambda, que também aceita o cabeçalho `Transfer-Encoding`. Esse endpoint pode ser chamado somente antes que o runtime comece a enviar uma resposta de invocação.
- Relatar erros intermediários usando trailers de erro em `/runtime/invocation/AwsRequestId/response`: para relatar erros ocorridos após o runtime começar a gravar a resposta da invocação, o runtime pode, opcionalmente, anexar cabeçalhos HTTP no final, chamados de `Lambda-Runtime-Function-Error-Type` e `Lambda-Runtime-Function-`

`Error-Body`. O Lambda trata isso como uma resposta com êxito e encaminha os metadados de erro fornecidos pelo runtime ao cliente.

> **Note**
>
> Para vincular cabeçalhos à direita, o runtime deve definir o valor do cabeçalho `Trailer` no início da solicitação HTTP. Isso é um requisito da especificação de codificação de transferência em blocos do HTTP/1.1.

- `Lambda-Runtime-Function-Error-Type`: o tipo de erro encontrado pelo runtime. Este cabeçalho consiste em um valor de string. O Lambda aceita qualquer string, mas recomendamos um formato *`<category.reason>`*. Por exemplo, `Runtime.APIKeyNotFound`.
- `Lambda-Runtime-Function-Error-Body`: informações sobre o erro codificadas em Base64.

Tutorial: criar um runtime personalizado

Neste tutorial, você criará uma função do Lambda com um runtime personalizado. Você começa incluindo o runtime no pacote de implantação da função. Em seguida, você o migra para uma camada gerenciada de forma independente da função. Por fim, você compartilha a camada de runtime com o mundo atualizando sua política de permissões com base em recursos.

Pré-requisitos

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em Criar uma função do Lambda com o console para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, a AWS Command Line Interface (AWS CLI) versão 2 será necessária. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

A seguinte saída deverá ser mostrada:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell.

> **Note**
>
> No Windows, alguns comandos da CLI do Bash que você costuma usar com o Lambda (como `zip`) não são compatíveis com os terminais integrados do sistema operacional. Para obter uma versão do Ubuntu com o Bash integrada no Windows, instale o Subsistema do Windows para Linux. Os exemplos de comandos da CLI neste guia usam a formatação Linux. Os comandos que incluem documentos JSON em linha deverão ser reformatados se você estiver usando a CLI do Windows.

Você precisa de uma função do IAM para criar uma função do Lambda. A função precisa de permissão para enviar registros ao CloudWatch Logs e acessar os AWS serviços que sua função usa. Se você ainda não tiver uma função para o desenvolvimento de funções, crie uma agora.

Para criar uma função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione Create role (Criar função).
3. Crie uma função com as propriedades a seguir.

   - Entidade confiável–Lambda.
   - Permissões — AWSLambdaBasicExecutionRole.
   - Role name (Nome da função): **`lambda-role`**.

   A AWSLambdaBasicExecutionRolepolítica tem as permissões que a função precisa para gravar registros em CloudWatch Logs.

Criar uma função

Criar uma função do Lambda com um runtime personalizado. Este exemplo inclui dois arquivos: um arquivo `bootstrap` de runtime e um manipulador de funções. Ambos são implementados em Bash.

1. Crie um diretório para o projeto e depois mude para esse diretório.

```
mkdir runtime-tutorial
cd runtime-tutorial
```

2. Crie um novo arquivo chamado `bootstrap`. Esse é o runtime personalizado.

   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
```

   O runtime carrega um script de função do pacote de implantação. Ele usa duas variáveis para localizar o script. `LAMBDA_TASK_ROOT` informa onde o pacote foi extraído e `_HANDLER` inclui o nome do script.

   Depois que o runtime carrega o script da função, ele usa a API de runtime para recuperar um evento de invocação do Lambda, transmite o evento para o manipulador e publica a resposta de volta no Lambda. Para obter o ID de solicitação, o runtime salva os cabeçalhos na resposta

da API em um arquivo temporário e lê o cabeçalho `Lambda-Runtime-Aws-Request-Id` do arquivo.

> **Note**
>
> Os runtimes têm responsabilidades adicionais, incluindo o processamento de erros e o fornecimento de informações de contexto para o manipulador. Para obter detalhes, consulte Requisitos.

3. Crie um script para a função. O exemplo de script a seguir define uma função do manipulador que usa dados de eventos, os registra no log `stderr` e os retorna.

   Example function.sh

   ```
   function handler () {
     EVENT_DATA=$1
     echo "$EVENT_DATA" 1>&2;
     RESPONSE="Echoing request: '$EVENT_DATA'"

     echo $RESPONSE
   }
   ```

   Agora, diretório `runtime-tutorial` será desta forma:

   ```
   runtime-tutorial
   # bootstrap
   # function.sh
   ```

4. Torne os arquivos executáveis e os adicione a um arquivo .zip. Esse é o pacote de implantação.

   ```
   chmod 755 function.sh bootstrap
   zip function.zip function.sh bootstrap
   ```

5. Crie uma função chamada `bash-runtime`. Para `--role`, insira o ARN do seu perfil de execução do 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. Invoque a função.

```
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}'
 response.txt --cli-binary-format raw-in-base64-out
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

Você obterá uma resposta parecida com esta:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

7. Verifique a resposta.

```
cat response.txt
```

Você obterá uma resposta parecida com esta:

```
Echoing request: '{"text":"Hello"}'
```

Criar uma camada

Para separar o código do runtime do código da função, crie uma camada que contenha apenas o runtime. As camadas permitem que você desenvolva dependências da sua função de forma independente e podem reduzir o uso de armazenamento quando você usa a mesma camada com várias funções. Para ter mais informações, consulte Gerenciar dependências do Lambda com camadas.

1. Crie um arquivo .zip que contenha o arquivo `bootstrap`.

```
zip runtime.zip bootstrap
```

2. Crie uma camada com o comando publish-layer-version.

```
aws lambda publish-layer-version --layer-name bash-runtime --zip-file fileb://
runtime.zip
```

Isso cria a primeira versão da camada.

Atualizar a função

Para usar a camada de runtime com a função, configure a função para usar a camada e remova o código de runtime da função.

1. Atualize a configuração de função para extrair na camada.

   ```
   aws lambda update-function-configuration --function-name bash-runtime \
   --layers arn:aws:lambda:us-east-1:123456789012:layer:bash-runtime:1
   ```

   Isso adiciona o runtime à função no diretório `/opt`. Para garantir que o Lambda use o runtime na camada, você precisa remover o `boostrap` do pacote de implantação da função, conforme mostrado nas próximas duas etapas.
2. Crie um arquivo .zip que contenha o código da função.

   ```
   zip function-only.zip function.sh
   ```
3. Atualize o código da função para incluir apenas o script do manipulador.

   ```
   aws lambda update-function-code --function-name bash-runtime --zip-file fileb://
   function-only.zip
   ```
4. Invoque a função para confirmar se ela funciona com a camada de runtime.

   ```
   aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}'
    response.txt --cli-binary-format raw-in-base64-out
   ```

   A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

   Você obterá uma resposta parecida com esta:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

5. Verifique a resposta.

```
cat response.txt
```

Você obterá uma resposta parecida com esta:

```
Echoing request: '{"text":"Hello"}'
```

Atualizar o runtime

1. Para registrar informações sobre o ambiente de execução, atualize o script de runtime para gerar variáveis de ambiente.

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. Crie um arquivo .zip que contenha a nova versão do arquivo `bootstrap`.

```
zip runtime.zip bootstrap
```

3. Crie uma nova versão da camada `bash-runtime`.

```
aws lambda publish-layer-version --layer-name bash-runtime --zip-file fileb://
runtime.zip
```

4. Configure a função para usar a nova versão da camada.

```
aws lambda update-function-configuration --function-name bash-runtime \
--layers arn:aws:lambda:us-east-1:123456789012:layer:bash-runtime:2
```

Compartilhar a camada

Para conceder permissão de uso de camadas a outra conta, adicione uma instrução à política de permissões da versão da camada usando o comando add-layer-version-permission. Em cada instrução, você pode conceder permissão a uma única conta, a todas as contas ou a uma organização.

O exemplo a seguir concede à conta 111122223333 acesso à versão 2 da camada `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
```

Você deve ver saída semelhante a:

```
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"}
```

As permissões se aplicam apenas a uma única versão de camada. Repita o processo sempre que criar uma nova versão da camada.

Limpeza

Exclua cada versão da camada.

```
aws lambda delete-layer-version --layer-name bash-runtime --version-number 1
aws lambda delete-layer-version --layer-name bash-runtime --version-number 2
```

Como a função contém uma referência à versão 2 da camada, ela ainda existe no Lambda. A função continua a funcionar, mas as funções não podem mais ser configuradas para usar a versão excluída. Se você modificar a lista de camadas na função, precisará especificar uma nova versão ou omitir a camada excluída.

Exclua a função com o comando delete-function.

```
aws lambda delete-function --function-name bash-runtime
```

Usar a vetorização AVX2 no Lambda

A AVX2 (Advanced Vector Extensions 2) é uma extensão de vetorização para o conjunto de instruções Intel x86 que pode desempenhar instruções de SIMD (Single Instruction, Multiple Data) sobre vetores de 256 bits. Para algoritmos vetorizáveis com operação altamente paralelizável, o uso da AVX2 pode melhorar a performance da CPU, resultando em latências mais baixas e maior throughput. Use o conjunto de instruções da AVX2 para cargas de trabalho com uso intensivo de computação, como inferência de machine learning, processamento multimídia, simulações científicas e aplicações de modelagem financeira.

> **Note**
>
> O Lambda arm64 usa a arquitetura NEON SIMD e oferece suporte a extensões X86 AVX2.

Para usar o AVX2 com sua função do Lambda, seu código de função deve acessar o código otimizado para AVX2. Em algumas linguagens, você pode instalar a versão de bibliotecas e pacotes compatível com a AVX2. Para outras linguagens, você pode recompilar o código e as dependências com os sinalizadores do compilador apropriados definidos (se o compilador aceitar vetorização automática). Você também pode compilar seu código com bibliotecas de terceiros que usam AVX2 para otimizar operações matemáticas. Por exemplo, Intel Math Kernel Library (Intel MKL), OpenBLAS (Basic Linear Algebra Subprograms) e AMD BLAS-like Library Instantiation Software (BLIS). Linguagens autovetorizadas, como Java, usam a AVX2 para cálculos automaticamente.

Você pode criar novas cargas de trabalho do Lambda ou mover cargas de trabalho habilitadas para AVX2 existentes para o Lambda sem nenhum custo adicional.

Para obter mais informações sobre a AVX2, consulte Extensões de vetor avançadas 2 na Wikipédia.

Compilar a partir da origem

Se sua função do Lambda usar uma biblioteca C ou C++ para executar operações vetorizáveis com uso alto uso computação, você poderá definir os sinalizadores do compilador apropriados e recompilar o código da função. Em seguida, o compilador vetorizará o código automaticamente.

Para o compilador `gcc` ou `clang`, adicione `-march=haswell` ao comando ou defina `-mavx2` como uma opção de comando.

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

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

Para usar uma biblioteca específica, siga as instruções na documentação da biblioteca para realizar a compilação e a criação dela. Por exemplo, para criar a TensorFlow partir do código-fonte, você pode seguir as instruções de instalação no TensorFlow site. Use a opção de compilação `-march=haswell`.

Ativar a AVX2 para Intel MKL

A Intel MKL é uma biblioteca de operações matemáticas otimizadas que usam implicitamente instruções AVX2 quando a plataforma de computação oferece compatibilidade. Frameworks como PyTorch compilados com Intel MKL por padrão, então você não precisa habilitar o AVX2.

Algumas bibliotecas, como a TensorFlow, oferecem opções em seu processo de criação para especificar a otimização do Intel MKL. Por exemplo, com TensorFlow, use a `--config=mkl` opção.

Você também pode criar bibliotecas científicas populares em Python, como SciPy e NumPy, com o Intel MKL. Para obter instruções sobre como criar essas bibliotecas com a Intel MKL, consulte Numpy/Scipy com Intel MKL e Intel Compilers no site da Intel.

Para obter mais informações sobre o Intel MKL e bibliotecas similares, consulte Math Kernel Library na Wikipedia, o site do OpenBLAS e o repositório AMD BLIS em. GitHub

Compatibilidade com AVX2 em outras linguagens

Se você não usar bibliotecas C ou C++ e não criar com a Intel MKL, ainda poderá ter certa melhoria de performance da AVX2 em suas aplicações. Observe que a melhoria real depende da capacidade do compilador ou intérprete de utilizar os recursos AVX2 em seu código.

Python

Os usuários do Python geralmente usam NumPy bibliotecas para cargas de SciPy trabalho com uso intensivo de computação. Você pode compilar essas bibliotecas para habilitar a AVX2 ou usar as versões das bibliotecas habilitadas para Intel MKL.

Nó

Para cargas de trabalho com uso intensivo de computação, use as versões habilitadas para AVX2 ou Intel MKL das bibliotecas de que você precisa.

Java

O compilador JIT do Java pode vetorizar automaticamente seu código para execução com instruções da AVX2. Para obter informações sobre como detectar código vetorizado, consulte a apresentação Code vectorization in the JVM no site OpenJDK.

Go

O compilador padrão do Go não tem compatibilidade com vetorização automática atualmente, mas você pode usar o gccgo, o compilador GCC para Go. Defina a opção do `-mavx2`:

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

Intrínsecos

É possível usar funções intrínsecas em várias linguagens para vetorizar seu código manualmente para usar a AVX2. No entanto, não recomendamos essa abordagem. Gravar código vetorizado manualmente requer um esforço significativo. Além disso, depurar e manter esse código é mais difícil do que usar código que dependa da autovetorização.

Configurar funções do AWS Lambda

Aprenda a configurar as principais funcionalidades e opções para a função do Lambda usando a API ou o console do Lambda.

Memória

Saiba como e quando aumentar memória para uma função.

Armazenamento efêmero

Saiba como e quando aumentar a capacidade do armazenamento temporário da função.

Timeout (Tempo limite)

Saiba como e quando aumentar o valor do tempo limite da função.

Variáveis de ambiente

É possível tornar o código da função portátil e manter os segredos externos ao código ao armazená-los na configuração da função usando variáveis de ambiente.

Redes de saída

É possível usar a função do Lambda com recursos da AWS em uma Amazon VPC. Conectar a função a uma VPC permite acessar recursos em uma sub-rede privada, como bancos de dados relacionais e caches.

Redes de entrada

É possível usar um endpoint da VPC de interface para invocar as funções do Lambda sem passar pela Internet pública.

Sistema de arquivos

É possível usar a função do Lambda para montar um Amazon EFS em um diretório local. Um sistema de arquivos permite que o código da função acesse e modifique recursos compartilhados com segurança e alta simultaneidade.

Aliases

É possível configurar seus clientes para invocar uma versão específica da função do Lambda ao usar um alias, em vez de atualizar o cliente.

Versões

Ao publicar uma versão da função, é possível armazenar o código e a configuração como um recurso separado que não pode ser alterado.

Streaming de respostas

É possível configurar seus URLs de função do Lambda para fazer o streaming de cargas de resposta de volta aos clientes. O streaming de respostas pode beneficiar aplicações sensíveis à latência ao melhorar o desempenho do tempo até o primeiro byte (TTFB). Isso ocorre porque é possível enviar respostas parciais de volta ao cliente assim que elas se tornarem disponíveis. Além disso, é possível usar o streaming de respostas para construir funções que retornem cargas maiores.

Configurar memória para uma função do Lambda

O Lambda aloca capacidade da CPU na proporção da quantidade de memória configurada. Memória é a quantidade de memória disponível para a função do Lambda no runtime. Você pode aumentar ou diminuir a memória e a capacidade da CPU alocadas para a função usando a configuração Memória. Você pode configurar a memória com um valor de 128 MB a 10.240 MB, em incrementos de 1 MB. Com 1.769 MB, uma função tem o equivalente a um vCPU (um vCPU-segundo de créditos por segundo).

Esta página descreve como e quando atualizar a configuração de memória para uma função do Lambda.

Seções

Determinar a configuração de memória apropriada para uma função do Lambda

A memória é a principal alavanca de controle de performance de uma função. A configuração padrão, 128 MB, é a configuração mais baixa possível. Recomendamos que você só use 128 MB para funções simples do Lambda, por exemplo, as que transformam e roteiam eventos para outros serviços da AWS. Uma alocação maior de memória pode melhorar a performance das funções que usam bibliotecas importadas, camadas do Lambda, o Amazon Simple Storage Service (Amazon S3) ou o Amazon Elastic File System (Amazon EFS). Adicionar memória aumenta proporcionalmente a quantidade de CPU, aumentando a capacidade computacional geral disponível. Se uma função depender de CPU, rede ou memória, aumentar a configuração de memória pode melhorar muito sua performance.

Para encontrar a configuração de memória correta para as funções, recomendamos que você use a ferramenta de código aberto AWS Lambda Power Tuning. Essa ferramenta usa o AWS Step Functions para executar várias versões de uma função do Lambda com diferentes alocações de

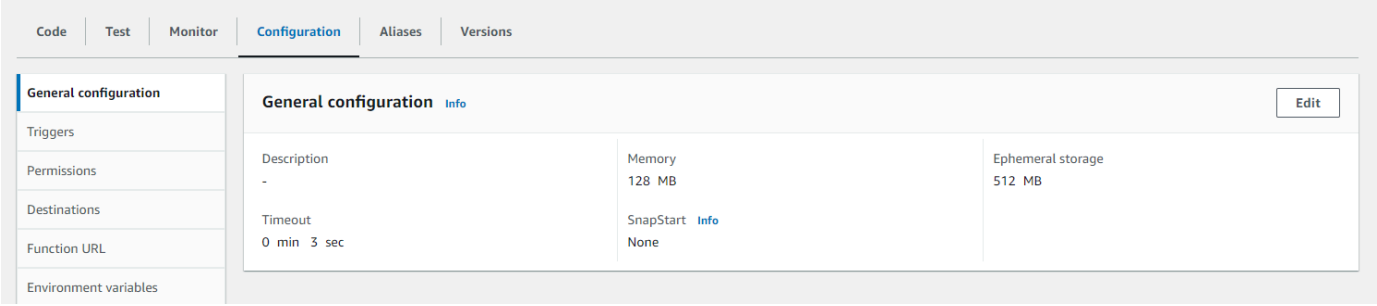
memória simultaneamente e medir a performance. A função entrada é executada em sua conta da AWS, realizando chamadas de HTTP e interação com o SDK em tempo real para medir a provável performance em um cenário de produção real. Você também pode implementar um processo de CI/CD para usar essa ferramenta para medir automaticamente a performance das novas funções que implantar.

Configurar a memória da função (console)

Você pode configurar a memória da sua função no console do Lambda.

Para atualizar a memória de uma função

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Configuração e depois selecione Configuração geral.



4. Em Configuração geral, escolha Editar.
5. Em Memória, defina um valor de 128 MB a 10.240 MB.
6. Escolha Salvar.

Configurar memória para uma função (AWS CLI)

Você pode usar o comando update-function-configuration para configurar a memória para a sua função.

Example

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

Configurar memória para uma função (AWS SAM)

Você pode usar o AWS Serverless Application Model para configurar a memória para a sua função. Atualize a propriedade MemorySize no arquivo `template.yaml` e execute o comando 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...
```

Aceitar recomendações de memória de função (console)

Se você tiver permissões de administrador no AWS Identity and Access Management (IAM), será possível optar por receber recomendações de memória de funções do Lambda do AWS Compute Optimizer. Para obter instruções sobre como aceitar recomendações de memória para sua conta ou organização, consulte Opting in your account no Manual do usuário do AWS Compute Optimizer.

> **Note**
>
> O Compute Optimizer oferece suporte apenas a funções que usam a arquitetura x86_64.

Quando você tiver aceitado e sua função do Lambda atender aos requisitos do Compute Optimizer, será possível visualizar e aceitar recomendações de memória de funções do Compute Optimizer no console do Lambda em Configuração geral.

Configurar armazenamento efêmero para funções do Lambda

O Lambda fornece armazenamento efêmero para funções no diretório `/tmp`. Esse armazenamento é temporário e exclusivo de cada ambiente de execução. Você pode controlar a quantidade de armazenamento efêmero alocado para a função usando a configuração Armazenamento efêmero. Você pode configurar o armazenamento efêmero com um valor de 512 MB a 10.240 MB, em incrementos de 1 MB. Todos os dados armazenados em `/tmp` são criptografados em repouso com uma chave gerenciada pela AWS.

Esta página descreve os casos de uso comuns e como atualizar o armazenamento efêmero para uma função do Lambda.

Seções

Casos de uso comuns para armazenamento efêmero maior

Aqui estão vários casos de uso comuns que se beneficiam com um armazenamento efêmero maior:

- Trabalhos de extração, transformação e carga (ETL): aumente o armazenamento efêmero quando o código executar computação intermediária ou baixar outros recursos para concluir o processamento. Mais espaço temporário permite a execução de trabalhos de ETL mais complexos em funções do Lambda.
- Inferência de machine learning (ML): muitas tarefas de inferência dependem de grandes arquivos de dados de referência, incluindo bibliotecas e modelos. Com mais armazenamento efêmero, você pode baixar modelos maiores do Amazon Simple Storage Service (Amazon S3) para `/tmp` e usá-los no processamento.
- Processamento de dados: para workloads que baixam objetos do Amazon S3 em resposta a eventos do S3, mais espaço em `/tmp` permite lidar com objetos maiores sem usar processamento em memória. Workloads que criam PDFs ou que processam mídia também se beneficiam com mais armazenamento efêmero.
- Processamento gráfico: o processamento de imagens é um caso de uso comum para aplicações baseadas no Lambda. Para workloads que processam arquivos TIFF ou imagens de satélite

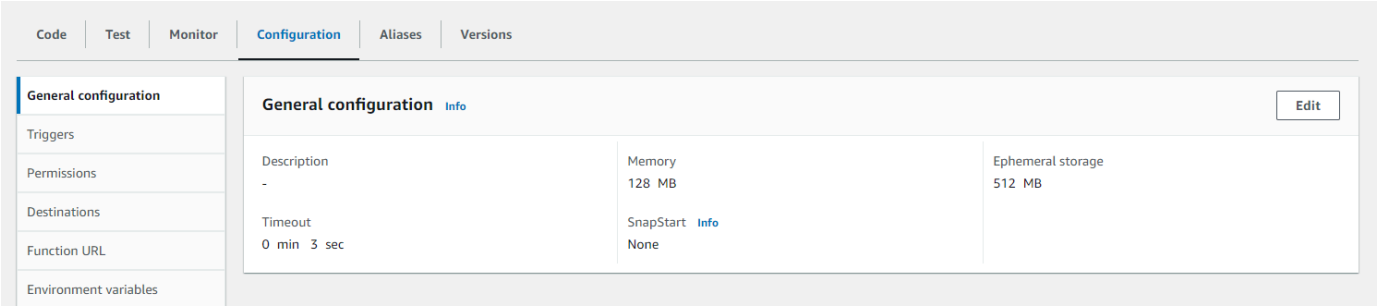
grandes, ter mais armazenamento efêmero torna mais fácil usar bibliotecas e realizar computação no Lambda.

Como configurar o armazenamento temporário (console)

Você pode configurar armazenamento efêmero no console do Lambda.

Para modificar o armazenamento efêmero de uma função

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Configuração e depois selecione Configuração geral.



4. Em Configuração geral, escolha Editar.
5. Para Armazenamento efêmero defina um valor de 512 MB a 10.240 MB, em incrementos de 1 MB.
6. Escolha Salvar.

Configurar armazenamento efêmero (AWS CLI)

Você pode usar o comando update-function-configuration para configurar armazenamento efêmero.

Example

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

Configurar armazenamento efêmero (AWS SAM)

Você pode usar o AWS Serverless Application Model para configurar o armazenamento efêmero da função. Atualize a propriedade EphemeralStorage no arquivo `template.yaml` e execute o comando 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...
```

Configurar tempo limite da função do Lambda

O Lambda executa o código por um determinado período de tempo antes de atingir o tempo limite. O tempo limite é o período de tempo máximo, em segundos, durante o qual uma função do Lambda pode ser executada. O valor padrão dessa configuração é de 3 segundos, mas você pode ajustá-lo em incrementos de 1 segundo até o valor máximo de 900 segundos (15 minutos).

Esta página descreve como e quando atualizar a configuração de tempo limite de uma função do Lambda.

Seções

Determinar a configuração de tempo imite apropriada para uma função do Lambda

Se o valor do tempo limite for próximo da duração média de uma função, há um risco maior de que a função atinja o tempo limite inesperadamente. A duração de uma função pode variar dependendo da quantidade de transferência e processamento de dados, e da latência de qualquer serviço com o qual a função interaja. Algumas causas comuns de tempo limite esgotado incluem:

- Os downloads do Amazon Simple Storage Service (Amazon S3) são maiores ou demoram mais que a média.
- Uma função faz uma solicitação a outro serviço, que leva mais tempo para responder.
- Os parâmetros fornecidos a uma função exigem mais complexidade computacional na função, o que faz com que a invocação leve mais tempo.

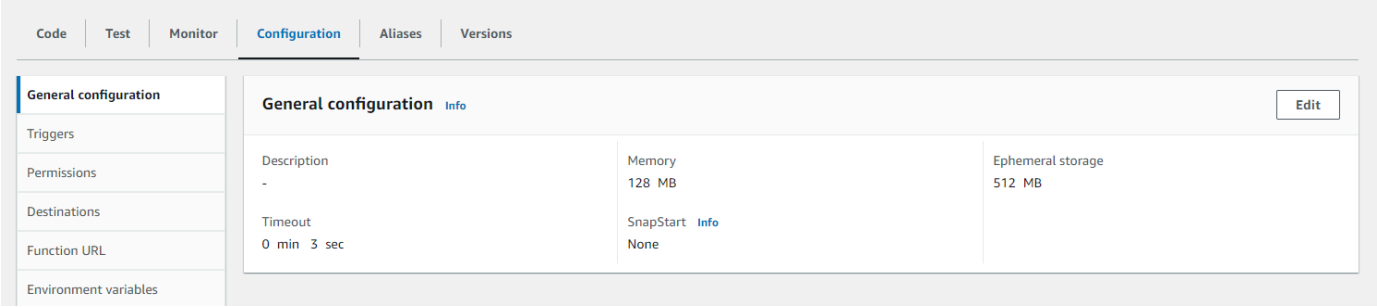
Ao testar sua aplicação, certifique-se de que os testes reflitam exatamente o tamanho e a quantidade de dados, bem como valores de parâmetros realistas. Os testes geralmente usam amostras pequenas por conveniência, mas você deve usar conjuntos de dados nos limites superiores do que é razoavelmente esperado para a workload.

Configurar tempo limite (console)

É possível configurar o tempo limite da função no console do Lambda.

Para modificar o tempo limite de uma função

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Configuração e depois selecione Configuração geral.



4. Em Configuração geral, escolha Editar.
5. Para Tempo limite, defina um valor entre 1 e 900 segundos (15 minutos).
6. Escolha Salvar.

Configurar tempo limite (AWS CLI)

Você pode usar o comando update-function-configuration para configurar o valor de tempo limite, em segundos. O exemplo de comando a seguir aumenta o tempo limite da função para 120 segundos (2 minutos).

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
  --timeout 120
```

Configurar tempo limite (AWS SAM)

Você pode usar o AWS Serverless Application Model para configurar o valor do tempo limite para a função. Atualize a propriedade Tempo limite no arquivo `template.yaml` e execute o comando 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...
```

Usar variáveis de ambiente no Lambda para configurar valores no código

É possível usar variáveis de ambiente para ajustar o comportamento da função sem atualizar o código. Uma variável de ambiente é um par de strings armazenadas na configuração específica da versão de uma função. O runtime do Lambda disponibiliza variáveis de ambiente para o código e define variáveis de ambiente adicionais que contêm informações sobre a função e a solicitação de invocação.

> **Note**
>
> Para aumentar a segurança, recomendamos que você use o AWS Secrets Manager em vez de variáveis de ambiente para armazenar as credenciais do banco de dados e outras informações confidenciais, como chaves de API ou tokens de autorização. Para obter mais informações, consulte Criar e gerenciar segredos com o AWS Secrets Manager.

As variáveis de ambiente não são avaliadas antes da invocação da função. Qualquer valor definido é considerado uma string literal e não expandido. Execute a avaliação da variável no seu código de função.

É possível configurar variáveis de ambiente no Lambda usando o console do Lambda, a AWS Command Line Interface (AWS CLI), o AWS Serverless Application Model (AWS SAM) ou um AWS SDK.

Console

Você define variáveis de ambiente na versão não publicada da sua função. Quando você publica uma versão, as variáveis de ambiente estão bloqueadas para essa versão junto com outras configurações específicas da versão.

Para criar uma variável de ambiente na sua função, defina uma chave e um valor. A função usa o nome da chave para recuperar o valor da variável de ambiente.

Para definir variáveis de ambiente no console do Lambda

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.

3. Escolha Configuration (Configuração) e, em seguida, Environment variables (Variáveis de ambiente).
4. Em Environment variables (Variáveis de ambiente), selecione Edit (Editar).
5. Escolha Add environment variable (Adicionar variável de ambiente).
6. Insira um par de chave e valor.

   Requisitos

   - As chaves começam com uma letra e têm pelo menos dois caracteres.
   - As chaves e os valores contêm somente letras, números e o caractere de sublinhado (_).
   - As chaves não são reservadas pelo Lambda.
   - O tamanho total de todas as variáveis de ambiente não excede 4 KB.
7. Escolha Salvar.

Gerar uma lista de variáveis de ambiente no editor de código do console

Você pode gerar uma lista de variáveis de ambiente no editor de código do Lambda. Esta é uma maneira rápida de referenciar as variáveis de ambiente enquanto você realiza a codificação.

1. Escolha a guia Código.
2. Escolha a guia Variáveis de ambiente.
3. Escolha Ferramentas, Mostrar variáveis de ambiente.

As variáveis de ambiente permanecem criptografadas quando estão listadas no editor de código do console. Se você habilitou os auxiliares de criptografia para a criptografia em trânsito, essas configurações permanecerão inalteradas. Para ter mais informações, consulte Proteger variáveis de ambiente no Lambda.

A lista de variáveis de ambiente é somente leitura e está disponível somente no console do Lambda. Este arquivo não está incluso quando você realiza download do arquivo .zip da função e não é possível adicionar variáveis de ambiente ao fazer upload deste arquivo.

AWS CLI

O exemplo a seguir define duas variáveis de ambiente em uma função denominada `my-function`.

```
aws lambda update-function-configuration \
  --function-name my-function \
  --environment "Variables={BUCKET=DOC-EXAMPLE-BUCKET,KEY=file.txt}"
```

Quando você aplica variáveis de ambiente com o comando `update-function-configuration`, todo o conteúdo da estrutura `Variables` é substituído. Para manter variáveis de ambiente existentes ao adicionar uma nova, inclua todos os valores existentes em sua solicitação.

Para obter a configuração atual, use o comando `get-function-configuration`.

```
aws lambda get-function-configuration \
  --function-name my-function
```

A seguinte saída deverá ser mostrada:

```
{
    "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",
    ...
}
```

É possível transmitir o ID de revisão da saída de `get-function-configuration` como um parâmetro para `update-function-configuration`. Isso garante que os valores não sejam alterados entre quando você lê a configuração e quando a atualiza.

Para configurar a chave de criptografia de uma função, defina a opção `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

Você pode usar o AWS Serverless Application Model para configurar as variáveis de ambiente da função. Atualize as propriedades Ambiente e Variáveis no arquivo `template.yaml` e execute o comando 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

Para gerenciar variáveis de ambiente usando um AWS SDK, use as operações de API a seguir.

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Para saber mais, consulte a documentação do AWS SDK para a linguagem de programação que você preferir.

Variáveis de ambiente com runtime definido

Os runtimes do Lambda definem diversas variáveis de ambiente durante a inicialização. A maioria das variáveis de ambiente fornece informações sobre a função ou o runtime. As chaves para essas variáveis de ambiente são reservadas e não podem ser definidas na configuração de sua função.

Variáveis de ambiente reservadas

- `_HANDLER`: o local do handler configurado na função.
- `_X_AMZN_TRACE_ID`— OCabeçalho do X-Ray. Essa variável de ambiente muda a cada invocação.
  - Essa variável de ambiente não está definida para runtimes somente para sistema operacional (a família `provided` de runtime). Você pode definir `_X_AMZN_TRACE_ID` para runtimes personalizados usando o cabeçalho da resposta `Lambda-Runtime-Trace-Id` do Próxima invocação.
  - Nas versões 17 e posteriores do runtime do Java, essa variável de ambiente não é usada. Em vez disso, o Lambda armazena informações de rastreamento na propriedade `com.amazonaws.xray.traceHeader` do sistema.
- `AWS_DEFAULT_REGION`: a Região da AWS padrão onde a função do Lambda é executada.
- `AWS_REGION`: a Região da AWS onde a função do Lambda é executada. Se estiver definido, esse valor substituirá `AWS_DEFAULT_REGION`.
  - Para obter mais informações sobre como usar as variáveis de ambiente da Região da AWS com AWS SDKs, consulte Região da AWS no Guia de referência de SDKs e ferramentas da AWS.
- `AWS_EXECUTION_ENV`: o identificador de runtime, prefixado por `AWS_Lambda_` (por exemplo, `AWS_Lambda_java8`). Essa variável de ambiente não está definida para runtimes somente para sistema operacional (a família `provided` de runtime).
- `AWS_LAMBDA_FUNCTION_NAME`: o nome da função.
- `AWS_LAMBDA_FUNCTION_MEMORY_SIZE`: a quantidade de memória disponível para a função em MB.
- `AWS_LAMBDA_FUNCTION_VERSION`: a versão da função que está sendo executada.
- `AWS_LAMBDA_INITIALIZATION_TYPE`: o tipo de inicialização da função, que é `on-demand`, `provisioned-concurrency` ou `snap-start`. Para obter informações, consulte Configurar a simultaneidade provisionada ou Aprimoramento da performance de inicialização com o Lambda SnapStart.

- `AWS_LAMBDA_LOG_GROUP_NAME`,`AWS_LAMBDA_LOG_STREAM_NAME`: o nome do grupo Amazon CloudWatch Logs e do fluxo do para a função. As variáveis de ambiente `AWS_LAMBDA_LOG_GROUP_NAME` e `AWS_LAMBDA_LOG_STREAM_NAME` não estão disponíveis nas funções do Lambda SnapStart.
- `AWS_ACCESS_KEY`, `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN`: as chaves de acesso obtidas da função de execução da função.
- `AWS_LAMBDA_RUNTIME_API`: (runtime personalizado) o host e a porta da API de runtime.
- `LAMBDA_TASK_ROOT`: o caminho para o código da função do Lambda.
- `LAMBDA_RUNTIME_DIR`: o caminho para bibliotecas de runtime.

As variáveis de ambiente adicionais a seguir não são reservadas e podem ser estendidas na configuração de sua função.

Variáveis de ambiente não reservadas

- `LANG`: a localidade do runtime (`en_US.UTF-8`).
- `PATH`: o caminho de execução (`/usr/local/bin:/usr/bin/:/bin:/opt/bin`).
- `LD_LIBRARY_PATH`: o caminho da biblioteca do 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) o caminho da biblioteca Node.js (`/opt/nodejs/node12/node_modules/:/opt/nodejs/node_modules:$LAMBDA_RUNTIME_DIR/node_modules`).
- `PYTHONPATH`: (Python 2.7, 3.6, 3.8) o caminho da biblioteca Python (`$LAMBDA_RUNTIME_DIR`).
- `GEM_PATH`: (Ruby) o caminho da biblioteca Ruby (`$LAMBDA_TASK_ROOT/vendor/bundle/ruby/2.5.0:/opt/ruby/gems/2.5.0`).
- `AWS_XRAY_CONTEXT_MISSING`: para o rastreamento do X-Ray, o Lambda define isso como `LOG_ERROR` para evitar a execução de erros de runtime no X-Ray SDK.
- `AWS_XRAY_DAEMON_ADDRESS`: para o rastreamento do X-Ray, o endereço IP e a porta do daemon do X-Ray.
- `AWS_LAMBDA_DOTNET_PREJIT`: para os runtimes do .NET 6 e .NET 7, defina essa variável para ativar ou desativar otimizações específicas de runtime do .NET. Os valores incluem `always`, `never` e `provisioned-concurrency`. Para ter mais informações, consulte Configurar a simultaneidade provisionada para uma função.

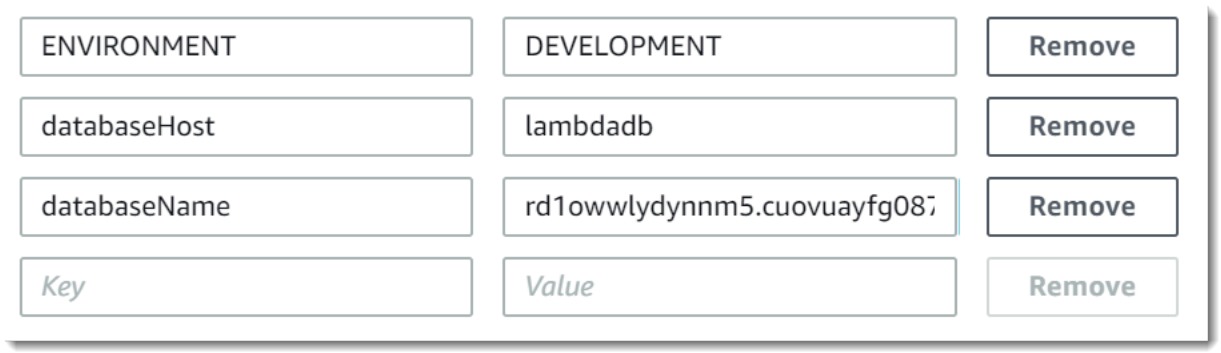
- TZ: o fuso horário do ambiente (UTC). O ambiente de execução usa o NTP para sincronizar o relógio do sistema.

Os valores de exemplo mostrados refletem os runtimes mais recentes. A presença de variáveis específicas ou seus valores pode variar nos runtimes anteriores.

Exemplo de cenário para variáveis de ambiente

Você pode usar variáveis de ambiente para personalizar o comportamento da função no ambiente de teste e no ambiente de produção. Por exemplo, você pode criar duas funções com o mesmo código, mas configurações diferentes. Uma função se conecta a um banco de dados de teste e a outra se conecta a um banco de dados de produção. Nessa situação, você usa variáveis de ambiente para transmitir o nome do host e outros detalhes de conexão do banco de dados para a função.

O exemplo a seguir mostra como definir o host do banco de dados e o nome do banco de dados como variáveis de ambiente.



Se quiser que o ambiente de teste gere mais informações de depuração do que o ambiente de produção, você poderá definir uma variável de ambiente para configurar o ambiente de teste para usar um registro em log mais detalhado ou um rastreamento mais detalhado.

Proteger variáveis de ambiente no Lambda

Para proteger suas variáveis de ambiente, você pode usar a criptografia do lado do servidor para proteger seus dados em repouso e a criptografia do lado do cliente para proteger seus dados em trânsito.

> **Note**
>
> Para aumentar a segurança do banco de dados, recomendamos que você use o AWS Secrets Manager em vez de variáveis de ambiente para armazenar as credenciais do banco de dados. Para ter mais informações, consulte Usar o AWS Lambda com o Amazon RDS.

Segurança em repouso

O Lambda sempre fornece criptografia do lado do servidor em repouso com AWS KMS key. Por padrão, o Lambda usa um Chave gerenciada pela AWS. Se esse comportamento padrão for adequado ao fluxo de trabalho, você não precisará configurar mais nada. O Lambda cria o Chave gerenciada pela AWS em sua conta e gerencia permissões para você. AWS não cobra para usar esta chave.

Se você preferir, pode fornecer uma AWS KMS chave gerenciada pelo cliente. Você pode fazer isso para ter controle sobre a alternância da chave KMS ou para atender aos requisitos de sua organização para gerenciar chaves KMS. Quando você fornece uma chave gerenciada pelo cliente, somente os usuários em sua conta com acesso à chave KMS podem visualizar ou gerenciar variáveis de ambiente na função.

As chaves gerenciadas pelo cliente incorrem em cobranças do AWS KMS padrão. Para obter mais informações, consulte Preços do AWS Key Management Service.

Segurança em trânsito

Para segurança adicional, você pode habilitar auxiliares para criptografia em trânsito, o que garante que suas variáveis de ambiente sejam criptografadas no lado do cliente para proteção em trânsito.

Para configurar a criptografia para suas variáveis de ambiente

1. Usar a AWS Key Management Service (AWS KMS) para criar quaisquer chaves gerenciadas pelo cliente para o Lambda usar para criptografia do lado do cliente e do servidor. Para obter mais informações, consulte Criar chaves no AWS Key Management Service Guia do desenvolvedor.
2. Usando o console do Lambda, navegue até a página Editar variáveis de ambiente.
    a. Abra a página Funções do console do Lambda.
    b. Escolha uma função.

c. Selecione Configuração e, depois, escolha Variáveis de ambiente na barra de navegação à esquerda.

d. Em Variáveis de ambiente, selecione Editar.

e. Expanda Encryption configuration (Configuração de criptografia).

3. (Opcional) Habilite os auxiliares de criptografia do console para usar a criptografia do lado do cliente com a finalidade de proteger os dados em trânsito.

   a. Em Criptografia em trânsito, escolha Habilitar auxiliares para criptografia em trânsito.

   b. Para cada variável de ambiente na qual deseja habilitar os auxiliares de criptografia do console, escolha Encrypt (Criptografar) ao lado da variável de ambiente.

   c. Em AWS KMS key para criptografar em trânsito, escolha uma chave gerenciada pelo cliente criada no início deste procedimento.

   d. Selecione Política da função de execução e copie a política. Essa política concede permissão à função de execução de sua função para descriptografar variáveis de ambiente.

      Salve essa política para usar na última etapa deste procedimento.

   e. Adicione à sua função o código que descriptografa as variáveis de ambiente. Para ver um exemplo, escolha Snippet de descriptografia de segredos.

4. (Opcional) Especifique a chave gerenciada pelo cliente para a criptografia em repouso.

   a. Escolha Uso de chave primária do cliente.

   b. Escolha uma chave gerenciada pelo cliente que você criou no início deste procedimento.

5. Escolha Salvar.

6. Configurar permissões

   Se você estiver usando uma chave gerenciada pelo cliente com criptografia do lado do servidor, conceda permissões a todos os usuários ou perfis que deseja visualizar ou gerenciar variáveis de ambiente na função. Para ter mais informações, consulte Gerenciar permissões para a chave KMS de criptografia do lado do servidor.

   Se você estiver habilitando a criptografia do lado do cliente para segurança em trânsito, sua função precisará de permissão para chamar o método operação da API `kms:Decrypt`. Adicione a política que você salvou anteriormente à Função de execução neste procedimento.

Gerenciar permissões para a chave KMS de criptografia do lado do servidor

Para usar a chave de criptografia padrão, não é necessária nenhuma permissão do AWS KMS para o usuário ou para a função de execução da função. Para usar uma chave gerenciada pelo cliente, é necessário permissão para usar a chave. O Lambda usa essas permissões para criar uma concessão na chave. Isso permite que o Lambda use-a para criptografia.

- `kms:ListAliases`: para visualizar as teclas no console do Lambda.
- `kms:CreateGrant`, `kms:Encrypt` – para configurar uma chave gerenciada pelo cliente em uma função.
- `kms:Decrypt` – para visualizar e gerenciar variáveis de ambiente criptografadas com uma chave gerenciada pelo cliente.

É possível obter essas permissões em sua Conta da AWS ou na política de permissões baseada em recursos de uma chave. `ListAliases` é fornecido pelas políticas gerenciadas para o Lambda. As políticas de chave concedem as permissões restantes aos usuários no grupo Usuários de chaves.

Os usuários sem permissões `Decrypt` ainda poderão gerenciar funções, mas não poderão visualizar variáveis de ambiente nem gerenciá-las no console do Lambda. Para impedir que um usuário visualize variáveis de ambiente, adicione uma declaração às permissões do usuário que negue o acesso à chave padrão, a uma chave gerenciada pelo cliente ou a todas as chaves.

Example Política do IAM: negar acesso pelo ARN da chave

```
{
    "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"
        }
    ]
}
```

Para obter detalhes sobre como gerenciar permissões de chave, consulte Key policies in AWS KMS no Guia do desenvolvedor do AWS Key Management Service.

Recuperar variáveis de ambiente do Lambda

Para recuperar variáveis de ambiente em seu código de função, use o método padrão para a sua linguagem de programação.

Node.js

```
let region = process.env.AWS_REGION
```

Python

```
import os
  region = os.environ['AWS_REGION']
```

> **Note**
>
> Em alguns casos, talvez seja necessário usar o seguinte 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
```

O Lambda armazena as variáveis de ambiente de forma segura, criptografando-as em repouso. É possível configurar o Lambda para usar uma chave de criptografia diferente, criptografar valores de variáveis de ambiente no lado do cliente ou definir variáveis de ambiente em um modelo do AWS CloudFormation com o AWS Secrets Manager.

Conceder acesso a funções do Lambda para recursos em uma Amazon VPC

Com a Amazon Virtual Private Cloud (Amazon VPC), você pode criar redes privadas em sua Conta da AWS para hospedar recursos como instâncias do Amazon Elastic Compute Cloud (Amazon EC2), do Amazon Relational Database Service (Amazon RDS) e do Amazon ElastiCache. Você pode dar à sua função do Lambda acesso a recursos hospedados em uma Amazon VPC ao anexar sua função à VPC por meio das sub-redes privadas que contêm os recursos. Siga as instruções nas seções a seguir para anexar uma função do Lambda a uma Amazon VPC usando o console Lambda, a AWS Command Line Interface (AWS CLI) ou AWS SAM.

> **Note**
>
> Cada função do Lambda é executada dentro de uma VPC de propriedade e gerenciada pelo serviço do Lambda. Essas VPCs recebem manutenção automática do Lambda e não são visíveis para os clientes. Configurar sua função para acessar outros recursos da AWS em uma Amazon VPC não tem efeito na VPC gerenciada pelo Lambda na qual sua função é executada.

Seções

Permissões obrigatórias do IAM

Para anexar uma função do Lambda a uma Amazon VPC em sua Conta da AWS, o Lambda precisa de permissões para criar e gerenciar as interfaces de rede que usa para dar à sua função acesso aos recursos na VPC.

As interfaces de rede que o Lambda cria são conhecidas como Hyperplane Elastic Network Interfaces, ou Hyperplane ENIs. Para saber mais sobre essas interfaces de rede, consulte the section called “Noções básicas de interfaces de rede elástica (ENIs) de hiperplano”.

Você pode dar à sua função as permissões necessárias anexando a política gerenciada da AWS AWSLambdaVPCAccessExecutionRole ao perfil de execução da função. Quando você cria uma nova função no console do Lambda e a anexa a uma VPC, o Lambda adiciona automaticamente essa política de permissões para você.

Se você preferir criar sua própria política de permissões do IAM, adicione todas as permissões a seguir:

- ec2:CreateNetworkInterface
- ec2:DescribeNetworkInterfaces: essa ação só funciona se for permitida em todos os recursos (`"Resource": "*"`).
- ec2:DescribeSubnets
- ec2:DeleteNetworkInterface: se você não especificar um ID de recurso para DeleteNetworkInterface no perfil de execução, sua função talvez não possa acessar a VPC. Especifique um ID de recurso exclusivo ou inclua todos os IDs de recursos, por exemplo, `"Resource": "arn:aws:ec2:us-west-2:123456789012:*/*"`.
- ec2:AssignPrivateIpAddresses
- ec2:UnassignPrivateIpAddresses

Observe que o perfil da sua função só precisa dessas permissões para criar as interfaces de rede, não para invocar sua função. Você ainda poderá invocar a função com sucesso quando ela estiver anexada a uma Amazon VPC, mesmo que você remova essas permissões do perfil de execução da função.

Para anexar sua função a uma VPC, o Lambda também precisa verificar os recursos de rede usando seu perfil de usuário do IAM. Seu perfil de usuário deve ter as seguintes permissões do IAM:

- ec2:DescribeSecurityGroups
- ec2:DescribeSubnets
- ec2:DescribeVpcs

> **Note**
>
> As permissões do Amazon EC2 que você concede ao perfil de execução da função são usadas pelo serviço Lambda para anexar sua função a uma VPC. No entanto, essas permissões também são concedidas implicitamente ao código da sua função. Isso significa que seu código de função pode fazer essas chamadas de API do Amazon EC2. Para obter conselhos sobre como seguir as práticas recomendadas de segurança, consulte the section called “Melhores práticas de segurança”.

Como anexar funções do Lambda a uma Amazon VPC em sua Conta da AWS

Anexe sua função a uma Amazon VPC em sua Conta da AWS usando o console do Lambda, a AWS CLI ou o AWS SAM. Se você estiver usando a AWS CLI ou o AWS SAM, ou anexando uma função existente a uma VPC usando o console do Lambda, o perfil de execução da função deve ter as permissões necessárias listadas na seção anterior.

As funções do Lambda não podem se conectar diretamente a uma VPC com a locação de instâncias dedicadas. Para se conectar a recursos em uma VPC dedicada, emparelhe-a com uma segunda VPC com locação padrão.

Lambda console

Para anexar uma função a uma Amazon VPC ao criá-la

1. Abra a página Funções do console do Lambda e escolha Criar função.
2. Em Basic information (Informações básicas), para Function name (Nome da função), insira um nome para a função.
3. Defina as configurações da função desta forma:
   a. Expanda Advanced settings (Configurações avançadas).
   b. Selecione Habilitar VPC e, em seguida, escolha a VPC à qual deseja anexar a função.
   c. (Opcional) Para permitir tráfego IPv6 de saída, selecione Permitir tráfego IPv6 para sub-redes de pilha dupla.

d. Escolha as sub-redes e os grupos de segurança para os quais criar a interface de rede. Se você selecionou Permitir tráfego IPv6 para sub-redes de pilha dupla, todas as sub-redes selecionadas deverão ter um bloco CIDR IPv4 e um bloco CIDR IPv6.

> **Note**
>
> Para acessar recursos privados, conecte a função às sub-redes privadas. Se sua função precisar de acesso à Internet, consulte the section called “Acesso à Internet para funções da VPC”. Conectar uma função a uma sub-rede pública não dá a ela acesso à Internet nem a um endereço IP público.

4. Escolha a opção Criar função.

Para anexar uma função existente a uma Amazon VPC

1. Abra a página Funções do console do Lambda e selecione a função.
2. Escolha a guia Configuração e depois VPC.
3. Selecione a opção Editar.
4. Em VPC, selecione a Amazon VPC à qual deseja anexar sua função.
5. (Opcional) Para permitir tráfego IPv6 de saída, selecione Permitir tráfego IPv6 para sub-redes de pilha dupla.
6. Escolha as sub-redes e os grupos de segurança para os quais criar a interface de rede. Se você selecionou Permitir tráfego IPv6 para sub-redes de pilha dupla, todas as sub-redes selecionadas deverão ter um bloco CIDR IPv4 e um bloco CIDR IPv6.

> **Note**
>
> Para acessar recursos privados, conecte a função às sub-redes privadas. Se sua função precisar de acesso à Internet, consulte the section called “Acesso à Internet para funções da VPC”. Conectar uma função a uma sub-rede pública não dá a ela acesso à Internet nem a um endereço IP público.

7. Escolha Salvar.

AWS CLI

Para anexar uma função a uma Amazon VPC ao criá-la

- Para criar uma função do Lambda e anexá-la a uma VPC, execute este comando da 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
```

Especifique suas próprias sub-redes e grupos de segurança e defina `Ipv6AllowedForDualStack` como `true` ou `false` de acordo com seu caso de uso.

Para anexar uma função existente a uma Amazon VPC

- Para anexar uma função existente a uma VPC, execute este comando da 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
```

Para desanexar sua função de uma VPC

- Para desanexar a função de uma VPC, execute este comando da CLI `update-function-configuration` com uma lista vazia de sub-redes e grupos de segurança da VPC.

```
aws lambda update-function-configuration --function-name my-function \
--vpc-config SubnetIds=[],SecurityGroupIds=[]
```

AWS SAM

Para anexar sua função a uma VPC

- Para anexar uma função do Lambda a uma Amazon VPC, adicione a propriedade `VpcConfig` à sua definição de função, conforme mostrado no modelo de exemplo a seguir. Para obter mais informações sobre essa propriedade, consulte AWS::Lambda::Function VpcConfig no Guia de usuário do AWS CloudFormation (a propriedade AWS SAM `VpcConfig` é passada diretamente para a propriedade `VpcConfig` de um recurso `AWS::Lambda::Function` do AWS CloudFormation).

```
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
```

Para obter mais informações sobre como configurar sua VPC no AWS SAM, consulte AWS::EC2::VPC no Guia do usuário do AWS CloudFormation.

Acesso à Internet quando anexado a uma VPC

Por padrão, as funções do Lambda têm acesso à Internet pública. Quando você anexar sua função a uma VPC, ela só poderá acessar os recursos disponíveis nessa VPC. Para conceder acesso à Internet para a função, também é necessário configurar a VPC para ter acesso à Internet. Para saber mais, consulte the section called “Acesso à Internet para funções da VPC”.

Práticas recomendadas para usar o Lambda com Amazon VPCs

Para garantir que sua configuração de VPC da Lambda atenda às diretrizes de práticas recomendadas, siga os conselhos nas seções a seguir.

Melhores práticas de segurança

Para anexar sua função do Lambda a uma VPC, você precisa conceder ao perfil de execução da função várias permissões do Amazon EC2. Essas permissões são necessárias para criar as interfaces de rede que sua função usa para acessar os recursos na VPC. No entanto, essas permissões também são concedidas implicitamente ao código da sua função. Isso significa que seu código de função tem permissão para fazer essas chamadas de API do Amazon EC2.

Para seguir o princípio do acesso com privilégios mínimos, adicione uma política de negação, como o exemplo a seguir, ao perfil de execução da sua função. Essa política impede que sua função faça chamadas para as APIs do Amazon EC2 que o serviço do Lambda usa para anexar sua função a uma 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"
                    ]
                }
            }
        }
    ]
}
```

A AWS fornece grupos de segurança e listas de controle de acesso (ACL) de rede para aumentar a segurança em sua VPC. Os grupos de segurança controlam o tráfego de entrada e de saída para seus recursos e as Network ACLs controlam o tráfego de entrada e de saída de suas sub-redes. Os grupos de segurança fornecem controle de acesso suficiente para a maioria das sub-redes. Você pode usar as ACLs de rede se desejar uma camada adicional de segurança para a VPC. Para obter diretrizes gerais sobre as práticas recomendadas de segurança ao usar Amazon VPCs, consulte Melhores práticas de segurança para a sua VPC no Guia do usuário do Amazon Virtual Private Cloud.

Práticas recomendadas de desempenho

Quando você anexa sua função a uma VPC, o Lambda verifica se há um recurso de rede disponível (ENI de hiperplano) que ela possa usar para se conectar. Os ENIs de hiperplano estão associados a uma combinação específica de grupos de segurança e sub-redes da VPC. Se você já anexou uma função a uma VPC, especificar as mesmas sub-redes e grupos de segurança ao anexar outra função significa que o Lambda pode compartilhar os recursos de rede e evitar a necessidade de criar um

novo ENI de hiperplano. Para obter mais informações sobre ENIs de hiperplano e seu ciclo de vida, consulte the section called “Noções básicas de interfaces de rede elástica (ENIs) de hiperplano”.

Noções básicas de interfaces de rede elástica (ENIs) de hiperplano

Uma ENI de hiperplano é um recurso gerenciado que atua como uma interface de rede entre sua função do Lambda e os recursos aos quais você deseja que sua função se conecte. O serviço do Lambda cria e gerencia essas ENIs automaticamente quando você anexa sua função a uma VPC.

As ENIs de hiperplano não são diretamente visíveis para você e você não precisa configurar ou gerenciá-las. No entanto, saber como elas funcionam pode ajudar você a entender o comportamento da sua função ao anexá-la a uma VPC.

Na primeira vez que você associa uma função a uma VPC usando uma combinação específica de sub-rede e grupo de segurança, o Lambda cria uma ENI de hiperplano. Outras funções da sua conta que usam a mesma combinação de sub-rede e grupo de segurança também podem usar essa ENI. Sempre que possível, o Lambda reutiliza ENIs existentes para otimizar a utilização de recursos e minimizar a criação de novas ENIs. Porém, cada ENI de hiperplano comporta até 65.000 conexões/portas. Se o número de conexões exceder esse limite, o Lambda escalará o número de ENIs automaticamente com base no tráfego de rede e nos requisitos de simultaneidade.

Para novas funções, enquanto o Lambda está criando uma ENI de hiperplano, sua função permanece no estado Pendente e você não pode invocá-la. Sua função fará a transição para o estado ativo somente quando a ENI de hiperplano estiver pronta, o que pode levar vários minutos. Para funções existentes, não é possível executar operações adicionais direcionadas à função, como criar versões ou atualizar o código da função, mas é possível continuar invocando versões anteriores da função.

> **Note**
>
> Se uma função do Lambda permanecer ociosa por 30 dias, o Lambda recuperará as ENIs de hiperplano não utilizadas e definirá o estado da função como ocioso. A próxima tentativa de invocação falha e a função entra novamente no estado pendente até que o Lambda conclua a criação ou alocação de uma ENI de hiperplano. Para obter mais informações sobre estados de funções do Lambda, consulte the section called “Estados de função”.

Para obter mais informações sobre o ciclo de vida de ENIs de hiperplano, consulte the section called “ENIs de hiperplano do Lambda”.

Usar chaves de condição do IAM para configurações de VPC

É possível usar chaves de condição específicas do Lambda para configurações de VPC a fim de fornecer controles de permissão adicionais para as funções do Lambda. Por exemplo, você pode exigir que todas as funções em sua organização estejam conectadas a uma VPC. Você também pode especificar as sub-redes e os grupos de segurança que os usuários da função podem e não podem usar.

O Lambda oferece suporte às seguintes chaves de condição em políticas do IAM:

- lambda:VpcIds: permitir ou negar uma ou mais VPCs.
- lambda:SubnetIds: permitir ou negar uma ou mais sub-redes.
- lambda:SecurityGroupIds: permitir ou negar um ou mais grupos de segurança.

As operações de API CreateFunction e UpdateFunctionConfiguration do Lambda oferecem suporte a essas chaves de condição. Para obter mais informações sobre o uso de chaves de condição em políticas do IAM, consulteElementos de política JSON do: condiçãonoIAM User Guide.

> **Tip**
>
> Se a função já incluir uma configuração de VPC de uma solicitação de API anterior, você poderá enviar uma solicitação `UpdateFunctionConfiguration` sem a configuração da VPC.

Políticas de exemplo com chaves de condição para configurações de VPC

Os exemplos a seguir demonstram como usar chaves de condição para configurações de VPC. Depois de criar uma instrução de política com as restrições desejadas, acrescente a instrução de política para o usuário ou a função de destino.

Os usuários devem implantar somente funções conectadas à VPC

Para garantir que todos os usuários implantem somente funções conectadas à VPC, você pode negar operações de criação e de atualização de funções que não incluam um ID de VPC válido.

Observe que o ID da VPC não é um parâmetro de entrada para a solicitação `CreateFunction` ou `UpdateFunctionConfiguration`. O Lambda recupera o valor do ID da VPC com base nos parâmetros da sub-rede e do grupo de segurança.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceVPCFunction",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "Null": {
           "lambda:VpcIds": "true"
        }
      }
    }
  ]
}
```

Negar acesso de usuários a VPCs, sub-redes ou grupos de segurança específicos

Para negar acesso de usuários a VPCs específicas, use `StringEquals` para verificar o valor da condição `lambda:VpcIds`. O exemplo a seguir nega aos usuários acesso à `vpc-1` e à `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"]
        }
      }
    }
```

Para negar acesso de usuários a sub-redes específicas, use `StringEquals` para verificar o valor da condição `lambda:SubnetIds`. O exemplo a seguir nega aos usuários acesso à `subnet-1` e à `subnet-2`.

```
{
      "Sid": "EnforceOutOfSubnet",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
            "lambda:SubnetIds": ["subnet-1", "subnet-2"]
        }
      }
    }
```

Para negar acesso de usuários a grupos de segurança específicos, use `StringEquals` para verificar o valor da condição `lambda:SecurityGroupIds`. O exemplo a seguir nega aos usuários acesso à `sg-1` e à `sg-2`.

```
{
      "Sid": "EnforceOutOfSecurityGroups",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
            "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
        }
      }
    }
  ]
}
```

Permitir que os usuários criem e atualizem funções com configurações de VPC específicas

Para permitir que os usuários acessem VPCs específicas, use `StringEquals` para verificar o valor da condição `lambda:VpcIds`. O exemplo a seguir dá aos usuários permissão para acessar `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"]
        }
      }
    }
```

Para permitir que os usuários acessem sub-redes específicas, use `StringEquals` para verificar o valor da condição `lambda:SubnetIds`. O exemplo a seguir dá aos usuários permissão para acessar `subnet-1` e `subnet-2`.

```
{
      "Sid": "EnforceStayInSpecificSubnets",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
            "lambda:SubnetIds": ["subnet-1", "subnet-2"]
```

```
        }
      }
    }
```

Para permitir que os usuários acessem grupos de segurança específicos, use `StringEquals` para verificar o valor da condição `lambda:SecurityGroupIds`. O exemplo a seguir dá aos usuários permissão para acessar `sg-1` e `sg-2`.

```
{
      "Sid": "EnforceStayInSpecificSecurityGroup",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
            "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
        }
      }
    }
  ]
}
```

Tutoriais de VPC

Nos tutoriais a seguir, você conecta uma função do Lambda a recursos na VPC.

- Tutorial: usar uma função do Lambda para acessar o Amazon RDS em uma Amazon VPC
- Tutorial: configurar uma função do Lambda para acessar o Amazon ElastiCache em uma Amazon VPC

Habilitar o acesso à Internet para funções do Lambda conectadas à VPC

Por padrão, as funções do Lambda são executadas em uma VPC gerenciada pelo Lambda que tem acesso à Internet. Para acessar recursos em uma VPC em sua conta, você pode adicionar uma configuração de VPC a uma função. A menos que a VPC tenha acesso à Internet, isso restringirá a função aos recursos dentro dessa VPC. Esta página explica como fornecer acesso à Internet às funções do Lambda conectadas à VPC.

Ainda não tenho uma VPC

Criar a VPC

O fluxo de trabalho Criar VPC cria todos os recursos de VPC necessários para que uma função do Lambda acesse a Internet pública usando uma sub-rede privada, inclusive sub-redes, gateway NAT, gateway da Internet e entradas na tabela de rotas.

Como criar a VPC

1. Abra o console da Amazon VPC em https://console.aws.amazon.com/vpc/.
2. No painel, escolha Criar VPC.
3. Em Resources to create (Recursos a serem criados), escolha VPC and more (VPC e mais).
4. Configurar a VPC
    a. Em Name tag auto-generation (Geração automática de tags de nome), insira um nome para a VPC.
    b. Em Bloco CIDR IPv4, é possível manter a sugestão padrão ou inserir o bloco CIDR exigido por sua aplicação ou rede.
    c. Se a sua aplicação se comunica usando endereços IPv6, escolha Bloco CIDR IPv6, Bloco CIDR IPv6 fornecido pela Amazon.
5. Configurar as sub-redes
    a. Em Número de zonas de disponibilidade, escolha 2. Recomendamos pelo menos duas AZs para alta disponibilidade.
    b. Em Number of public subnets (Número de sub-redes públicas), escolha 2.
    c. Em Number of private subnets (Número de sub-redes privadas), escolha 2.

d. É possível manter o bloco CIDR padrão para a sub-rede pública ou, alternativamente, expandir Personalizar blocos CIDR da sub-rede e inserir um bloco CIDR. Para obter mais informações, consulte Blocos CIDR de sub-rede.

6. Em Gateways NAT, escolha 1 por zona de disponibilidade para melhorar a resiliência.
7. Em Gateway da Internet somente de saída, escolha Sim se tiver optado por incluir um bloco CIDR IPv6.
8. Em Endpoints da VPC, mantenha o valor padrão (Gateway do S3). Não há custo para essa opção. Para obter mais informações, consulte Tipos de endpoints da VPC para o Amazon S3.
9. Em Opções de DNS, mantenha as configurações padrão.
10. Escolha Criar VPC.

Configurar a função do Lambda

Como configurar uma VPC ao criar uma função

1. Abra a página Funções do console do Lambda.
2. Escolha a opção Criar função.
3. Em Basic information (Informações básicas), para Function name (Nome da função), insira um nome para a função.
4. Expanda Advanced settings (Configurações avançadas).
5. Selecione Habilitar VPC e, em seguida, escolha uma VPC.
6. (Opcional) Para permitir tráfego IPv6 de saída, selecione Permitir tráfego IPv6 para sub-redes de pilha dupla.
7. Em Sub-redes, selecione todas as sub-redes privadas. As sub-redes privadas podem acessar a Internet por meio do gateway NAT. Conectar uma função a uma sub-rede pública não permite que ela tenha acesso à Internet.

> **Note**
>
> Se você selecionou Permitir tráfego IPv6 para sub-redes de pilha dupla, todas as sub-redes selecionadas deverão ter um bloco CIDR IPv4 e um bloco CIDR IPv6.

8. Em Grupos de segurança, selecione um grupo de segurança que permita tráfego de saída.
9. Escolha a opção Criar função.

O Lambda criará automaticamente um perfil de execução com a política AWSLambdaVPCAccessExecutionRole gerenciada pela AWS. As permissões nessa política são necessárias apenas para criar interfaces de rede elástica para a configuração da VPC, mas não para invocar a função. Para aplicar permissões com privilégios mínimos, você pode remover a política AWSLambdaVPCAccessExecutionRole do seu perfil de execução após criar a função e a configuração da VPC. Para ter mais informações, consulte Permissões obrigatórias do IAM.

Como configurar uma VPC para uma função existente

Para adicionar uma configuração de VPC a uma função existente, o perfil de execução da função precisa ter permissão para criar e gerenciar interfaces de rede elástica. A política AWSLambdaVPCAccessExecutionRole gerenciada pela AWS inclui as permissões necessárias. Para aplicar permissões com privilégios mínimos, você pode remover a política AWSLambdaVPCAccessExecutionRole do seu perfil de execução após criar a configuração da VPC.

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha a guia Configuração e depois VPC.
4. Em VPC, selecione Edit (Editar).
5. Selecionar a VPC.
6. (Opcional) Para permitir tráfego IPv6 de saída, selecione Permitir tráfego IPv6 para sub-redes de pilha dupla.
7. Em Sub-redes, selecione todas as sub-redes privadas. As sub-redes privadas podem acessar a Internet por meio do gateway NAT. Conectar uma função a uma sub-rede pública não permite que ela tenha acesso à Internet.

   > **Note**
   >
   > Se você selecionou Permitir tráfego IPv6 para sub-redes de pilha dupla, todas as sub-redes selecionadas deverão ter um bloco CIDR IPv4 e um bloco CIDR IPv6.

8. Em Grupos de segurança, selecione um grupo de segurança que permita tráfego de saída.
9. Escolha Salvar.

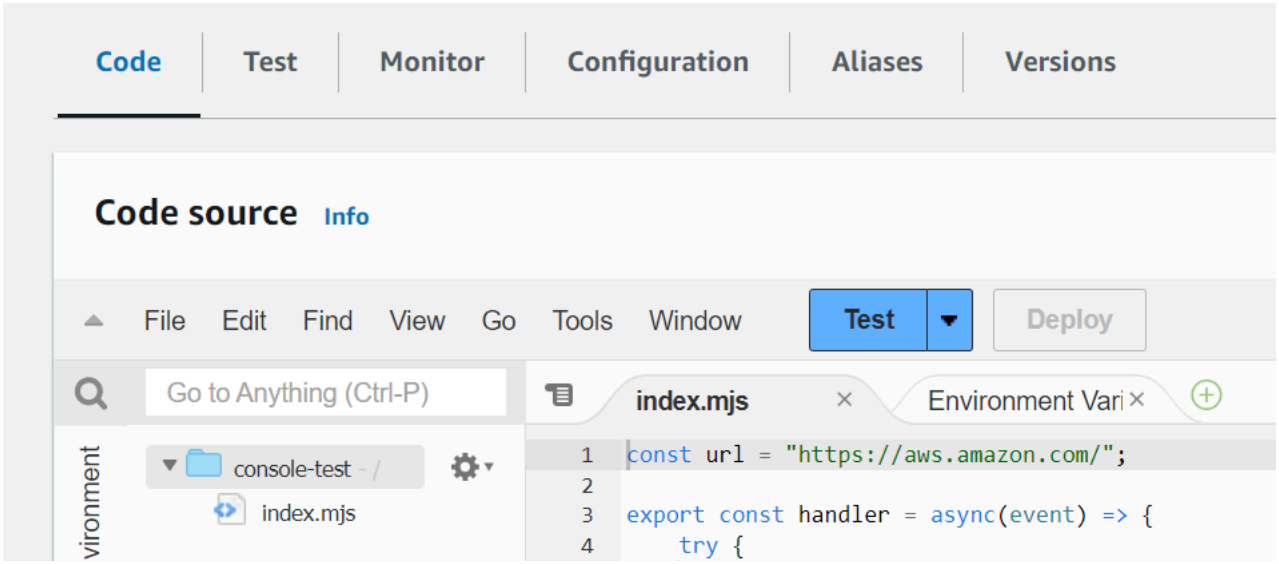
Testar a função

Use o seguinte exemplo de código para confirmar se a sua função conectada à VPC pode acessar a Internet pública. Se for bem-sucedido, o código retornará um código de status `200`. Se não for bem-sucedido, o tempo-limite da função esgotará.

Node.js

Este exemplo usa `fetch`, que está disponível no runtime `nodejs18.x` e em versões posteriores.

1. No painel Código-fonte no console do Lambda, cole o código a seguir no arquivo index.mjs. A função faz uma solicitação HTTP GET para um endpoint público e retorna o código de resposta HTTP para testar se a função tem acesso à Internet pública.

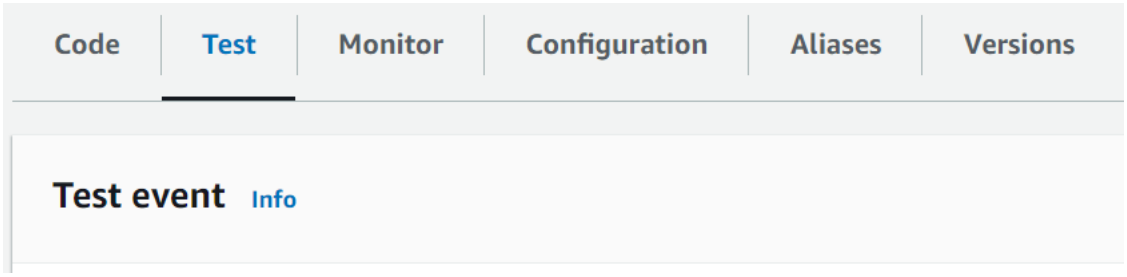


Example – Solicitação HTTP com async/await

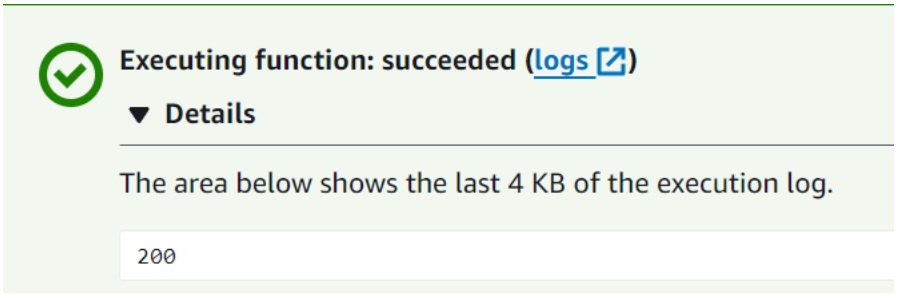
```
const url = "https://aws.amazon.com/";

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. Escolha Implantar.
3. Selecione a guia Testar.



4. Escolha Testar.
5. A função retorna um código de status `200`. Isso significa que a função tem acesso de saída à Internet.

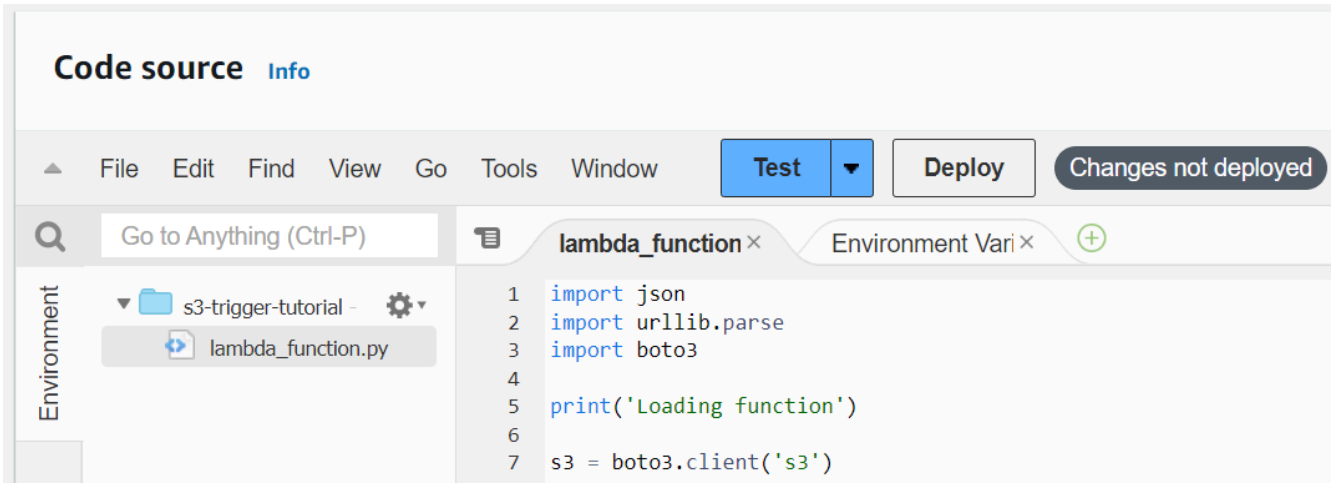


Se a função não conseguir acessar a Internet pública, você receberá uma mensagem de erro como esta:

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

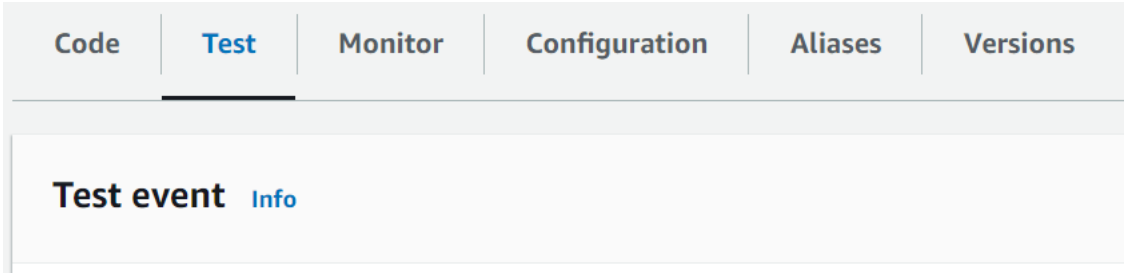
Python

1. No painel Código-fonte no console do Lambda, cole o código a seguir no arquivo lambda_function.py. A função faz uma solicitação HTTP GET para um endpoint público e retorna o código de resposta HTTP para testar se a função tem acesso à Internet pública.

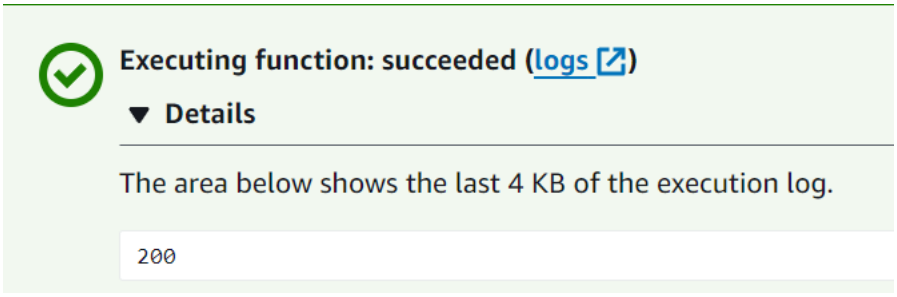



```
import urllib.request

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. Escolha Implantar.
3. Selecione a guia Testar.



4. Escolha Testar.
5. A função retorna um código de status `200`. Isso significa que a função tem acesso de saída à Internet.



Se a função não conseguir acessar a Internet pública, você receberá uma mensagem de erro como esta:

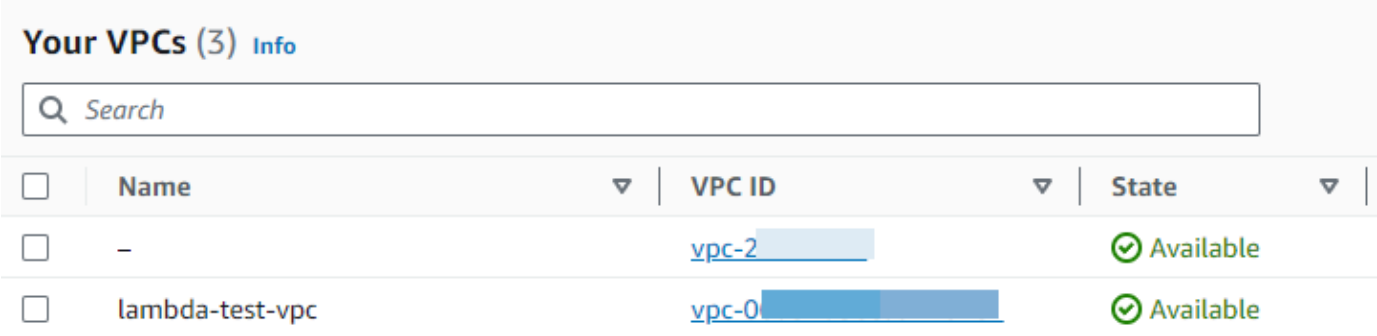
```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

Já tenho uma VPC

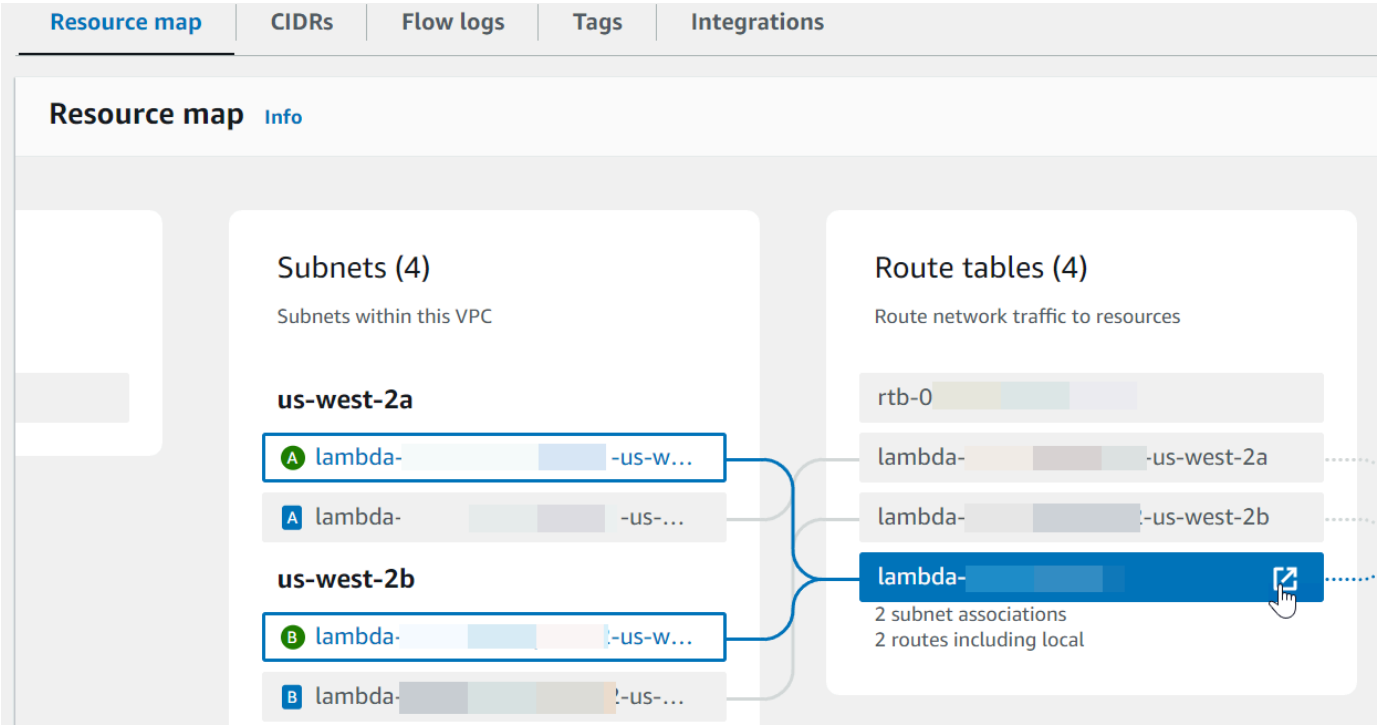
Se você já tiver uma VPC, mas precisar configurar o acesso público à Internet para uma função do Lambda, siga estas etapas. Este procedimento pressupõe que a VPC tenha pelo menos duas sub-redes. Se você não tiver duas sub-redes, consulte Criar uma sub-rede no Guia do usuário da Amazon VPC.

Verifique a configuração da tabela de rotas

1. Abra o console da Amazon VPC em https://console.aws.amazon.com/vpc/.
2. Escolha o ID da VPC.



3. Role para baixo até a seção Mapa de recursos. Anote os mapeamentos da tabela de rotas. Abra cada tabela de rotas mapeada para uma sub-rede.

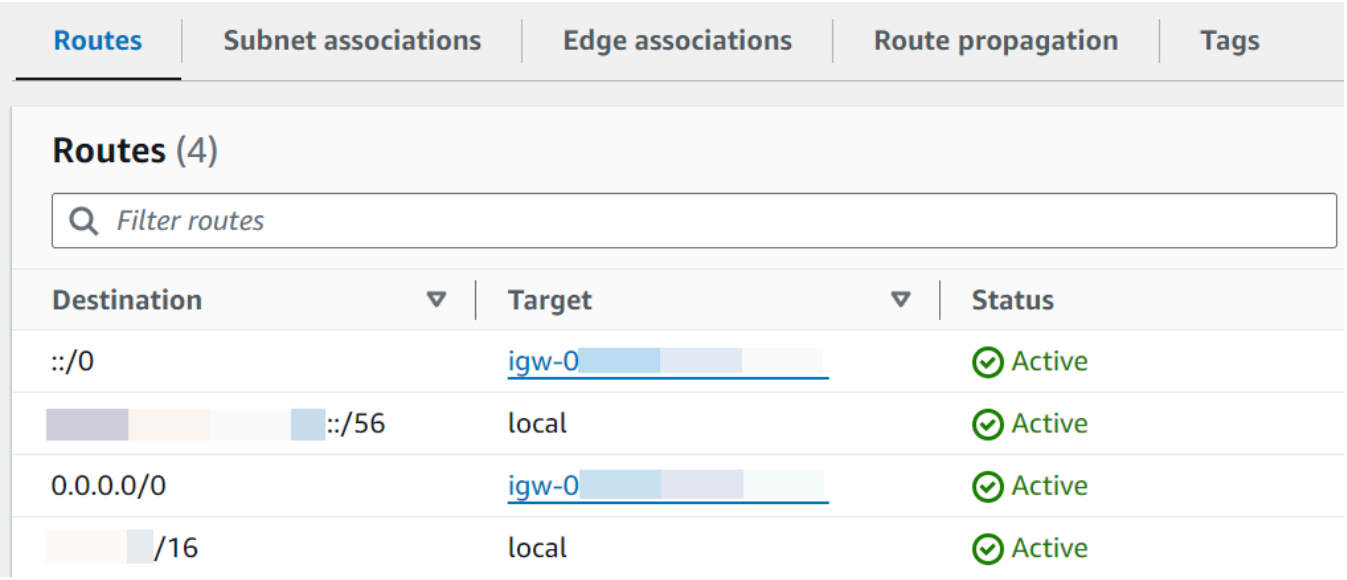


4. Role para baixo até a guia Rotas. Examine as rotas para determinar se uma das opções a seguir é verdadeira. É necessário atender a cada um desses requisitos com uma tabela de rotas separada.

   - O tráfego vinculado à Internet (`0.0.0.0/0` para IPv4, `::/0` para IPv6) é roteado para um gateway da Internet (`igw-xxxxxxxxxx`). Isso significa que a sub-rede associada à tabela de rotas é uma sub-rede pública.

     > **Note**
     >
     > Se sua sub-rede não tiver um bloco CIDR IPv6, você verá somente a rota IPv4 (`0.0.0.0/0`).

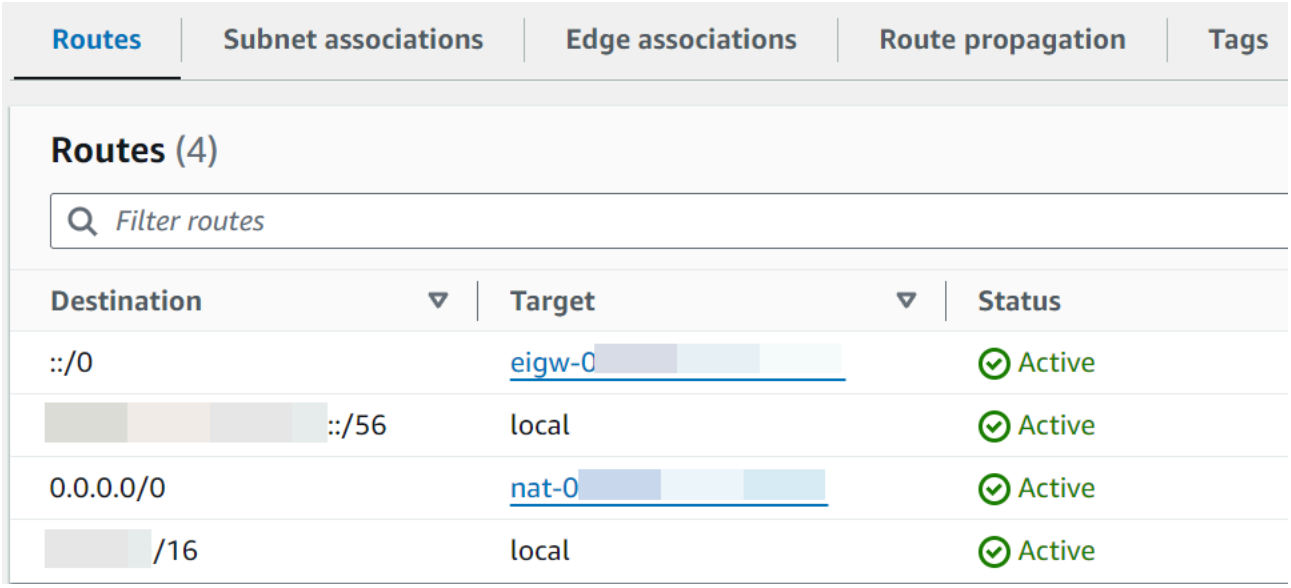
Example tabela de rotas de sub-rede pública



- O tráfego vinculado à Internet para IPv4 (`0.0.0.0/0`) é roteado para um gateway NAT (`nat-xxxxxxxxxx`) associado a uma sub-rede pública. Isso significa que a sub-rede é uma sub-rede privada capaz de acessar a Internet por meio do gateway NAT.

> **Note**
>
> Se a sub-rede tiver um bloco CIDR IPv6, a tabela de rotas também deverá rotear o tráfego IPv6 direcionado à Internet (`::/0`) para um gateway da Internet somente de saída (`eigw-xxxxxxxxxx`). Se sua sub-rede não tiver um bloco CIDR IPv6, você verá somente a rota IPv4 (`0.0.0.0/0`).

Example tabela de rotas de sub-rede privada



5. Repita a etapa anterior até ter revisado cada tabela de rotas associada a uma sub-rede em sua VPC e confirmar que tem uma tabela de rotas com um gateway da Internet e uma tabela de rotas com um gateway NAT.

   Se você não tiver duas tabelas de rotas, uma com uma rota para um gateway da Internet e outra com uma rota para um gateway NAT, siga estas etapas a fim de criar os recursos e as entradas da tabela de rotas ausentes.

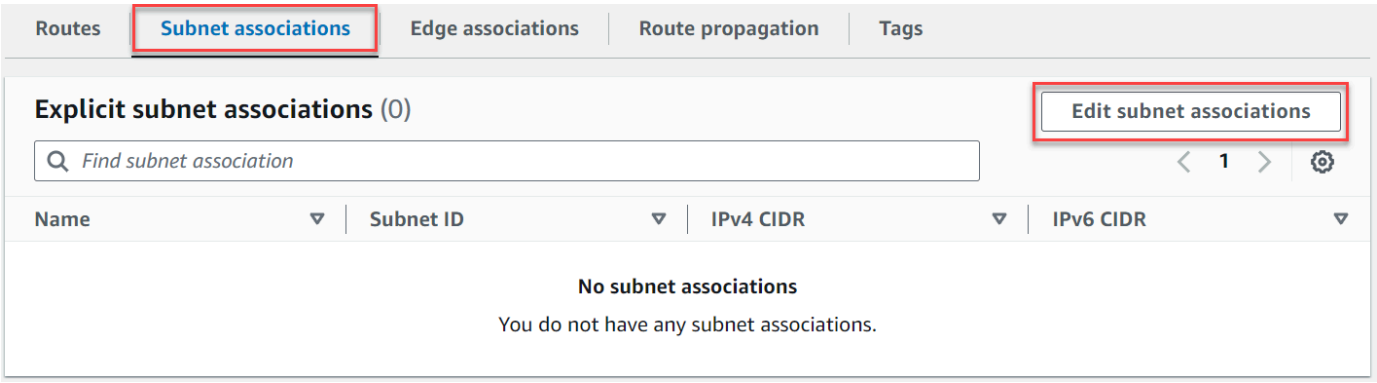
Crie uma tabela de rotas

Siga estas etapas para criar uma tabela de rotas e associá-la a uma sub-rede.

Para criar uma tabela de rotas personalizada usando o console da Amazon VPC

1. Abra o console da Amazon VPC em https://console.aws.amazon.com/vpc/.
2. No painel de navegação, escolha Route tables.
3. Escolha Create Route Table (Criar tabela de rotas).
4. (Opcional) Em Name (Nome), insira um nome para a tabela de rotas.
5. Em VPC, escolha sua VPC.
6. (Opcional) Para adicionar uma etiqueta, escolha Add new tag (Adicionar nova etiqueta) e insira a chave e o valor da etiqueta.

7. Escolha Create Route Table (Criar tabela de rotas).
8. Na guia Subnet Associations (Associações da sub-rede) selecione Edit subnet associations (Editar associações da sub-rede).



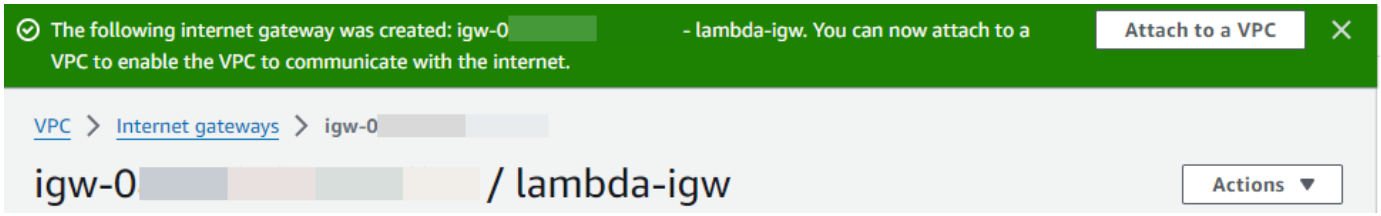
9. Marque a caixa de seleção para a sub-rede associada à tabela de rotas.
10. SelecioneSave associations (Salvar associações).

Criar um gateway da internet

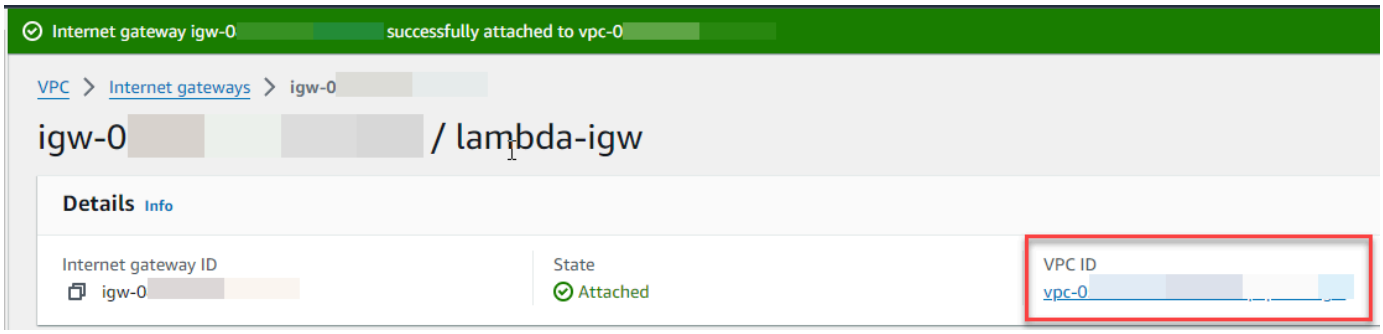
Siga estas etapas para criar um gateway da Internet, anexá-lo à sua VPC e adicioná-lo à tabela de rotas da sua sub-rede pública.

Para criar um gateway da Internet

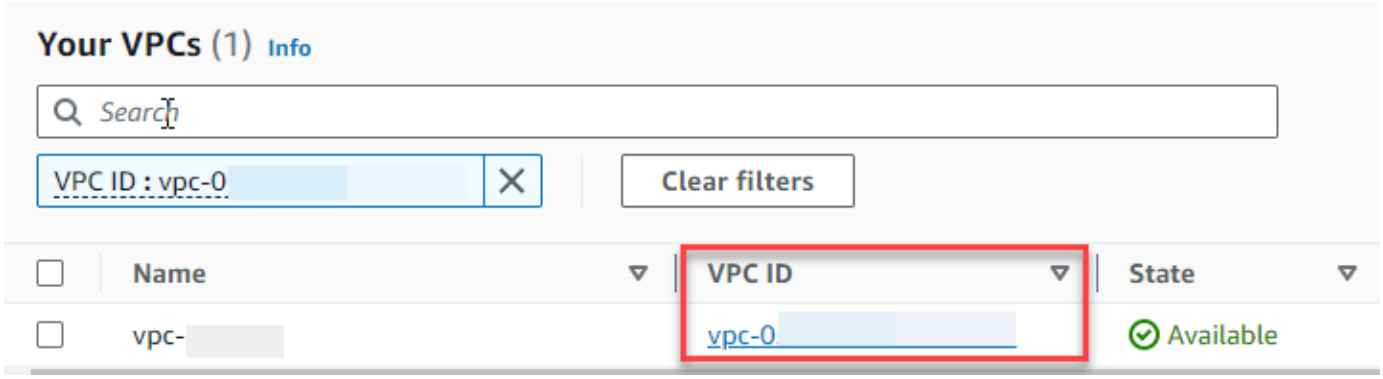
1. Abra o console da Amazon VPC em https://console.aws.amazon.com/vpc/.
2. No painel de navegação, escolha Internet gateways (Gateways da Internet).
3. Escolha Criar gateway da Internet.
4. (Opcional) Insira um nome para o gateway da Internet.
5. (Opcional) Para adicionar uma tag, escolha Add new tag (Adicionar nova tag) e insira a chave e o valor da tag.
6. Escolha Criar gateway da Internet.
7. Escolha Anexar a uma VPC no banner na parte superior da tela, selecione uma VPC disponível e escolha Anexar gateway da Internet.



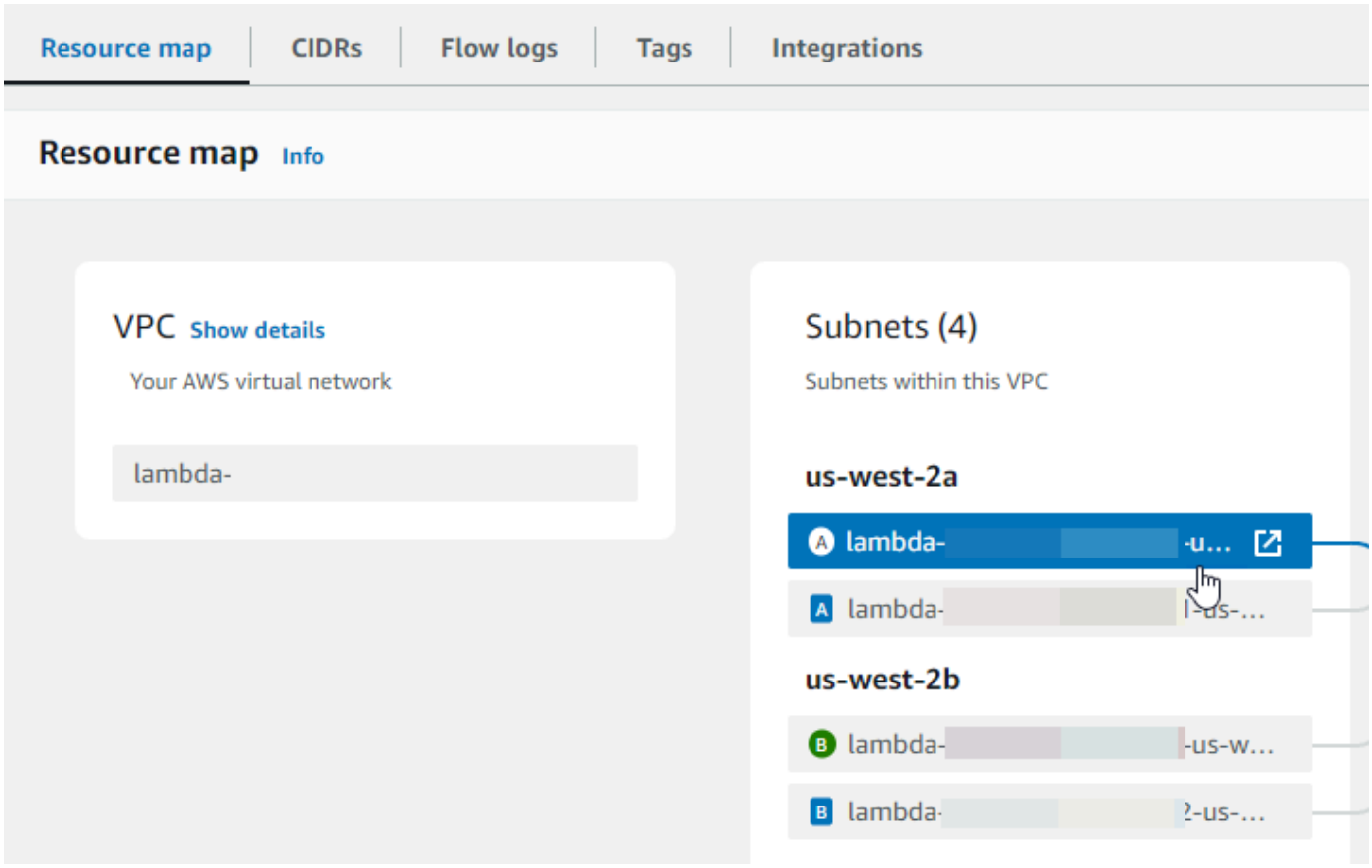
8. Escolha o ID da VPC.



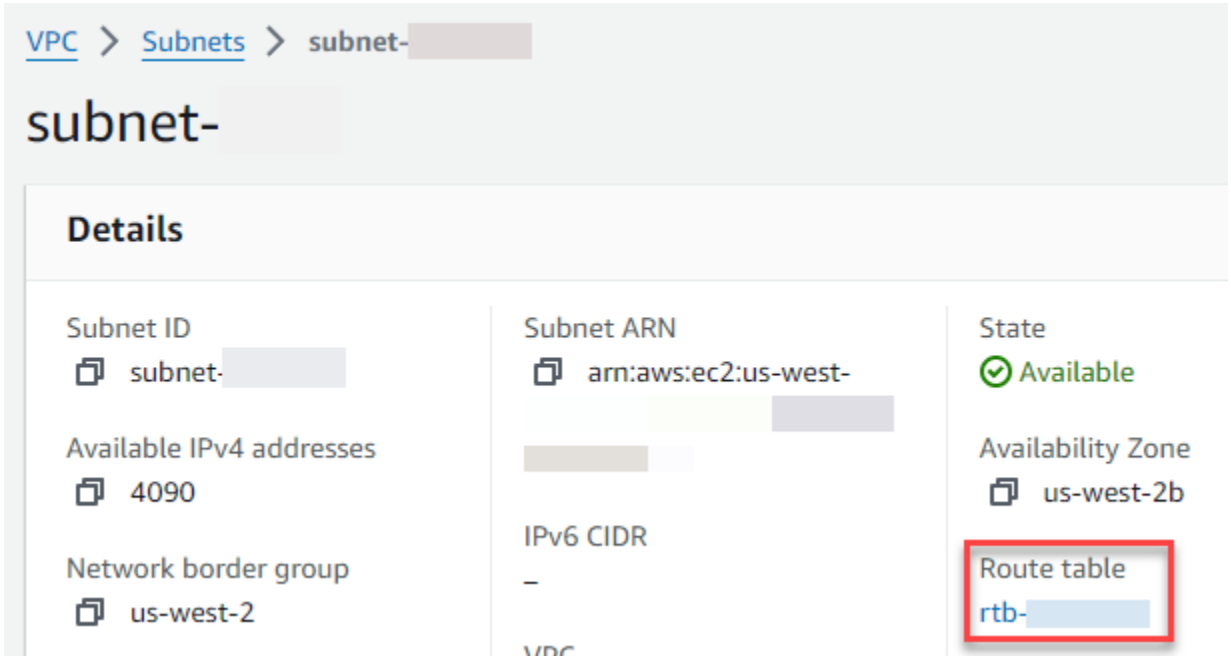
9. Escolha o ID da VPC novamente para abrir a página de detalhes da VPC.



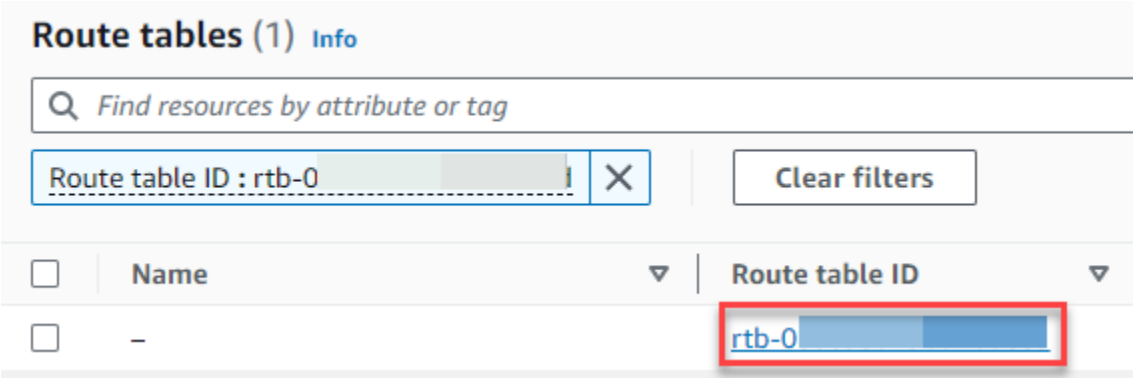
10. Role para baixo até a seção Mapa de recursos e escolha uma sub-rede. Os detalhes sobre a sub-rede são exibidos em uma nova guia.



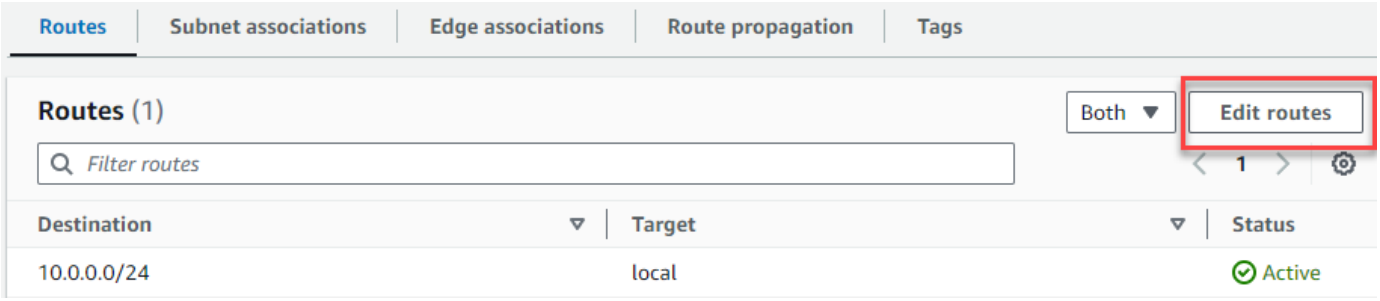
11. Escolha o link em Tabela de rotas.



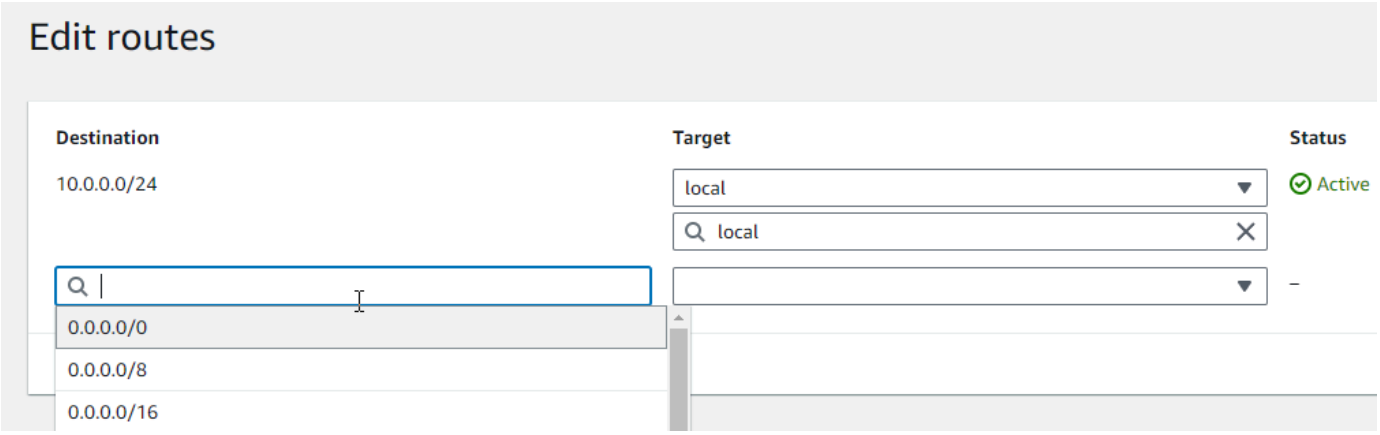
12. Escolha o ID da tabela de rotas para abrir a respectiva página de detalhes da tabela de rotas.



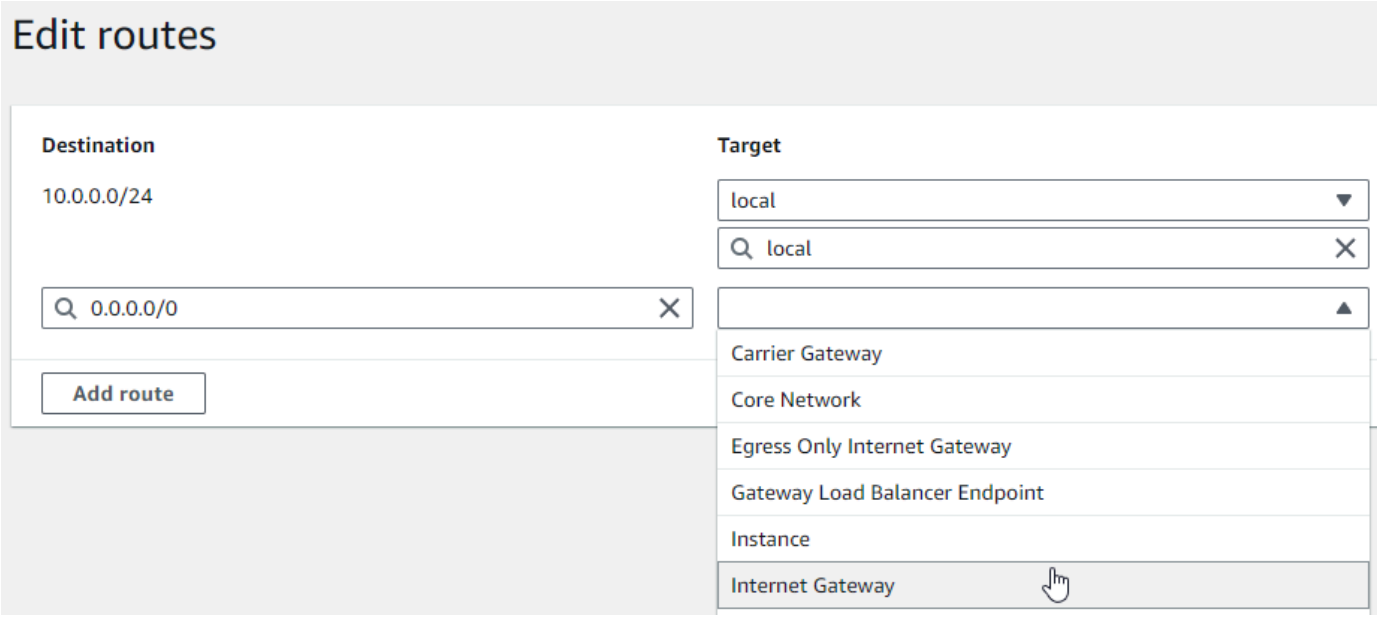
13. Em Rotas, escolha Editar rotas.



14. Escolha Adicionar rota e, em seguida, insira `0.0.0.0/0` na caixa Destino.



15. Em Destino, selecione Gateway da Internet e, em seguida, escolha o gateway da Internet que você criou anteriormente. Se a sub-rede tiver um bloco CIDR IPv6, você também deverá adicionar uma rota para `::/0` ao mesmo gateway da Internet.



16. Escolha Salvar alterações.

Criar um gateway NAT

Siga estas etapas para criar um gateway NAT, associá-lo a uma sub-rede pública e adicioná-lo à tabela de rotas da sua sub-rede privada.

Para criar um gateway NAT e associá-lo a uma sub-rede pública

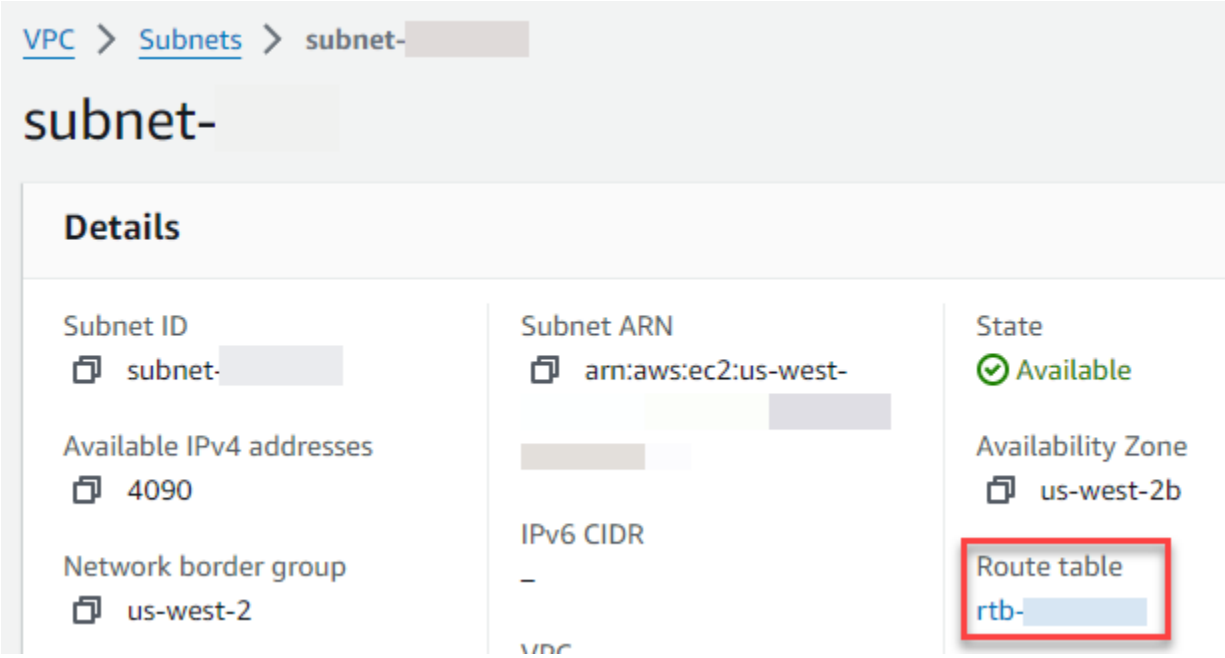
1. No painel de navegação, escolha Gateways NAT.
2. Escolha Criar um gateway NAT.
3. (Opcional) Insira um nome para o gateway NAT.
4. Em Sub-rede, selecione uma sub-rede pública na sua VPC. (Uma sub-rede pública é uma sub-rede que tem uma rota direta para um gateway da Internet em sua tabela de rotas.)

> **Note**
>
> Os gateways NAT estão associados a uma sub-rede pública, mas a entrada da tabela de rotas está na sub-rede privada.

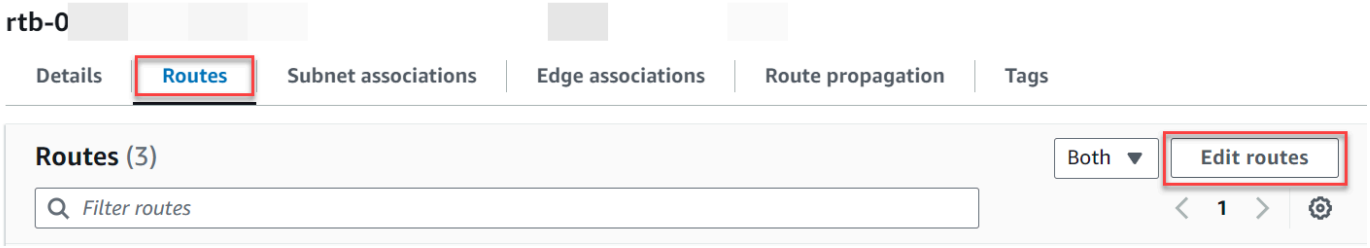
5. Em ID de alocação de IP elástico, selecione um endereço IP elástico ou escolha Alocar IP elástico.
6. Escolha Criar um gateway NAT.

Para adicionar uma rota ao gateway NAT na tabela de rotas da sub-rede privada

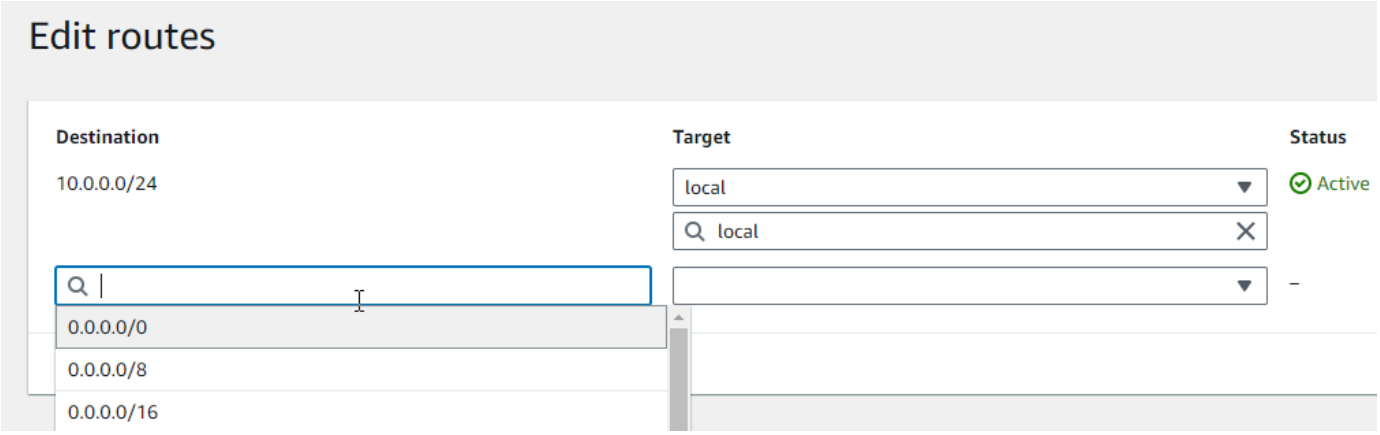
1. No painel de navegação, escolha Sub-redes.
2. Selecione uma sub-rede privada em sua VPC. (Uma sub-rede privada é uma sub-rede que não tem uma rota direta para um gateway da Internet em sua tabela de rotas.)
3. Escolha o link em Tabela de rotas.



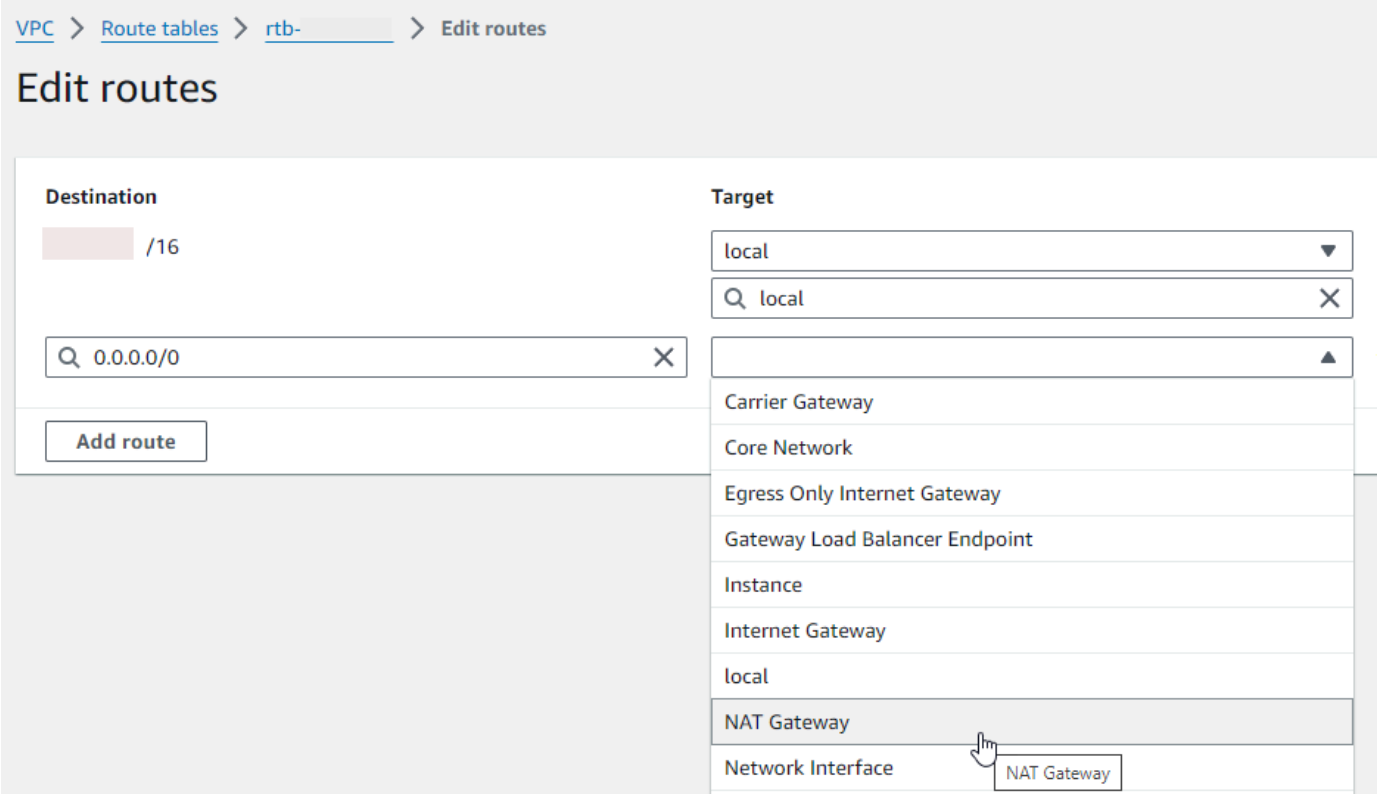
4. Role para baixo e escolha a guia Rotas e Editar rotas



5. Escolha Adicionar rota e, em seguida, insira `0.0.0.0/0` na caixa Destino.



6. Em Destino, selecione Gateway NAT e, em seguida, escolha o gateway NAT que você criou anteriormente.



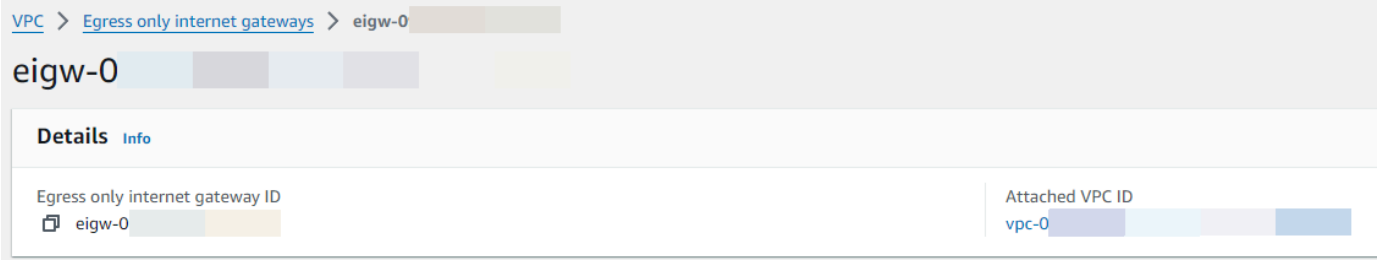
7. Escolha Salvar alterações.

Criar um gateway da Internet somente de saída (somente IPv6)

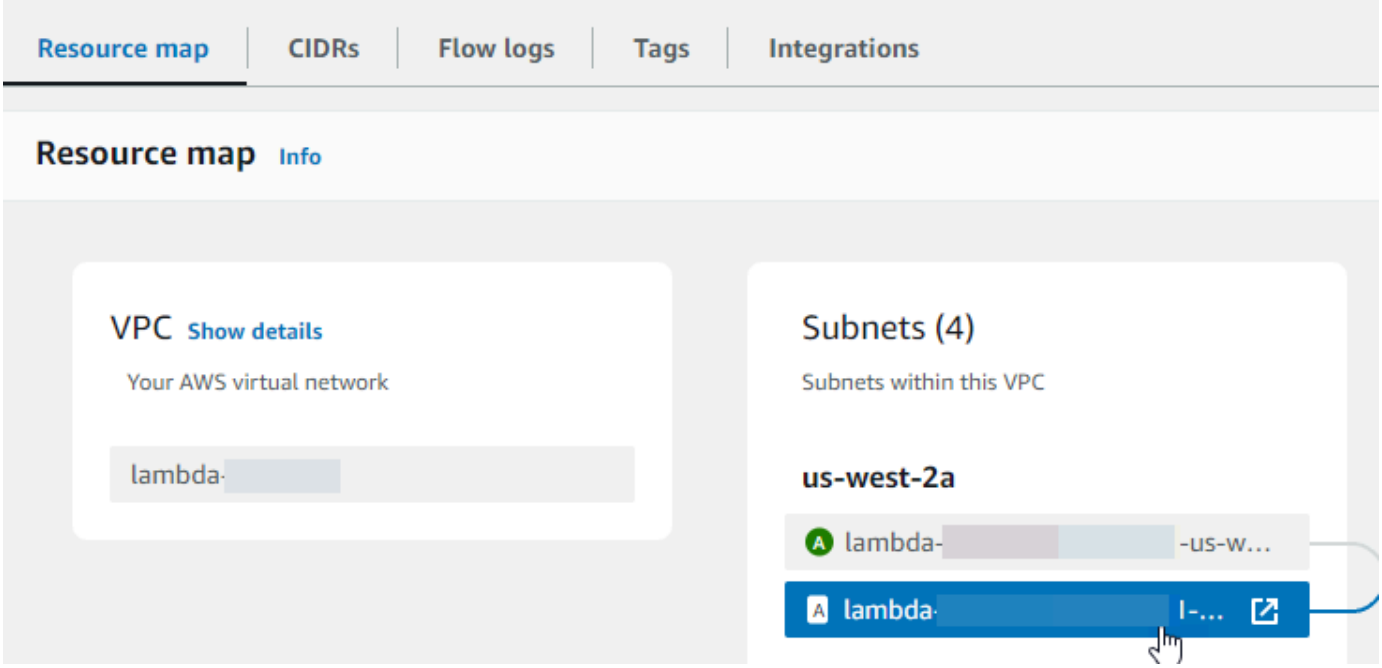
Siga estas etapas para criar um gateway da Internet somente de saída e adicioná-lo à tabela de rotas da sua sub-rede privada.

Como criar um gateway da Internet somente de saída para a VPC

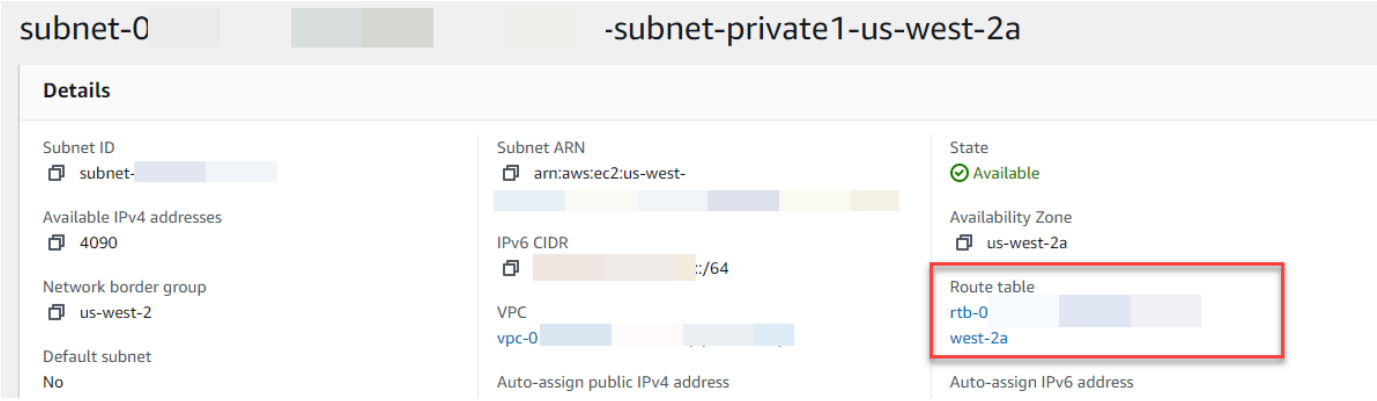
1. No painel de navegação, escolha Gateways da Internet somente de saída.
2. Escolha Criar um Gateway da Internet somente de saída.
3. (Opcional) Insira um nome.
4. Selecione a VPC para a qual será criado um gateway de Internet somente de saída.
5. Escolha Criar um Gateway da Internet somente de saída.
6. Escolha o link em ID da VPC anexada.



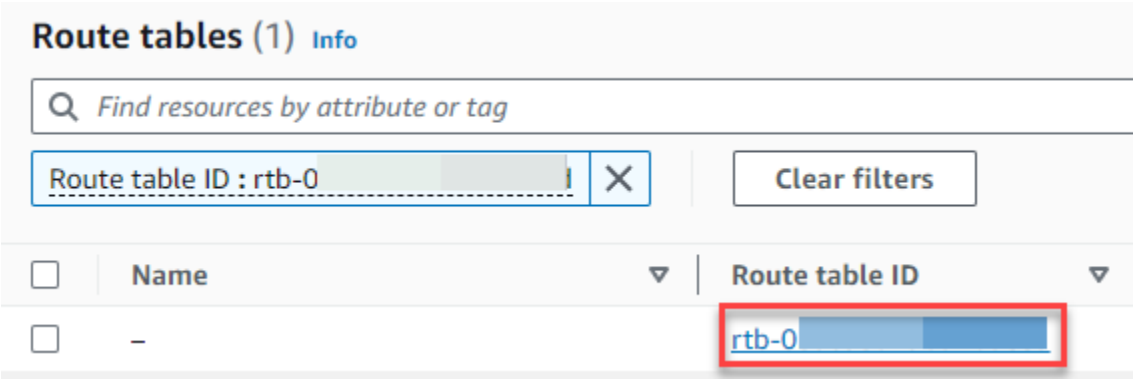
7. Escolha o link em ID da VPC para abrir a página de detalhes da VPC.
8. Role para baixo até a seção Mapa de recursos e escolha uma sub-rede privada. Os detalhes sobre a sub-rede são exibidos em uma nova guia.



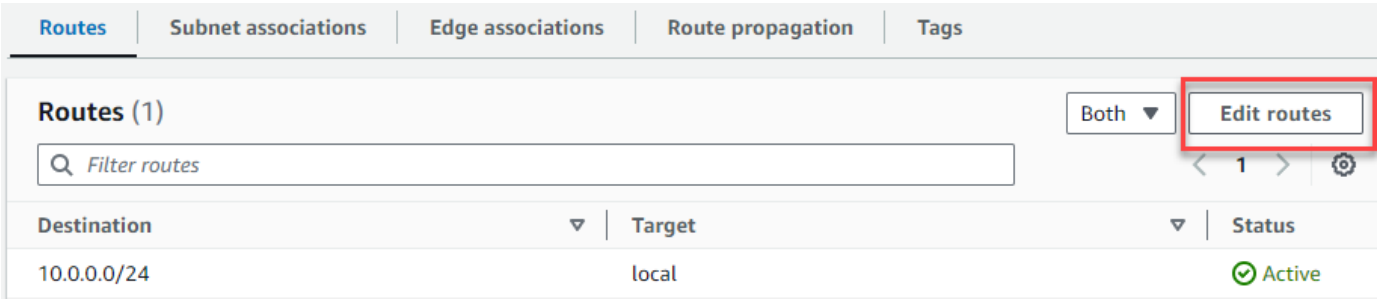
9. Escolha o link em Tabela de rotas.



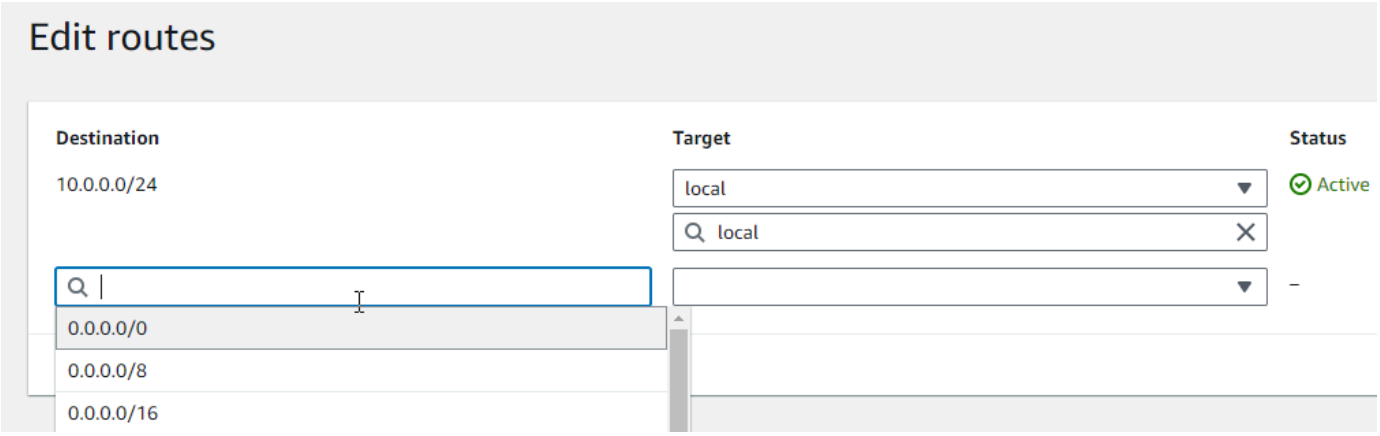
10. Escolha o ID da tabela de rotas para abrir a respectiva página de detalhes da tabela de rotas.



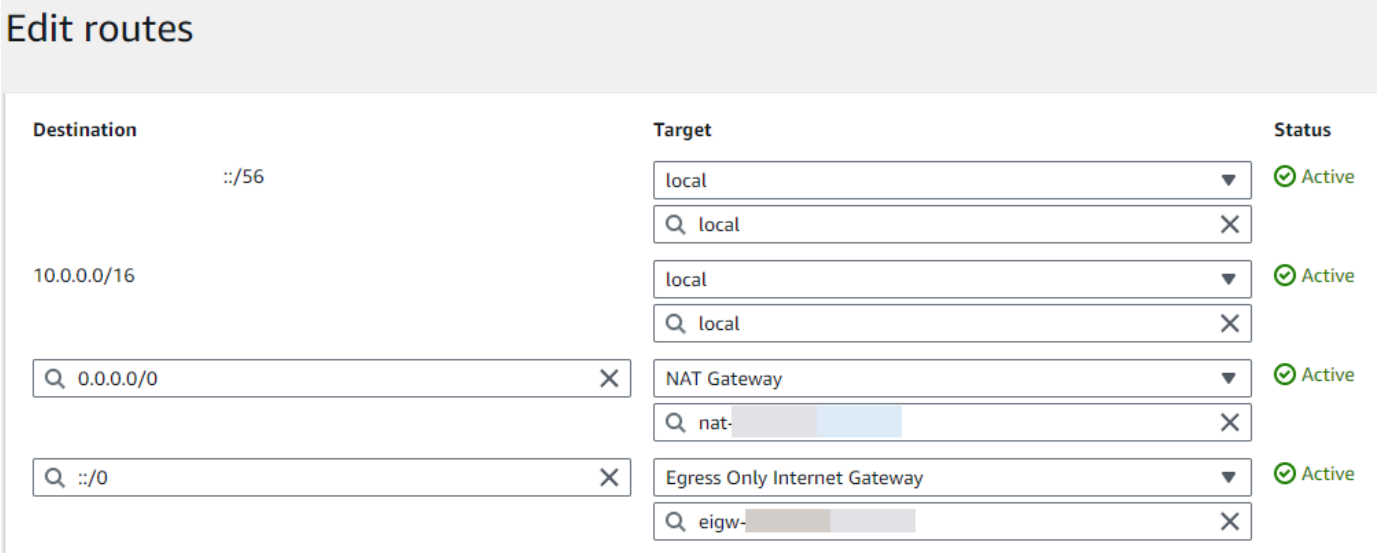
11. Em Rotas, escolha Editar rotas.



12. Escolha Adicionar rota e, em seguida, insira `::/0` na caixa Destino.



13. Em Destino, selecione Gateway da Internet somente de saída e, em seguida, escolha o gateway que você criou anteriormente.



14. Escolha Salvar alterações.

Configurar a função do Lambda

Como configurar uma VPC ao criar uma função

1. Abra a página Funções do console do Lambda.
2. Escolha a opção Criar função.
3. Em Basic information (Informações básicas), para Function name (Nome da função), insira um nome para a função.
4. Expanda Advanced settings (Configurações avançadas).
5. Selecione Habilitar VPC e, em seguida, escolha uma VPC.
6. (Opcional) Para permitir tráfego IPv6 de saída, selecione Permitir tráfego IPv6 para sub-redes de pilha dupla.
7. Em Sub-redes, selecione todas as sub-redes privadas. As sub-redes privadas podem acessar a Internet por meio do gateway NAT. Conectar uma função a uma sub-rede pública não permite que ela tenha acesso à Internet.

> **Note**
>
> Se você selecionou Permitir tráfego IPv6 para sub-redes de pilha dupla, todas as sub-redes selecionadas deverão ter um bloco CIDR IPv4 e um bloco CIDR IPv6.

8. Em Grupos de segurança, selecione um grupo de segurança que permita tráfego de saída.
9. Escolha a opção Criar função.

O Lambda criará automaticamente um perfil de execução com a política AWSLambdaVPCAccessExecutionRole gerenciada pela AWS. As permissões nessa política são necessárias apenas para criar interfaces de rede elástica para a configuração da VPC, mas não para invocar a função. Para aplicar permissões com privilégios mínimos, você pode remover a política AWSLambdaVPCAccessExecutionRole do seu perfil de execução após criar a função e a configuração da VPC. Para ter mais informações, consulte Permissões obrigatórias do IAM.

Como configurar uma VPC para uma função existente

Para adicionar uma configuração de VPC a uma função existente, o perfil de execução da função precisa ter permissão para criar e gerenciar interfaces de rede elástica. A política AWSLambdaVPCAccessExecutionRole gerenciada pela AWS inclui as permissões necessárias. Para aplicar permissões com privilégios mínimos, você pode remover a política AWSLambdaVPCAccessExecutionRole do seu perfil de execução após criar a configuração da VPC.

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha a guia Configuração e depois VPC.
4. Em VPC, selecione Edit (Editar).
5. Selecionar a VPC.
6. (Opcional) Para permitir tráfego IPv6 de saída, selecione Permitir tráfego IPv6 para sub-redes de pilha dupla.
7. Em Sub-redes, selecione todas as sub-redes privadas. As sub-redes privadas podem acessar a Internet por meio do gateway NAT. Conectar uma função a uma sub-rede pública não permite que ela tenha acesso à Internet.

> **Note**
>
> Se você selecionou Permitir tráfego IPv6 para sub-redes de pilha dupla, todas as sub-redes selecionadas deverão ter um bloco CIDR IPv4 e um bloco CIDR IPv6.

8. Em Grupos de segurança, selecione um grupo de segurança que permita tráfego de saída.
9. Escolha Salvar.

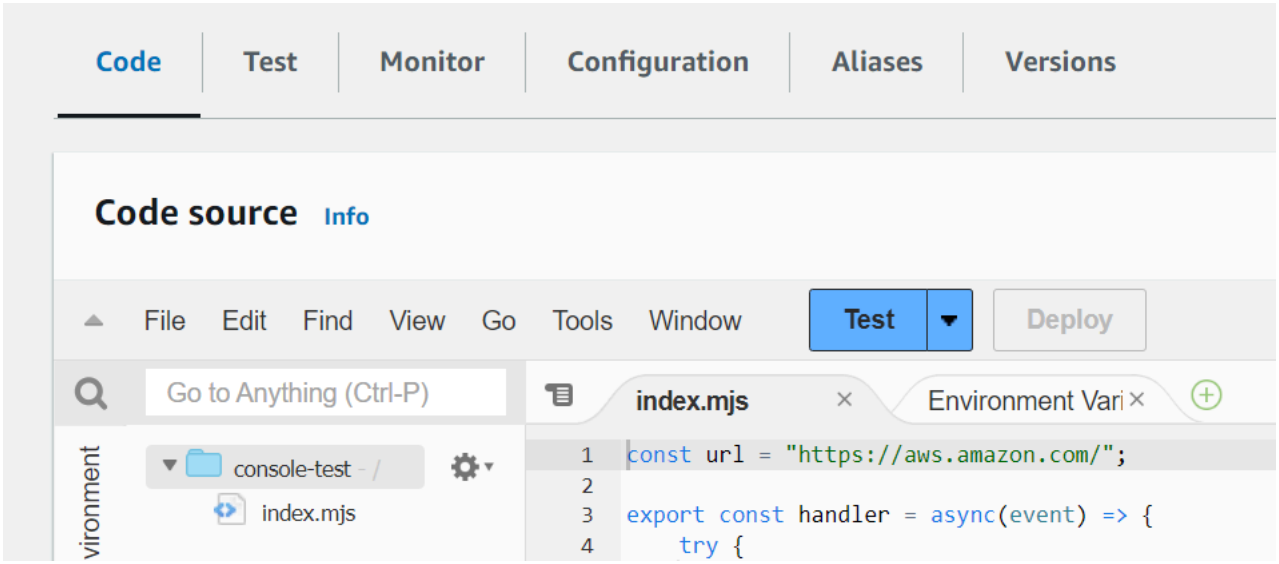
Testar a função

Use o seguinte exemplo de código para confirmar se a sua função conectada à VPC pode acessar a Internet pública. Se for bem-sucedido, o código retornará um código de status `200`. Se não for bem-sucedido, o tempo-limite da função esgotará.

Node.js

Este exemplo usa `fetch`, que está disponível no runtime `nodejs18.x` e em versões posteriores.

1. No painel Código-fonte no console do Lambda, cole o código a seguir no arquivo index.mjs. A função faz uma solicitação HTTP GET para um endpoint público e retorna o código de resposta HTTP para testar se a função tem acesso à Internet pública.

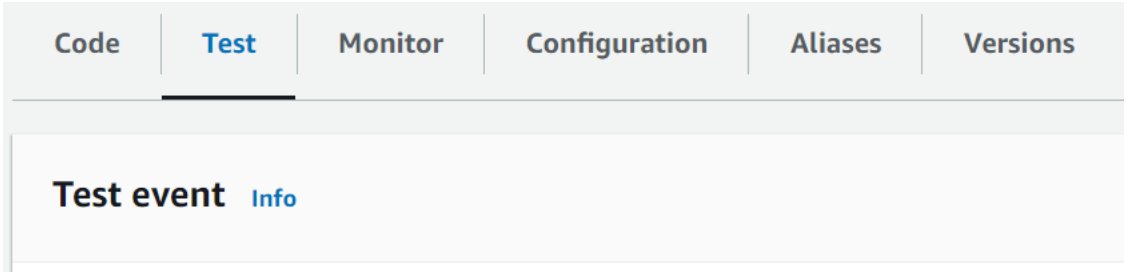


Example – Solicitação HTTP com async/await

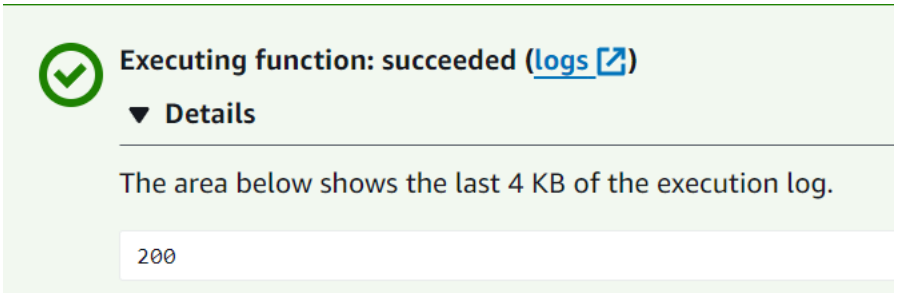
```
const url = "https://aws.amazon.com/";
```

```
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. Escolha Implantar.
3. Selecione a guia Testar.



4. Escolha Testar.
5. A função retorna um código de status `200`. Isso significa que a função tem acesso de saída à Internet.

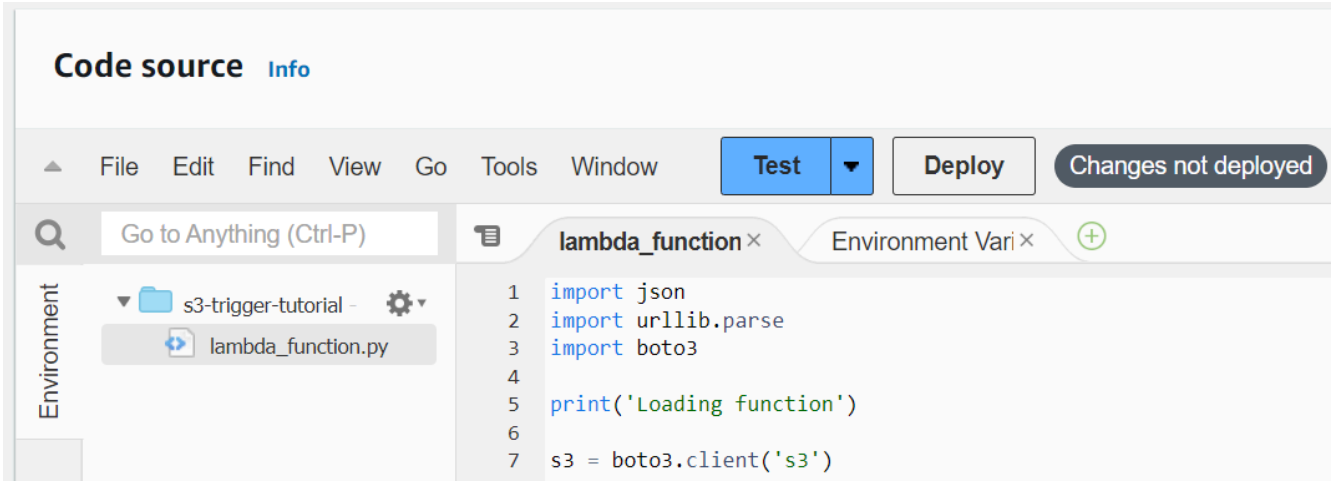


Se a função não conseguir acessar a Internet pública, você receberá uma mensagem de erro como esta:

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

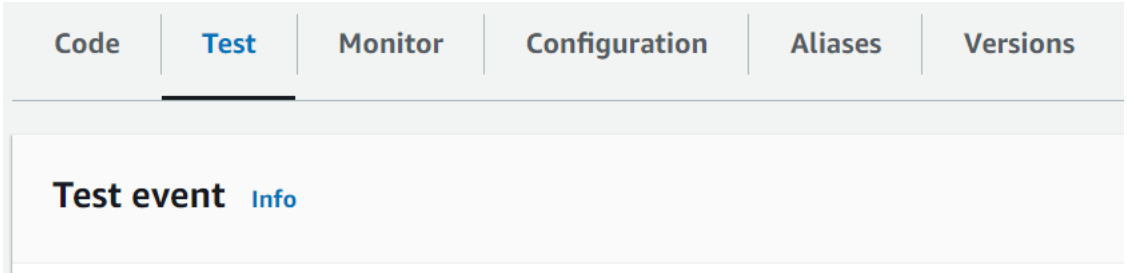
Python

1. No painel Código-fonte no console do Lambda, cole o código a seguir no arquivo lambda_function.py. A função faz uma solicitação HTTP GET para um endpoint público e retorna o código de resposta HTTP para testar se a função tem acesso à Internet pública.



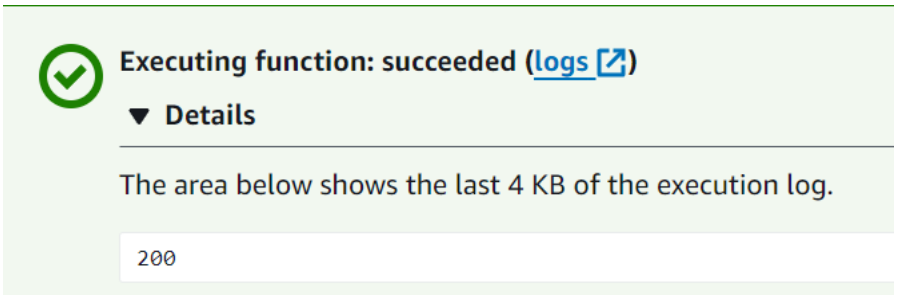

```
import urllib.request

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. Escolha Implantar.
3. Selecione a guia Testar.



4. Escolha Testar.

5. A função retorna um código de status `200`. Isso significa que a função tem acesso de saída à Internet.



Se a função não conseguir acessar a Internet pública, você receberá uma mensagem de erro como esta:

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

Como conectar endpoints da VPC de interface de entrada para o Lambda

Se você usar a Amazon Virtual Private Cloud (Amazon VPC) para hospedar os recursos da AWS, poderá estabelecer uma conexão entre a VPC e o Lambda. Você pode usar essa conexão para invocar a função do Lambda sem passar pela Internet pública.

Para estabelecer uma conexão privada entre a VPC e o Lambda, crie um endpoint da VPC de interface. Os endpoints de interface são desenvolvidos pelo AWS PrivateLink, o que permite que você acesse APIs do Lambda de forma privada, sem um gateway da Internet, um dispositivo NAT, uma conexão VPN ou uma conexão do AWS Direct Connect. As instâncias na VPC não precisam de endereços IP públicos para a comunicação com APIs do Lambda. O tráfego de rede entre a VPC e o Lambda não sai da rede da AWS.

Cada endpoint de interface é representado por uma ou mais interfaces de rede elástica nas sub-redes. Uma interface de rede fornece um endereço IP privado que serve como um ponto de entrada do tráfego para o Lambda.

Seções

Considerações para endpoints de interface do Lambda

Antes de configurar um endpoint de interface para o Lambda, analise o tópico Interface endpoint properties and limitations no Manual do usuário da Amazon VPC.

É possível chamar qualquer uma das operações de API do Lambda pela VPC. Por exemplo, você pode invocar a função do Lambda chamando a API `Invoke` de dentro da VPC. Para obter a lista completa de APIs do Lambda, consulte Ações na Referência da API do Lambda.

`use1-az3` é uma região de capacidade limitada para funções da VPC do Lambda. Não se deve usar sub-redes nessa zona de disponibilidade com funções do Lambda, pois isso pode resultar em redução da redundância da zona em caso de interrupção.

Keep-alive para conexões persistentes

O Lambda limpa conexões ociosas ao longo do tempo, portanto, é necessário usar uma diretiva de keep-alive para manter conexões persistentes. A tentativa de reutilizar uma conexão ociosa ao invocar uma função resultará em um erro de conexão. Para manter sua conexão persistente, use a diretiva keep-alive associada ao runtime. Para obter um exemplo, consulte Reutilizar Conexões com keep-alive em Node.js no Guia do desenvolvedor do AWS SDK for JavaScript.

Considerações sobre faturamento

Não há custo adicional para acessar uma função do Lambda por meio de um endpoint de interface. Para obter mais informações sobre o preço do Lambda, consulte Preço do AWS Lambda.

O preço padrão para o AWS PrivateLink aplica-se a endpoints de interface para o Lambda. Sua conta da AWS é cobrada por cada hora que um endpoint de interface é provisionado em cada zona de disponibilidade e pelos dados processados por meio do endpoint de interface. Para obter mais informações sobre preço do endpoint de interface, consulte Preço do AWS PrivateLink.

Considerações sobre emparelhamento de VPC

É possível conectar outras VPCs à VPC endpoints de interface usando oEmparelhamento de VPC. O emparelhamento de VPC é uma conexão de rede entre duas VPCs. É possível estabelecer uma conexão de emparelhamento entre suas próprias duas VPCs ou com uma VPC de outra conta da AWS. As VPCs também podem estar em duas regiões da AWS diferentes.

O tráfego entre VPCs emparelhadas permanece na rede da AWS e não passa pela Internet pública. Depois que as VPCs estiverem emparelhadas, recursos como instâncias do Amazon Elastic Compute Cloud (Amazon EC2), instâncias do Amazon Relational Database Service (Amazon RDS) ou funções do Lambda habilitadas para VPC em ambas as VPCs podem acessar a API do Lambda por meio de endpoints de interface criados em uma das VPCs.

Criar um endpoint de interface para o Lambda

Você pode criar um endpoint de interface para o Lambda usando o console da Amazon VPC ou aAWS Command Line Interface(AWS CLI). Para mais informações, consulte Criar um endpoint de interface no Guia do usuário da Amazon VPC.

Para criar um endpoint de interface para o Lambda (console)

1. Abra a página Endpoints no console da Amazon VPC.

2. Escolha Criar Endpoint.
3. Em Categoria de serviço, verifique se a opção Serviços da AWS está selecionada.
4. Em Service Name (Nome do serviço), selecione com.amazonaws.***região***.lambda. Verifique se o Type (Tipo) é Interface.
5. Escolher uma VPC e sub-redes
6. Para habilitar o DNS privado para o endpoint de interface, marque a caixa de seleção Enable DNS Name (Habilitar nome de DNS). Recomendamos que você habilite nomes DNS privados para seus endpoints da VPC para Serviços da AWS. Isso garante que as solicitações que usam os endpoints de serviço público, como solicitações feitas por meio de um AWS SDK, cheguem ao seu endpoint da VPC.
7. Em Security group (Grupo de segurança), selecione um ou mais grupos de segurança.
8. Escolha Criar endpoint.

Para usar a opção de DNS privado, defina `enableDnsHostnames` e `enableDnsSupportattributes` da VPC. Para obter mais informações, consulte Viewing and updating DNS support for your VPC no Manual do usuário da Amazon VPC. Se você habilitar o DNS privado para o endpoint de interface, poderá fazer solicitações de API para o Lambda usando seu nome DNS padrão para a região, por exemplo, `lambda.us-east-1.amazonaws.com`. Para obter mais endpoints de serviço, consulte Endpoints e cotas de serviço no Referência geral da AWS.

Para mais informações, consulte Acessar um serviço por um endpoint de interface no Guia do usuário da Amazon VPC.

Para obter informações sobre como criar e configurar um endpoint usando o AWS CloudFormation, consulte o recurso AWS::EC2::VPCEndpoint no Manual do usuário do AWS CloudFormation.

Para criar um endpoint de interface para o Lambda (AWS CLI)

Use o comando `create-vpc-endpoint` e especifique o ID da VPC, o tipo de endpoint da VPC (interface), o nome do serviço, as sub-redes que usarão o endpoint e os grupos de segurança associados às interfaces de rede do endpoint. Por exemplo:

```
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
```

Criar uma política de endpoint de interface para o Lambda

Para controlar quem pode usar o endpoint de interface e quais funções do Lambda o usuário pode acessar, é possível anexar uma política ao endpoint. Essa política especifica as seguintes informações:

- A entidade principal que pode executar ações.
- As ações que o principal pode executar.
- Os recursos nos quais o principal pode executar ações.

Para mais informações, consulte Controlar o acesso a serviços com VPC endpoints no Guia do usuário da Amazon VPC.

Exemplo: política de endpoint de interface para ações do Lambda

Veja a seguir um exemplo de uma política de endpoint para o Lambda. Quando anexada a um endpoint, essa política permite que o usuário `MyUser` invoque a função `my-function`.

> **Note**
>
> Você precisa incluir o ARN das funções qualificada e não qualificada no recurso.

```
{
   "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:*"
```

```
            ]
        }
    ]
}
```

Configurar o acesso ao sistema de arquivos para funções do Lambda

É possível configurar uma função para montar um sistema de arquivos Amazon Elastic File System (Amazon EFS) em um diretório local. Com o Amazon EFS, o código da função pode acessar e modificar os recursos compartilhados de forma segura e com alta simultaneidade.

Seções

Função de execução e permissões de usuário

Se o sistema de arquivos não tiver uma política do AWS Identity and Access Management (IAM) configurada pelo usuário, o EFS usará uma política padrão, que concede acesso total a qualquer cliente que possa se conectar ao sistema de arquivos usando um destino de montagem do sistema de arquivos. Se o sistema de arquivos tiver uma política do IAM configurada pelo usuário, a função de execução da função precisa ter permissões `elasticfilesystem`.

Permissões da função de execução

- elasticfilesystem:ClientMount
- elasticfilesystem:ClientWrite (não obrigatório para conexões somente leitura)

Essas permissões estão incluídas na política gerenciada AmazonElasticFileSystemClientReadWriteAccess. Além disso, a função de execução deve ter as permissões necessárias para conectar à VPC do sistema de arquivos.

Quando você configura um sistema de arquivos, o Lambda usa as suas permissões para verificar os destinos de montagem. Para configurar uma função para se conectar a um sistema de arquivos, seu usuário precisa das permissões a seguir:

Permissões de usuário

- elasticfilesystem:DescribeMountTargets

Configurar um sistema de arquivos e ponto de acesso

Crie um sistema de arquivos no Amazon EFS com um destino de montagem em cada zona de disponibilidade à qual a função se conecta. Para performance e resiliência, use pelo menos duas zonas de disponibilidade. Por exemplo, em uma configuração simples, é possível ter uma VPC com duas sub-redes privadas em zonas de disponibilidade separadas. A função se conecta às duas sub-redes e um destino de montagem está disponível em cada uma delas. Verifique se o tráfego de NFS (porta 2049) é permitido pelos grupos de segurança usados pela função e pelos destinos de montagem.

> **Note**
>
> Ao criar um arquivo de sistemas, você escolhe um modo de performance que não pode ser alterado posteriormente. O modo Uso geral possui menor latência e o modo Máx. E/S oferece suporte a um throughput e IOPS máximo mais alto. Para obter ajuda na escolha, consulte Performance do Amazon EFS no Manual do usuário do Amazon Elastic File System.

Um ponto de acesso conecta cada instância da função ao destino de montagem correto para a zona de disponibilidade à qual ele se conecta. Para obter a melhor performance, crie um ponto de acesso com um caminho não raiz e limite o número de arquivos criados em cada diretório. O exemplo a seguir cria um diretório chamado `my-function` no sistema de arquivos e define o ID do proprietário como 1001 com permissões de diretório padrão (755).

Example configuração do ponto de acesso

- Nome: `files`
- ID do usuário: `1001`
- ID do grupo: `1001`
- Caminho: `/my-function`
- Permissões: `755`
- ID de usuário do proprietário: `1001`

- ID de usuário do grupo: `1001`

Quando uma função usa o ponto de acesso, ele recebe o ID de usuário 1001 e tem acesso total ao diretório.

Para obter mais informações, consulte os seguintes tópicos no Manual do usuário do Amazon Elastic File System:

- Criar recursos para o Amazon EFS
- Trabalhar com usuários, grupos e permissões

Como conectar-se a um sistema de arquivos (console)

Uma função se conecta a um sistema de arquivos pela rede local em uma VPC. As sub-redes às quais a função se conecta podem ser as mesmas sub-redes que contêm pontos de montagem para o sistema e arquivos, ou sub-redes na mesma zona de disponibilidade que pode rotear o tráfego de NFS (porta 2049) para o sistema de arquivos.

> **Note**
>
> Se a função ainda não estiver conectada a uma VPC, consulte Conceder acesso a funções do Lambda para recursos em uma Amazon VPC.

Como configurar o acesso ao sistema de arquivos

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e Files systems (Sistemas de arquivos).
4. Em Sistema de arquivos, escolha Adicionar sistema de arquivos.
5. Configure as seguintes propriedades:
   - Sistema de arquivos do EFS: o ponto de acesso de um sistema de arquivos na mesma VPC.
   - Caminho de montagem local: o local onde o sistema de arquivos está montado na função do Lambda, começando com `/mnt/`.

Definição de preço

O Amazon EFS cobra pelo armazenamento e pelo throughput, com taxas que variam de acordo com a classe de armazenamento. Para obter mais detalhes, consulte .Preço do Amazon EFS.

O Lambda cobra pela transferência de dados entre VPCs. Isso se aplica somente se a VPC da função for emparelhada com outra VPC em um sistema de arquivos. As taxas são as mesmas para a transferência de dados do Amazon EC2 entre VPCs na mesma região. Para obter detalhes, consulte Preço do Lambda.

Para obter mais informações sobre integração do Lambda com o Amazon EFS, consulteUso do Amazon EFS com o Lambda.

Usar um sistema de arquivos do Amazon EFS em outra Conta da AWS para uma função do Lambda

Você pode configurar uma função para montar um sistema de arquivos do Amazon EFS em outra Conta da AWS. Antes de montar o sistema de arquivos, você deve garantir o seguinte:

- o emparelhamento da VPC deve ser configurado e as rotas apropriadas devem ser adicionadas às tabelas de rotas em cada VPC.
- O grupo de segurança do sistema de arquivos do Amazon EFS que você deseja montar deve ser configurado para permitir acesso de entrada do grupo de segurança associado à sua função do Lambda.
- As sub-redes devem ser criadas em cada VPC com as IDs de zona de disponibilidade (AZ) correspondentes.
- Os nomes de host DNS devem ser ativados nas duas VPCs.

Para que sua função do Lambda acesse um sistema de arquivos do Amazon EFS em outra Conta da AWS, esse sistema de arquivos também deve ter uma política de sistema de arquivos que conceda permissão para sua função. Para saber como criar uma política de sistema de arquivos, consulte Creating file system policies no Guia do usuário do Amazon Elastic File System.

Veja a seguir um exemplo de política que dá às funções do Lambda em uma conta especificada permissão para realizar todas as ações de API em um sistema de arquivos.

```
{
    "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**
>
> O exemplo de política mostrado usa o caractere curinga (“*”) para conceder permissões para funções do Lambda na Conta da AWS especificada para realizar qualquer operação de API no sistema de arquivos. Isso inclui excluir o sistema de arquivos. Para limitar as operações que outras Contas da AWS podem executar em seu sistema de arquivos, especifique explicitamente as ações que você deseja permitir. Para obter uma lista de possíveis operações de API, consulte Ações, recursos e chaves de condição do Amazon Elastic File System.

Para configurar a montagem do sistema de arquivos entre contas, você usa a operação `update-function-configuration` da AWS Command Line Interface (AWS CLI).

Para montar um sistema de arquivos em outra Conta da AWS, execute o comando a seguir. Use seu próprio nome de função e substitua o nome do recurso da Amazon (ARN) pelo ARN do ponto de acesso do Amazon EFS para o sistema de arquivos que você deseja montar. `LocalMountPath` é o caminho pelo qual a função pode acessar o sistema de arquivos, começando com `/mnt/`. Certifique-se de que o caminho de montagem do Lambda corresponda ao caminho do ponto de acesso do sistema de arquivos. Por exemplo, se o ponto de acesso for `/efs`, o caminho de montagem do Lambda deverá ser `/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
```

Criar um alias para uma função do Lambda

É possível criar aliases para sua função do Lambda. Um alias do Lambda é um ponteiro para uma versão da função que pode ser atualizada. Os usuários da função podem acessar a versão da função usando o nome do recurso da Amazon (ARN) do alias. Ao implantar uma nova versão, é possível atualizar o alias para usar a nova versão ou dividir o tráfego entre duas versões.

Seções

Como criar um alias da função (console)

Você pode criar um alias da função usando o console do Lambda.

Para criar um alias

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Aliases e, em seguida, escolha Create alias (Criar alias).
4. Na página Create alias (Criar alias), faça o seguinte:
    a. Insira um Name (Nome) para o alias.
    b. (Opcional) Insira uma Description (Descrição) do alias.
    c. Em Version (Versão), escolha uma versão da função para a qual você deseja que o alias indique.
    d. (Opcional) Para configurar o roteamento no alias, expanda Weighted alias (Alias ponderado). Para ter mais informações, consulte Configuração de roteamento de alias.
    e. Escolha Salvar.

Gerenciar aliases com a API do Lambda

Para criar um alias usando a AWS Command Line Interface (AWS CLI), use o comando `create-alias`.

```
aws lambda create-alias --function-name my-function --name alias-name --function-version version-number --description " "
```

Para alterar um alias para indicar uma nova versão da função, use o comando `update-alias`.

```
aws lambda update-alias --function-name my-function --name alias-name --function-version version-number
```

Para excluir um alias, use o comando `delete-alias`.

```
aws lambda delete-alias --function-name my-function --name alias-name
```

Os comandos da AWS CLI nas etapas anteriores correspondem às seguintes operações de API do Lambda:

- CreateAlias
- UpdateAlias
- DeleteAlias

Gerenciamento de aliases com o AWS SAM e o AWS CloudFormation

É possível criar e gerenciar aliases de função usando o AWS Serverless Application Model (AWS SAM) e o AWS CloudFormation.

Para saber como declarar um alias de função em um modelo do AWS SAM, consulte a página AWS::Serverless::Function no Guia do desenvolvedor do AWS SAM. Para obter informações sobre como criar e configurar aliases usando o AWS CloudFormation, consulte AWS::Lambda::Alias no Guia do usuário do AWS CloudFormation.

Usar aliases

Cada alias tem um ARN exclusivo. Um alias pode apontar somente para uma versão de função, e não para outro alias. O alias pode ser atualizado para apontar para uma nova versão da função.

Fontes de eventos como Amazon Simple Storage Service (Amazon S3) invocam sua função do Lambda. Essas origens de evento mantêm um mapeamento que identifica a função a ser invocada quando ocorrem eventos. Se você especificar um alias da função do Lambda na configuração de mapeamento, não precisará atualizar o mapeamento quando a versão da função for alterada. Para ter mais informações, consulte Como o Lambda processa registros de origens de eventos baseadas em fluxos e filas.

Em uma política de recursos, é possível conceder permissões para fontes de eventos para usar sua função do Lambda. Se um ARN de alias for especificado na política, não será necessário atualizar a política quando a versão da função for alterada.

Políticas de recursos

É possível usar uma política baseada em recurso para conceder acesso a um serviço, recurso ou conta à sua função. O escopo dessa permissão depende se você a aplica a um alias, uma versão ou a toda a função. Por exemplo, se você usar um nome de alias (como `helloworld:PROD`), a permissão permite invocar a função `helloworld` usando o ARN do alias (`helloworld:PROD`).

Se você tentar invocar a função sem um alias ou uma versão específica, receberá um erro de permissão. Esse erro de permissão ainda ocorrerá mesmo se você tentar invocar diretamente a versão da função associada ao alias.

Por exemplo, o comando da AWS CLI a seguir concede permissões do Amazon S3 para invocar o alias PROD da função `helloworld` quando o Amazon S3 está agindo em nome de `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
```

Para obter mais informações sobre como usar nomes de recursos em políticas, consulte Ajustar as seções de Recursos e Condições das políticas.

Configuração de roteamento de alias

Use a configuração de roteamento em um alias para enviar uma parte do tráfego para uma segunda versão de função. Por exemplo, você pode reduzir o risco de implantar uma nova versão

configurando o alias para enviar a maior parte do tráfego para a versão existente e apenas uma pequena porcentagem de tráfego para a nova versão.

Observe que o Lambda usa um modelo probabilístico simples para distribuir o tráfego entre as duas versões de função. Em níveis de tráfego baixos, você pode ver uma alta variação entre a porcentagem configurada e real de tráfego em cada versão. Se sua função usa simultaneidade provisionada, você pode evitarInvocações de transbordamentoconfigurando um número maior de instâncias de simultaneidade provisionadas durante o tempo em que o roteamento de alias está ativo.

Você pode apontar um alias para um máximo de duas versões de função do Lambda. As versões devem atender aos seguintes critérios:

- As duas versões devem ter a mesma função de execução.
- Ambas as versões devem ter a mesma configuração de fila de mensagens mortas ou configuração de nenhuma fila de mensagens mortas.
- Ambas as versões devem ser publicadas. O alias não pode apontar para `$LATEST`.

Para configurar o roteamento em um alias

> **Note**
>
> Verifique se a função tem pelo menos duas versões publicadas. Para criar versões adicionais, siga as instruções em Versões da função do Lambda.

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Aliases e, em seguida, escolha Create alias (Criar alias).
4. Na página Create alias (Criar alias), faça o seguinte:
    a. Insira um Name (Nome) para o alias.
    b. (Opcional) Insira uma Description (Descrição) do alias.
    c. Em Version (Versão), escolha a primeira versão da função para a qual você deseja que o alias aponte.
    d. Expanda Weighted alias (Alias ponderado).

e. Em Additional version (Versão adicional), escolha a segunda versão da função para a qual você deseja que o alias aponte.

f. Em Weight (%) (Peso (%)), insira um valor de peso para a função. Peso é a porcentagem do tráfego atribuído a essa versão quando o alias é invocado. A primeira versão recebe o peso residual. Por exemplo, se você especificar 10% para Additional version (Versão adicional), a primeira versão receberá automaticamente a atribuição de 90 por cento.

g. Escolha Salvar.

Configurar o roteamento de alias usando a CLI

Use os comandos `create-alias` e `update-alias` da AWS CLI para configurar os pesos de tráfego entre duas versões de uma função. Ao criar ou atualizar o alias, o peso do tráfego é especificado no parâmetro `routing-config`.

O exemplo a seguir cria um alias de função do Lambda chamado routing-alias que aponta para a versão 1 da função. A versão 2 da função recebe 3 por cento do tráfego. Os 97 por cento do tráfego restantes são roteados para a versão 1.

```
aws lambda create-alias --name routing-alias --function-name my-function --function-
version 1  \
--routing-config AdditionalVersionWeights={"2"=0.03}
```

Use o comando `update-alias` para aumentar a porcentagem de tráfego de entrada para a versão 2. No exemplo a seguir, o tráfego é aumentado para 5%.

```
aws lambda update-alias --name routing-alias --function-name my-function  \
--routing-config AdditionalVersionWeights={"2"=0.05}
```

Para rotear todo o tráfego para a versão 2, use o comando `update-alias` para alterar a propriedade `function-version` para apontar o alias para a versão 2. O comando também redefine a configuração de roteamento.

```
aws lambda update-alias --name routing-alias --function-name my-function  \
--function-version 2 --routing-config AdditionalVersionWeights={}
```

Os comandos da AWS CLI nas etapas anteriores correspondem às seguintes operações de API do Lambda:

- CreateAlias
- UpdateAlias

Determinar qual versão foi invocada

Ao configurar pesos de tráfego entre duas versões da função, há duas maneiras de determinar a versão da função do Lambda que foi chamada:

- CloudWatch Logs: o Lambda emite automaticamente uma entrada de log `START` que contém o ID da versão invocada para Amazon CloudWatch Logs em cada invocação da função. Veja um exemplo a seguir:

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

  Para invocações de alias, o Lambda usa a dimensão `Executed Version` para filtrar os dados de métrica pela versão invocada. Para ter mais informações, consulte Trabalhar com métricas de funções Lambda.
- Carga de resposta (invocações síncronas) – As respostas às invocações de função síncrona incluem um cabeçalho `x-amz-executed-version` para indicar qual versão de função foi invocada.

Versões da função do Lambda

É possível usar versões para gerenciar a implantação das suas funções. Por exemplo, você pode publicar uma nova versão de uma função para testes beta sem afetar os usuários da versão de produção estável. O Lambda cria uma nova versão da sua função sempre que você publicá-la. A nova versão é uma cópia da versão não publicada da função. A versão não publicada é chamada $LATEST.

> **Note**
>
> Para criar uma nova versão da sua função, é necessário primeiro fazer alterações na versão não publicada ($LATEST). Essas alterações podem incluir atualizar o código ou modificar as configurações. Se a $LATEST for idêntica a uma versão publicada anteriormente, você não poderá criar uma nova versão até implantar alterações na $LATEST.

Depois de publicar uma versão da função, o código, o runtime, a arquitetura, a memória, as camadas e a maioria das outras definições de configuração da função são imutáveis. Isso significa que você não pode alterar essas configurações sem publicar uma nova versão de $LATEST. Você pode configurar os seguintes itens para uma versão de função publicada:

- Triggers
- Destinos
- Simultaneidade provisionada
- Invocação assíncrona
- Conexões e proxies de banco de dados

> **Note**
>
> Ao usar os controles de gerenciamento de runtime com o modo Automático, a versão do runtime usada pela versão da função é atualizada automaticamente. Ao usar Function update (Atualização de função) ou o modo Manual, a versão de runtime não será atualizada. Para ter mais informações, consulte the section called “Atualizações de runtime”.

Seções

- Como criar versões de função
- Usar versões
- Conceder permissões

Como criar versões de função

O código e as configurações das funções somente podem ser alterados na versão não publicada de uma função. Quando você publica uma versão, o Lambda bloqueia o código e a maioria das configurações para manter uma experiência consistente para os usuários dessa versão.

Você pode criar uma versão de função usando o console do Lambda.

Para criar uma nova versão de função

1. Abra a página Funções do console do Lambda.
2. Escolha uma função e, em seguida, Versions (Versões).
3. Na página de configuração de versões, escolha Publish new version (Publicar nova versão).
4. (Opcional) Insira uma descrição de versão.
5. Selecione Publish.

Como alternativa, é possível publicar uma versão de uma função usando a operação de API PublishVersion.

O comando da AWS CLI a seguir publica uma nova versão de uma função. A resposta retorna as informações de configuração sobe a versão da função, incluindo o número da versão e o ARN da função com o sufixo da versão.

```
aws lambda publish-version --function-name my-function
```

A seguinte saída deverá ser mostrada:

```
{
  "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**
>
> O Lambda atribui números de sequência monotônicos crescentes para o controle de versionamento. O Lambda nunca reutiliza números de versão, mesmo após você excluir e recriar uma função.

Usar versões

É possível fazer referência à sua função do Lambda usando um ARN qualificado ou um ARN não qualificado.

- ARN qualificado: o ARN da função com um sufixo da versão. O exemplo a seguir faz referência à versão 42 da função `helloworld`.

  ```
  arn:aws:lambda:aws-region:acct-id:function:helloworld:42
  ```

- ARN não qualificado: o ARN da função sem um sufixo da versão.

  ```
  arn:aws:lambda:aws-region:acct-id:function:helloworld
  ```

É possível usar um ARN qualificado ou não qualificado em todas as operações de API relevantes. No entanto, não é possível usar um ARN não qualificado para criar um alias.

Se você decidir não publicar versões de funções, será possível invocar a função usando o ARN qualificado ou não qualificado no mapeamento da origem do evento. Quando você invoca uma função usando um ARN não qualificado, o Lambda invoca implicitamente $LATEST.

O Lambda só publica uma nova versão da função se o código nunca tiver sido publicado ou se o código foi alterado em relação à última versão publicada. Se não houver nenhuma alteração, a versão da função permanecerá na última versão publicada.

O ARN qualificado para cada versão de função do Lambda é exclusivo. Depois de publicar uma versão, não é possível alterar o ARN ou o código da função.

Conceder permissões

É possível usar uma política baseada em recurso ou uma política baseada em identidade para conceder acesso à sua função. O escopo da permissão depende se você aplicar a política a uma função ou a uma versão de uma função. Para obter mais informações sobre nomes de recursos de função em políticas, consulte Ajustar as seções de Recursos e Condições das políticas.

É possível simplificar o gerenciamento de fontes de eventos e políticas do AWS Identity and Access Management (IAM) utilizando aliases de funções. Para ter mais informações, consulte Criar um alias para uma função do Lambda.

Configuração de uma função do Lambda para o streaming de respostas

É possível configurar seus URLs de função do Lambda para fazer o streaming de cargas de resposta de volta aos clientes. O streaming de respostas pode beneficiar aplicações sensíveis à latência ao melhorar o desempenho do tempo até o primeiro byte (TTFB). Isso ocorre porque é possível enviar respostas parciais de volta ao cliente assim que elas se tornarem disponíveis. Além disso, é possível usar o streaming de respostas para construir funções que retornem cargas maiores. As cargas de streams de resposta têm um limite flexível de 20 MB em comparação com o limite de 6 MB para respostas armazenadas em buffer. O streaming de uma resposta também significa que sua função não precisa caber em toda a resposta na memória. Para respostas muito grandes, isso pode reduzir a quantidade de memória que você precisa configurar para sua função.

A velocidade com que o Lambda faz o streaming das suas respostas depende do tamanho da resposta. A taxa de streaming dos primeiros 6 MB da resposta da sua função é ilimitada. Para respostas maiores que 6 MB, o restante da resposta está sujeito a um limite de largura de banda. Para obter mais informações sobre largura de banda de streaming, consulte Limites de largura de banda para streaming de resposta.

O streaming de respostas gera um custo. Para obter mais informações, consulte Preços do AWS Lambda.

O Lambda oferece suporte ao streaming de respostas em runtimes gerenciados por Node.js. Em outras linguagens, é possível usar um runtime personalizado com uma integração personalizada da API de runtime para transmitir respostas ou usar o Lambda Web Adapter. Você pode transmitir respostas por meio de URLs da função Lambda, do AWS SDK ou usando a API Lambda. InvokeWithResponseStream

> **Note**
>
> Ao testar sua função por meio do console do Lambda, você sempre verá as respostas como armazenadas em buffer.

Escrita de funções habilitadas para o streaming de respostas

Escrever o manipulador para funções de streaming de resposta é diferente dos padrões típicos do manipulador. Ao escrever funções de streaming, certifique-se de fazer o seguinte:

- Encapsule sua função com o decorador `awslambda.streamifyResponse()` fornecido pelos runtimes nativos do Node.js.
- Encerre a stream adequadamente para garantir que todo o processamento de dados seja concluído.

Configuração de uma função de manipulador para o streaming de respostas

Para indicar ao runtime que o Lambda deve fazer o streaming das respostas da sua função, é necessário encapsular sua função com o decorador `streamifyResponse()`. Isso faz com que o runtime use o caminho lógico adequado para o streaming de respostas e habilita a função a fazer o streaming das respostas.

O decorador `streamifyResponse()` aceita uma função que aceita os seguintes parâmetros:

- `event`: fornece informações sobre o evento de invocação do URL da função, como o método HTTP, os parâmetros da consulta e o corpo da solicitação.
- `responseStream`: fornece um stream gravável.
- `context`: fornece métodos e propriedades com informações sobre a invocação, a função e o ambiente de execução.

O objeto `responseStream` é um `writableStream de Node.js`. Como em qualquer stream desse tipo, use o método `pipeline()`.

Example manipulador habilitado para streaming de resposta

```
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);
});
```

Embora `responseStream` ofereça o método `write()` para gravar no stream, recomendamos que você use `pipeline()` sempre que possível. O uso de `pipeline()` garante que o stream gravável não seja sobrecarregado por um stream legível mais rápido.

Encerramento do stream

Certifique-se de encerrar o stream corretamente antes que o manipulador retorne. O método `pipeline()` lida com isso automaticamente.

Para outros casos de uso, chame o método `responseStream.end()` para encerrar corretamente um stream. Esse método sinaliza que nenhum outro dado deve ser gravado no stream. Esse método não é necessário se você gravar no stream com `pipeline()` ou `pipe()`.

Example Exemplo de encerramento de um stream com pipeline()

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

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

Example Exemplo de encerramento de um stream sem pipeline()

```
exports.handler = awslambda.streamifyResponse(async (event, responseStream, _context)
 => {
  responseStream.write("Hello ");
  responseStream.write("world ");
  responseStream.write("from ");
  responseStream.write("Lambda!");
  responseStream.end();
});
```

Invocação de uma função habilitada para streaming de resposta usando URLs de função do Lambda

> **Note**
>
> É necessário invocar sua função usando um URL da função para fazer o streaming das respostas.

É possível invocar funções habilitadas para streaming de resposta alterando o modo de invocação do URL da função. O modo de invocação determina qual operação de API o Lambda usa para invocar sua função. Os modos de invocação disponíveis são:

- `BUFFERED`: esta é a opção padrão. O Lambda invoca sua função usando a operação `Invoke` da API. Os resultados da invocação estarão disponíveis quando a carga estiver concluída. O tamanho máximo da carga é de 6 GB.
- `RESPONSE_STREAM`: permite que sua função faça o streaming dos resultados da carga assim que eles se tornem disponíveis. O Lambda invoca sua função usando a operação `InvokeWithResponseStream` da API. O tamanho máximo da carga de resposta é de 20 GB. Contudo, é possível solicitar um aumento de cota.

Você ainda pode invocar sua função sem streaming de resposta chamando diretamente a operação `Invoke` da API. No entanto, o Lambda faz o streaming de todas as cargas de resposta para invocações que passam pela URL da função até que você altere o modo de invocação para `BUFFERED`.

Para definir o modo de invocação de um URL da função (console)

1. Abra a página Funções do console do Lambda.
2. Escolha o nome da função para a qual você deseja definir o modo de invocação.
3. Escolha a guia Configuration (Configuração) e depois Function URL (URL de função).
4. Escolha Editar e, em seguida, escolha Configurações adicionais.
5. Em Modo de invocação, escolha o modo de invocação desejado.
6. Escolha Salvar.

Para definir o modo de invocação de um URL da função (AWS CLI)

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

Para definir o modo de invocação de um URL da função (AWS CloudFormation)

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

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

Para obter mais informações sobre a configuração de URLs de função, consulte URLs de função do Lambda.

Limites de largura de banda para streaming de resposta

Os primeiros 6 MB da carga útil de resposta da sua função têm largura de banda ilimitada. Após essa intermitência inicial, o Lambda faz o streaming da sua resposta a uma taxa máxima de 2 MBps. Se as respostas da sua função nunca excederem 6 MB, esse limite de largura de banda nunca será aplicado.

> **Note**
>
> Os limites de largura de banda se aplicam somente à carga útil da resposta da sua função e não ao acesso à rede por sua função.

A taxa de largura de banda ilimitada varia de acordo com diversos fatores, incluindo a velocidade de processamento da sua função. Normalmente, você pode esperar uma taxa superior a 2 MBps para os primeiros 6 MB da resposta da sua função. Se sua função estiver fazendo o streaming de uma resposta para um destino fora da AWS, a taxa de streaming também dependerá da velocidade da conexão externa com a Internet.

Tutorial: criação de um função do Lambda de streaming de resposta com um URL da função

Neste tutorial, você criará uma função do Lambda definida como um arquivo .zip com um endpoint público do URL da função que retorna um fluxo de resposta. Para obter mais informações sobre a configuração de URLs de função, consulte Criar e gerenciar URLs de função.

Pré-requisitos

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em Criar uma função do Lambda com o console para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, a AWS Command Line Interface (AWS CLI) versão 2 será necessária. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

A seguinte saída deverá ser mostrada:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell.

> **Note**
>
> No Windows, alguns comandos da CLI do Bash que você costuma usar com o Lambda (como `zip`) não são compatíveis com os terminais integrados do sistema operacional. Para obter uma versão do Ubuntu com o Bash integrada no Windows, instale o Subsistema do Windows para Linux. Os exemplos de comandos da CLI neste guia usam a formatação Linux. Os comandos que incluem documentos JSON em linha deverão ser reformatados se você estiver usando a CLI do Windows.

Criar uma função de execução

Crie a função de execução que dá à sua função do Lambda permissão para acessar recursos da AWS.

Para criar uma função de execução

1. Abra a página Roles (Funções) no console do AWS Identity and Access Management (IAM).
2. Selecione Criar função.
3. Crie uma função com as seguintes propriedades:
   - Tipo de entidade confiável: serviço da AWS
   - Caso de uso: Lambda
   - Permissões: AWSLambdaBasicExecutionRole

- Role name (Nome da função): **`response-streaming-role`**

A política AWSLambdaBasicExecutionRole tem as permissões de que a função necessita para gravar logs no Amazon CloudWatch Logs. Após criar o perfil, anote seu nome do recurso da Amazon (ARN). Você precisará dele na próxima etapa.

Crie uma função de streaming de respostas (AWS CLI)

Crie uma função do Lambda de streaming resposta com um endpoint de URL da função usando a AWS Command Line Interface (AWS CLI).

Para criar uma função que possa fazer o streaming de respostas

1. Copie o exemplo de código a seguir em um arquivo com o nome `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. Crie um pacote de implantação.

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

3. Crie uma função do Lambda com o comando `create-function`. Substitua o valor de `--role` pelo ARN do perfil da etapa anterior.

```
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
```

Para criar um URL da função

1. Adicione uma política baseada em recurso à sua função para permitir acesso público ao URL da função. Substitua o valor de `--principal` pela sua ID da Conta da 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. Crie um endpoint de URL para a função com o comando `create-function-url-config`.

```
aws lambda create-function-url-config \
  --function-name my-streaming-function \
  --auth-type AWS_IAM \
  --invoke-mode RESPONSE_STREAM
```

Testar o endpoint de URL de função

Teste sua integração invocando sua função. É possível abrir o URL da sua função em um navegador, ou usar o curl.

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

A URL da nossa função usa o tipo de autenticação `IAM_AUTH`. Isso significa que você precisa assinar solicitações com sua chave de acesso e chave secreta da AWS. No comando anterior, substitua `<key:token>` pela ID da chave de acesso da AWS. Insira sua chave secreta da AWS, quando solicitada. Se você não tiver sua chave secreta da AWS, é possível usar credenciais da AWS temporárias em vez disso.

Limpe os recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Para excluir a função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Escolha Excluir.
4. Insira o nome do perfil no campo de entrada de texto e escolha Delete (Excluir).

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Implantar funções do Lambda

Você pode implantar código nas funções do Lambda carregando um arquivo .zip ou criando e carregando uma imagem de contêiner.

Tópicos

Arquivos .zip

Um arquivo .zip inclui o código da aplicação e as dependências dele. Quando você cria funções usando o console do Lambda ou um toolkit, o Lambda cria automaticamente um arquivo .zip do seu código.

Quando você cria funções com a API do Lambda, ferramentas da linha de comando ou AWS SDKs, é necessário criar o pacote de implantação. Você também deve criar um pacote de implantação se sua função usar uma linguagem compilada ou para adicionar dependências a ela. Para implantar o código da sua função, faça upload do pacote de implantação do Amazon Simple Storage Service (Amazon S3) ou de sua máquina local.

Você pode fazer o upload de um arquivo .zip como seu pacote de implantação usando o console do Lambda,AWS Command Line Interface(AWS CLI) ou para um bucket do Amazon Simple Storage Service (Amazon S3).

Permissões de arquivos do pacote de implantação

O runtime do Lambda precisa de permissão para ler os arquivos no pacote de implantação. Na notação octal de permissões do Linux, o Lambda precisa de 644 permissões para arquivos não executáveis (rw-r--r--) e 755 permissões (rwxr-xr-x) para diretórios e arquivos executáveis.

No Linux e no MacOS, use o comando `chmod` para alterar as permissões de arquivo em arquivos e diretórios do seu pacote de implantação. Por exemplo, para dar a um arquivo executável as permissões corretas, execute o comando a seguir.

```
chmod 755 <filepath>
```

Para alterar as permissões de arquivo no Windows, consulte Set, View, Change, or Remove Permissions on an Object na documentação do Microsoft Windows.

Imagens de contêiner

Você pode empacotar seu código e suas dependências como uma imagem de contêiner usando ferramentas como a command line interface (CLI – interface da linha de comando) do Docker. Em seguida, você pode fazer upload da imagem para o registro do contêiner hospedado no Amazon Elastic Container Registry (Amazon ECR).

Quando você invoca a função, o Lambda implanta a imagem do contêiner em um ambiente de execução. O Lambda inicializa qualquerextensõesE, em seguida, executa o código de inicialização da função (o código fora do manipulador principal). Observe que a duração da inicialização da função está incluída no tempo de execução cobrado.

O Lambda então executa a função chamando o ponto de entrada de código especificado na configuração da função (as configurações de imagem de contêiner ENTRYPOINT e CMD).

A AWS oferece um conjunto de imagens de base de código aberto que você pode usar para criar a imagem do contêiner para o código de função. Você também pode usar imagens de base alternativas de outros registros de contêiner. A AWS também oferece um cliente de runtime de código aberto que você adiciona à sua imagem de base alternativa para torná-la compatível com o serviço do Lambda.

Além disso, a AWS oferece um emulador de interface de runtime para você testar as funções localmente usando ferramentas como a CLI do Docker.

> **Note**
>
> Você cria cada imagem de contêiner para ser compatível com uma das arquiteturas de conjunto de instruções compatíveis com o Lambda. O Lambda fornece imagens básicas para cada uma das arquiteturas do conjunto de instruções. O Lambda também fornece imagens básicas que suportam ambas as arquiteturas.
> A imagem que você criar para sua função deverá ser direcionada para apenas uma das arquiteturas.

Não há cobrança adicional para empacotar e implantar funções como imagens de contêiner. Quando uma função implantada como uma imagem de contêiner é invocada, você paga por solicitações de invocação e duração da execução. Você será cobrado com relação ao armazenamento de imagens de contêiner no Amazon ECR. Para obter mais informações, consulte a Definição de preço do Amazon ECR.

Segurança de imagem

Quando o Lambda faz download pela primeira vez da imagem do contêiner da fonte original (Amazon ECR), a imagem do contêiner é otimizada, criptografada e armazenada usando métodos de criptografia convergentes autenticados. Todas as chaves necessárias para descriptografar os dados do cliente são protegidas usando chaves gerenciadas pelo cliente AWS KMS. Para acompanhar e auditar o uso de chaves gerenciadas pelo cliente do Lambda, você pode visualizar os AWS CloudTrail logs.

Implantar funções do Lambda como arquivos .zip

Quando você cria uma função do Lambda, você empacota o código da função em umpacote de implantação. O Lambda é compatível com dois tipos de pacotes de implantação: imagens de contêiner e arquivos .zip. O fluxo de trabalho para criar uma função depende do tipo de pacote de implantação. Para criar uma função definida como uma imagem de contêiner, consulte the section called “Imagens de contêiner”.

Você pode usar o console do Lambda e a API do Lambda para criar uma função definida com um arquivo de arquivo.zip. Você também pode carregar um arquivo.zip atualizado para alterar o código da função.

> **Note**
>
> Você não pode alterar o tipo de pacote de implantação (.zip ou imagem de contêiner) de uma função existente. Por exemplo, você não pode converter uma função de imagem de contêiner para usar um arquivo compactado .zip. É necessário criar uma nova função.

Tópicos

Como criar a função

Ao criar uma função definida com um arquivo de arquivo .zip, você escolhe um modelo de código, a versão de idioma e a função de execução da função. Você adiciona o código da função depois do Lambda cria a função.

Para criar a função

1. Abra a página Funções do console do Lambda.

2. Escolha a opção Criar função.
3. SelecioneAuthor from scratch (Começar do zero)ouUse a blueprint (Usar um esquema)Como criar sua função do.
4. Em Basic information (Informações básicas), faça o seguinte:
   a. Em Function name (Nome da função), insira o nome da função. Os nomes das funções têm um limite de 64 caracteres de comprimento.
   b. Para o Runtime, escolha a versão do idioma a ser usada para sua função.
   c. (Opcional) Em Architecture (Arquitetura), escolha a arquitetura do conjunto de instruções a ser usado para sua função. O valor da arquitetura padrão é X86_64. Ao criar o pacote de implantação para sua função, verifique se ela é compatível com esta arquitetura de conjunto de instruções.
5. (Opcional) Em Permissões, expanda Alterar função de execução padrão. Crie uma função de execução ou use uma existente.
6. (Opcional) Expanda Advanced settings (Configurações avançadas). Você pode escolher umConfiguração de assinatura de códigopara a função do. Você também pode configurar um (Amazon VPC) para que a função acesse.
7. Escolha a opção Criar função.

O Lambda cria a nova função. Agora é possível usar o console do para adicionar o código da função e configurar outros parâmetros e recursos da função. Para obter instruções sobre implantação de código, consulte a página do manipulador de runtime que a função usa.

Node.js

Implantar funções do Lambda em Node.js com arquivos .zip

Python

Trabalhar com arquivos .zip para funções do Lambda em Python

Ruby

Como trabalhar com arquivos .zip para funções do Lambda em Ruby

Java

Implantar funções do Lambda em Java com arquivos .zip ou JAR

Go

- Implantar funções do Lambda em Go com arquivos .zip

C#

- Criar e implantar funções do Lambda em C# com arquivos .zip

PowerShell

- Implemente funções PowerShell Lambda com arquivos de arquivos.zip

Usando o editor de código do console

O console cria uma função do Lambda com um único arquivo de origem. Para linguagens de desenvolvimento de scripts, você pode editar esse arquivo e adicionar mais arquivos usando o editor de códigos integrado. Para salvar suas alterações, selecione Salvar. Em seguida, para executar seu código, escolhaTeste.

> **Note**
>
> O console do Lambda usa o AWS Cloud9 para fornecer um ambiente de desenvolvimento integrado no navegador. Você também pode usar o AWS Cloud9 para desenvolver funções do Lambda em seu próprio ambiente. Para obter mais informações, consulte Working with AWS Lambda functions using the AWS Toolkit no Guia do usuário do AWS Cloud9.

Quando você salva seu código de função, o console do Lambda cria um pacote de implantação de arquivos .zip. Quando desenvolver o código de função fora do console (usando um IDE), você precisará criar um pacote de implantação para carregar o código na função do Lambda.

Atualizar código de função

Para linguagens de desenvolvimento de scripts (Node.js, Python, e Ruby), você pode editar o código de sua função no editor de código incorporado. Se o tamanho do código for superior a 3 MB, se você precisar adicionar bibliotecas ou para linguagens incompatíveis com o editor (Java, Go, C#), é necessário fazer upload do código da função como um arquivo.zip. Se o arquivo .zip for menor que 50 MB, você poderá fazer upload do arquivo .zip da sua máquina local. Se o arquivo for maior que 50 MB, faça upload do arquivo para a função desde um bucket do Amazon S3.

Para carregar código de função como um arquivo.zip

1. Abra a página Funções do console do Lambda.
2. Escolha a função a ser atualizada e escolha aCódigoGuia.
3. Em Fonte de código), escolha Fazer upload de.
4. Escolha .zip file (Arquivo .zip) e, em seguida, escolha Upload (Fazer upload).
   - No seletor de arquivos, selecione a nova versão da imagem e escolha Open (Abrir) e, em seguida, Save (Salvar).
5. (Alternativa à etapa 4) EscolhaLocalização do Amazon S3.
   - Na caixa de texto, insira o URL do link do S3 do arquivamento de arquivo.zip e, depois, escolhaSave (Salvar).

Como alterar o runtime

Se você atualizar a configuração da função para usar um novo runtime, talvez seja necessário atualizar o código da função para ser compatível com o novo runtime. Ao atualizar a configuração da função para usar um runtime diferente, você deverá fornecer um novo código de função compatível com o runtime e a arquitetura. Para obter instruções de como criar um pacote de implantação para o código da função, consulte a página do manipulador para o runtime usado pela função.

Para alterar o runtime

1. Abra a página Funções do console do Lambda.
2. Escolha a função a ser atualizada e escolha aCódigoGuia.
3. Role para baixo até a seção Configurações de runtime abaixo do editor de código.
4. Selecione a opção Editar.
   a. Em Runtime, selecione o identificador de runtime.
   b. Em Handler (Manipulador), especifique o manipulador de sua função.
   c. Em Architecture (Arquitetura), escolha a arquitetura do conjunto de instruções a ser usado para sua função.
5. Escolha Salvar.

Alterar a arquitetura

Antes de alterar a arquitetura do conjunto de instruções, é necessário garantir que o código da função seja compatível com a arquitetura de destino.

Se você usa Node.js, Python ou Ruby e edita o código da sua função no editor integrado, o código existente poderá ser executado sem modificação.

No entanto, se você fornecer seu código de função usando um pacote de implantação de arquivo.zip, deverá preparar um novo arquivo .zip compilado e criado corretamente para o runtime de destino e a arquitetura do conjunto de instruções. Para obter instruções, consulte a página do manipulador do runtime da sua função.

Para alterar a arquitetura do conjunto de instruções

1. Abra a página Funções do console do Lambda.
2. Escolha a função a ser atualizada e escolha aCódigoGuia.
3. SobConfigurações de execução, escolhaEdite.
4. Em Architecture (Arquitetura), escolha a arquitetura do conjunto de instruções a ser usado para sua função.
5. Escolha Salvar.

Usar a API do Lambda

Para criar e configurar uma função que usa um arquivo .zip, use as seguintes operações de API:

- CreateFunction
- UpdateFunctionCode
- UpdateFunctionConfiguration

AWS CloudFormation

Você pode usarAWS CloudFormationComo criar uma função do Lambda que usa um arquivo .zip. No seu modelo do AWS CloudFormation, o recurso `AWS::Lambda::Function` especifica a função do Lambda. Para obter descrições das propriedades no recurso `AWS::Lambda::Function`, consulte AWS::Lambda::Function no Manual do usuário do AWS CloudFormation.

No recurso `AWS::Lambda::Function`, defina as seguintes propriedades para criar uma função definida como um arquivo .zip:

- AWS::Lambda::Function
  - PackageType: definido como `Zip`.
  - Código — Insira o nome do bucket do Amazon S3 e o nome do arquivo.zip na caixa`S3Bucket`e`S3Key`campos. Para Node.js ou Python, você pode fornecer código-fonte inline da sua função do Lambda.
  - Runtime: defina o valor do runtime.
  - Arquitetura: defina o valor da arquitetura como `arm64` para usar o processador AWS Graviton2. Por padrão, o valor da arquitetura é `x86_64`.

Criar uma função do Lambda usando uma imagem de contêiner

O código da função do AWS Lambda consiste em scripts ou programas compilados e as dependências deles. Você usa umpacote de implantaçãopara implantar seu código de função no Lambda. O Lambda é compatível com dois tipos de pacotes de implantação: imagens de contêiner e arquivos .zip.

Existem três maneiras de criar uma imagem de contêiner para uma função do Lambda:

- Usar uma imagem base da AWS para Lambda

  As imagens base da AWS são pré-carregadas com um runtime de linguagem, um cliente de interface de runtime para gerenciar a interação entre o Lambda e o código da sua função e um emulador de interface de runtime para testes locais.

- Usar uma imagem base somente para sistema operacional da AWS

  As imagens base somente para sistema operacional da AWS contêm uma distribuição do Amazon Linux e o emulador de interface de runtime. Essas imagens são comumente usadas para criar imagens de contêiner para linguagens compiladas, como Go e Rust e para uma linguagem ou versão de linguagem para a qual o Lambda não fornece uma imagem base, como Node.js 19. Você também pode usar imagens base somente para sistema operacional para implementar um runtime personalizado. Para tornar a imagem compatível com o Lambda, você deve incluir um cliente de interface de runtime para sua linguagem na imagem.

- Usar uma imagem base que não é da AWS

  Você também pode usar uma imagem base alternativa de outro registro de contêiner, como Alpine Linux ou Debian. Você também pode usar uma imagem personalizada criada por sua organização. Para tornar a imagem compatível com o Lambda, você deve incluir um cliente de interface de runtime para sua linguagem na imagem.

> **Tip**
>
> Para reduzir o tempo necessário para que as funções do contêiner do Lambda se tornem ativas, consulte Use multi-stage builds na documentação do Docker. Para criar imagens de contêiner eficientes, siga as Melhores práticas para gravar Dockerfiles.

Para criar uma função do Lambda de uma imagem de contêiner, crie sua imagem localmente e carregue-a em um repositório do Amazon Elastic Container Registry (Amazon ECR). Em seguida, especifique o URI do repositório quando criar a função. O repositório do Amazon ECR deve estar na mesma Região da AWS que a função do Lambda. É possível criar uma função usando uma imagem em uma conta da AWS diferente desde que a imagem esteja na mesma região da função do Lambda. Para ter mais informações, consulte Permissões entre contas do Amazon ECR.

Esta página explica os tipos de imagem base e os requisitos para criar imagens de contêiner compatíveis com o Lambda.

> **Note**
>
> Você não pode alterar o tipo de pacote de implantação (.zip ou imagem de contêiner) de uma função existente. Por exemplo, você não pode converter uma função de imagem de contêiner para usar um arquivo compactado .zip. É necessário criar uma nova função.

Tópicos

Requisitos

Instale a AWS Command Line Interface (AWS CLI) versão 2 e a CLI do Docker. Além disso, observe os seguintes requisitos:

- A imagem de contêiner deve implementar o API de tempo de execução do Lambda. Os clientes de interface de runtime de código aberto da AWS implementam a API. Você pode adicionar um cliente de interface de runtime à sua imagem base preferida para torná-lo compatível com o Lambda.

- Deve ser possível executar a imagem do contêiner em um sistema de arquivos somente leitura. Seu código de função pode acessar um diretório `/tmp` gravável com entre 512 MB e 10.240 MB, em incrementos de 1 MB, de armazenamento.
- O usuário padrão do Lambda deve ser capaz de ler todos os arquivos necessários para executar seu código de função. O Lambda segue as práticas recomendadas de segurança definindo um usuário padrão do Linux com permissões menos privilegiadas. Verifique se o código da aplicação não depende de arquivos que outros usuários do Linux estejam restringidos de executar.
- O Lambda é compatível apenas com imagens de contêiner baseadas em Linux.
- O Lambda fornece imagens base multiarquitetura. No entanto, a imagem que você criar para sua função deverá ser direcionada para apenas uma das arquiteturas. O Lambda não oferece suporte a funções que usam imagens de contêiner multiarquitetura.

Usar uma imagem base da AWS para Lambda

Você pode usar uma das imagens base da AWS para o Lambda criar a imagem de contêiner do código da função. As imagens de base são pré-carregadas com um runtime de linguagem e outros componentes necessários para executar uma imagem de contêiner no Lambda. Você adiciona seu código de função e as dependências à imagem base e, em seguida, empacota-os como uma imagem de contêiner.

A AWS fornece atualizações periodicamente para as imagens base da AWS para o Lambda. Se o Dockerfile incluir o nome da imagem na propriedade FROM, seu cliente Docker extrairá a versão mais recente da imagem do repositório do Amazon ECR. Para usar a imagem base atualizada, você deve reconstruir a imagem do contêiner e atualizar o código da função.

As imagens base do Node.js 20, Python 3.12, Java 21, AL2023 e versões posteriores são baseadas na imagem de contêiner mínimo do Amazon Linux 2023. Imagens base anteriores usam o Amazon Linux 2. O AL2023 oferece várias vantagens em relação ao Amazon Linux 2, incluindo uma área de implantação menor e versões atualizadas de bibliotecas, como `glibc`.

As imagens baseadas no AL2023 usam o `microdnf` (com link simbólico `dnf`) como o gerenciador de pacotes, em vez do `yum`, que é o gerenciador de pacotes padrão no Amazon Linux 2. O `microdnf` é uma implementação autônoma do `dnf`. Para obter uma lista dos pacotes incluídos nas imagens baseadas no AL2023, consulte as colunas Contêiner mínimo em Comparar pacotes instalados em imagens de contêiner do Amazon Linux 2023. Para obter mais informações sobre as diferenças entre o AL2023 e o Amazon Linux 2, consulte Introdução ao runtime do Amazon Linux 2023 para AWS Lambda no blog AWS Compute.

> **Note**
>
> Para executar imagens baseadas no AL2023 localmente, inclusive com o AWS Serverless Application Model (AWS SAM), você deve usar o Docker versão 20.10.10 ou posterior.

Para criar uma imagem de contêiner usando uma imagem base da AWS, escolha as instruções para sua linguagem preferencial:

- Node.js
- TypeScript (usa uma imagem base Node.js)
- Python
- Java
- Go
- .NET
- Ruby

Usar uma imagem base somente para sistema operacional da AWS

As imagens base somente para sistema operacional da AWS contêm uma distribuição do Amazon Linux e o emulador de interface de runtime. Essas imagens são comumente usadas para criar imagens de contêiner para linguagens compiladas, como Go e Rust e para uma linguagem ou versão de linguagem para a qual o Lambda não fornece uma imagem base, como Node.js 19. Você também pode usar imagens base somente para sistema operacional para implementar um runtime personalizado. Para tornar a imagem compatível com o Lambda, você deve incluir um cliente de interface de runtime para sua linguagem na imagem.

Tags	Runtime	Sistema operacional	Dockerfile	Desaprovação
al2023	Runtime somente para sistema operacional	Amazon Linux 2023	Dockerfile para runtime somente para sistema operacional no GitHub	

Tags	Runtime	Sistema operacional	Dockerfile	Desaprovação
al2	Runtime somente para sistema operacional	Amazon Linux 2	Dockerfile para runtime somente para sistema operacional no GitHub	

Galeria pública do Amazon Elastic Container Registry: gallery.ecr.aws/lambda/provided

Usar uma imagem base que não é da AWS

O Lambda é compatível com qualquer imagem que esteja em conformidade com um dos seguintes formatos de manifesto de imagem:

- Esquema 2 de manifesto V2 de imagem de Docker (usado com o Docker versão 1.10 e posteriores)
- Especificações de Open Container Initiative (OCI – Iniciativa de contêiner aberto) (v1.0.0 e posteriores)

O Lambda oferece suporte a um tamanho máximo de imagem descompactada de 10 GB, incluindo todas as camadas.

> **Note**
>
> Para tornar a imagem compatível com o Lambda, você deve incluir um cliente de interface de runtime para sua linguagem na imagem.

Clientes de interface de runtime

Se você usar uma imagem base somente para sistema operacional ou uma imagem base alternativa, deverá incluir um cliente de interface de runtime na imagem. O cliente de interface de runtime deve estender o API de tempo de execução do Lambda, que gerencia a interação entre o Lambda e o código da sua função. A AWS fornece clientes de interface de runtime de código aberto para as seguintes linguagens:

- Node.js
- Python
- Java
- .NET
- Go
- Ruby
- Rust o cliente de runtime do Rust é um pacote experimental. Ele está sujeito a alterações, e é destinado apenas a fins de avaliação.

Caso esteja usando uma linguagem que não tenha um cliente de interface de runtime fornecido pela AWS, você deverá criar seu próprio cliente de interface de runtime.

Permissões do Amazon ECR

Antes de criar uma função do Lambda com base em uma imagem de contêiner, você deve criar a imagem localmente e carregá-la no repositório do Amazon ECR. Quando criar a função, especifique o URI do repositório do Amazon ECR.

Certifique-se de que as permissões para o usuário ou o perfil que cria a função incluam `GetRepositoryPolicy` e `SetRepositoryPolicy`.

Por exemplo, use o console do IAM para criar uma função com a seguinte política:

```
{
  "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"
    }
  ]
}
```

Políticas do repositório do Amazon ECR

Para uma função na mesma conta que a imagem de contêiner no Amazon ECR, você pode adicionar as permissões `ecr:BatchGetImage` e `ecr:GetDownloadUrlForLayer` à política do repositório do Amazon ECR. O exemplo a seguir mostra a política mínima:

```
{
        "Sid": "LambdaECRImageRetrievalPolicy",
        "Effect": "Allow",
        "Principal": {
          "Service": "lambda.amazonaws.com"
        },
        "Action": [
          "ecr:BatchGetImage",
          "ecr:GetDownloadUrlForLayer"
        ]
    }
```

Para obter mais informações sobre as permissões de repositório do Amazon ECR, consulte Políticas de repositório privado no Guia do usuário do Amazon Elastic Container Registry.

Se o repositório do Amazon ECR não incluir essas permissões, o Lambda adicionará`ecr:BatchGetImage`e`ecr:GetDownloadUrlForLayer`para as permissões do repositório de imagens do contêiner. O Lambda só poderá adicionar essas permissões se a entidade principal que chamar o Lambda tiver as permissões `ecr:getRepositoryPolicy` e `ecr:setRepositoryPolicy`.

Para visualizar ou editar suas permissões de repositório do Amazon ECR, siga as instruções em Configurar uma instrução de política de repositório no Guia do usuário do Amazon Elastic Container Registry.

Permissões entre contas do Amazon ECR

Uma conta diferente na mesma região pode criar uma função que usa uma imagem de contêiner de propriedade da sua conta. No exemplo a seguir, a política de permissões de repositório do Amazon ECR precisa das instruções apresentadas a seguir para conceder acesso à conta número 123456789012.

- CrossAccountPermission: permite que a conta 123456789012 crie e atualize funções do Lambda que usam imagens desse repositório ECR.

- LambdaecrimagecrossAccountRetrievalPolicy: o Lambda eventualmente definirá o estado de uma função como inativo se ela não for invocada por um período prolongado. Essa declaração é necessária para que o Lambda possa recuperar a imagem do contêiner para otimização e armazenamento em cache em nome da função pertencente a 123456789012.

Example — Adicionar permissão entre contas ao repositório

```
{
  "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:*"
        }
      }
    }
  ]
}
```

Para dar acesso a várias contas, você adiciona os IDs de conta à lista Principal na política `CrossAccountPermission` e à lista Avaliação da condição em `LambdaECRImageCrossAccountRetrievalPolicy`.

Se você estiver trabalhando com várias contas em uma organização da AWS, recomendamos enumerar cada ID de conta na política de permissões do ECR. Essa abordagem se alinha com a prática recomendada de segurança da AWS de definir permissões estreitas nas políticas do IAM.

Além das permissões do Lambda, o usuário ou o perfil que cria a função também deve ter as permissões `BatchGetImage` e `GetDownloadUrlForLayer`.

Ciclo de vida da função

Depois de carregar uma imagem de contêiner nova ou atualizada, o Lambda otimiza a imagem antes que a função possa processar chamadas. O processo de otimização pode levar alguns segundos. A função permanece no estado `Pending` até que o processo seja concluído. A função então faz a transição para o estado `Active`. Enquanto o estado for `Pending`, você poderá invocar a função, mas ocorrerá uma falha em outras operações na função. As invocações que ocorrem enquanto uma atualização de imagem está em andamento executam o código da imagem anterior.

Se uma função não é invocada por várias semanas, o Lambda recupera a versão otimizada e a função muda para o estado `Inactive`. Para reativar a função, você deve chamá-la. O Lambda rejeita a primeira invocação e a função entra no`Pending`até que o Lambda reotimize a imagem. A função então retorna ao estado `Active`.

O Lambda busca periodicamente a imagem de contêiner associada a partir do repositório do Amazon ECR. Se a imagem correspondente do contêiner não existir mais no Amazon ECR ou as permissões forem revogadas, a função irá inserir o estado `Failed` e o Lambda retornará uma falha para qualquer chamada de função.

É possível usar a API do Lambda para obter informações sobre o estado de uma função. Para ter mais informações, consulte Estados da função do Lambda.

Compreender os métodos de invocação de funções do Lambda

Após implantar sua função do Lambda, você poderá invocá-la de várias maneiras:

- O console do Lambda: use o console do Lambda para criar rapidamente um evento de teste para invocar sua função.
- O AWS SDK: use o AWS SDK para invocar programaticamente sua função.
- A API Invoke: use a API Invoke do Lambda para invocar diretamente sua função.
- A AWS Command Line Interface (AWS CLI): use o comando `aws lambda invoke` da AWS CLI para invocar diretamente sua função diretamente da linha de comando.
- Um endpoint HTTP(S) de URL da função: use URLs de função para criar um endpoint HTTP(S) dedicado que você pode usar para invocar a função.

Todos esses métodos são formas diretas de invocar sua função. No Lambda, um caso de uso comum é invocar sua função com base em um evento que ocorre em outro lugar da sua aplicação. Alguns serviços podem invocar uma função do Lambda a cada novo evento. Isso é chamado de acionador. Para serviços baseados em fluxos e filas, o Lambda invoca a função com lotes de registros. Isso é chamado de mapeamento de origem de eventos.

Quando você invocar uma função, poderá optar por invocá-la de forma síncrona ou assíncrona. Com a invocação síncrona, você aguarda a função processar o evento e retornar uma resposta. Com a invocação assíncrona, o Lambda coloca o evento na fila para processamento e retorna uma resposta imediatamente. O parâmetro de solicitação `InvocationType` na API Invoke determina como o Lambda invocará sua função. Um valor de `RequestResponse` indica invocação síncrona e um valor de `Event` indica invocação assíncrona.

Se a invocação da função resultar em um erro, para invocações síncronas, visualize a mensagem de erro na resposta e repita a invocação manualmente. Para invocações assíncronas, o Lambda processa novas tentativas automaticamente e pode enviar registros de invocação para um destino.

Invocação síncrona

Quando você invoca uma função de forma síncrona, o Lambda executa a função e aguarda uma resposta. Quando a função é concluída, o Lambda retorna a resposta do código da função com dados adicionais, como a versão da função que foi invocada. Para invocar uma função de forma síncrona com o AWS CLI, use o comando `invoke`.

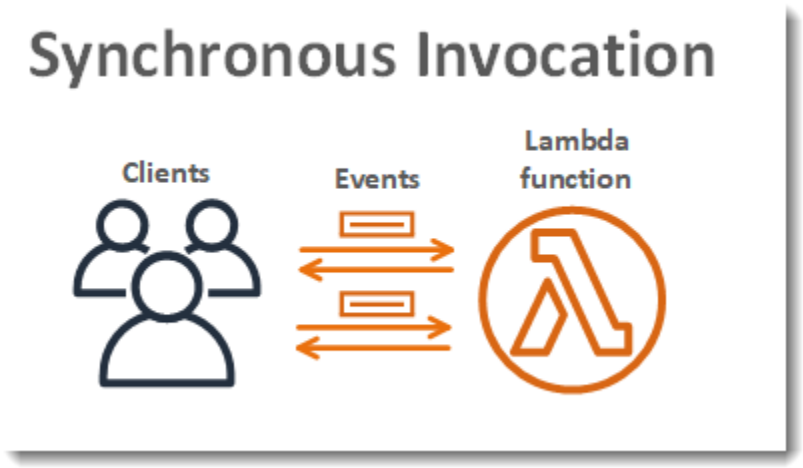
```
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{ "key": "value" }' response.json
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

A seguinte saída deverá ser mostrada:

```
{
    "ExecutedVersion": "$LATEST",
    "StatusCode": 200
}
```

O diagrama a seguir mostra clientes invocando uma função do Lambda do de forma síncrona. O Lambda envia os eventos diretamente para a função e envia a resposta da função de volta para o invocador.



O `payload` é uma string que contém um evento no formato JSON. O nome do arquivo em que o AWS CLI grava a resposta da função é `response.json`. Se a função retornar um objeto ou erro,

o corpo da resposta será o objeto ou erro no formato JSON. Se a função sair sem erro, o corpo da resposta seránull.

> **Note**
>
> O Lambda não espera a conclusão das extensões externas para enviar a resposta. As extensões externas são executadas como processos independentes no ambiente de execução e continuam em execução após a conclusão da invocação da função. Para ter mais informações, consulte Ampliar funções do Lambda usando extensões do Lambda.

A saída do comando, que é exibida no terminal, inclui informações de cabeçalhos na resposta do Lambda. Isso inclui a versão que processou o evento (útil quando você usa aliases) e o código de status retornado pelo Lambda. Se o Lambda foi capaz de executar a função, o código de status é 200, mesmo que a função tenha retornado um erro.

> **Note**
>
> Para funções com um longo tempo limite, o cliente pode ser desconectado durante a invocação síncrona enquanto aguarda por uma resposta. Configure seu cliente HTTP, SDK, firewall, proxy ou sistema operacional para permitir conexões longas com tempo limite ou configurações de ativação.

Se o Lambda não for capaz de executar a função, o erro será exibido na saída.

```
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload value response.json
```

A seguinte saída deverá ser mostrada:

```
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]
```

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- AWS CLI – Configuração rápida com `aws configure`

Você pode usar a AWS CLI para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

A seguinte saída deverá ser mostrada:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `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
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

A seguinte saída deverá ser mostrada:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e Ubuntu no Windows. Os usuários do macOS precisam usar `base64 -D`.

Para obter mais informações sobre a API `Invoke`, incluindo uma lista completa de parâmetros, cabeçalhos e erros, consulte Invoke.

Quando invocar uma função diretamente, você poderá verificar a resposta quanto a erros e tentar novamente. A AWS CLI e o AWS SDK também realizam automaticamente novas tentativas em erros de serviço, controle de utilização e tempo limite do cliente. Para ter mais informações, consulte Compreender o comportamento de novas tentativas no Lambda.

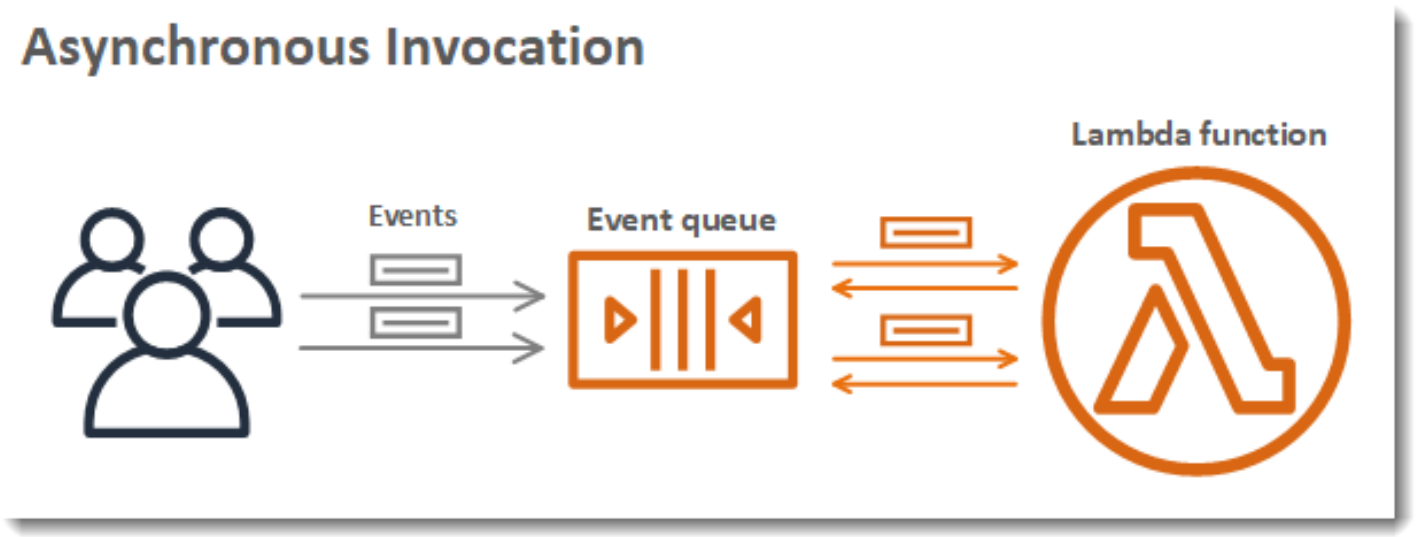
Invocação assíncrona

Vários Serviços da AWS, como o Amazon Simple Storage Service (Amazon S3) e o Amazon Simple Notification Service (Amazon SNS), invocam funções de forma assíncrona para processar eventos. Ao invocar uma função de forma assíncrona, não aguarde uma resposta do código da função. Você entrega o evento para o Lambda e ele cuida do resto. Você pode configurar como o Lambda processa os erros e pode enviar registros de invocação para um recurso downstream, como o Amazon Simple Queue Service (Amazon SQS) ou o Amazon EventBridge (EventBridge), para encadear componentes da aplicação.

Seções

Como o Lambda trabalha com invocações assíncronas

O diagrama a seguir mostra clientes invocando uma função do Lambda de forma assíncrona. O Lambda coloca os eventos em fila antes de enviá-los para a função.



Para invocação de forma assíncrona, o Lambda coloca o evento em uma fila e retorna uma resposta bem-sucedida sem informações adicionais. Um processo separado lê os eventos na fila e os envia

para sua função. Para invocar uma função de maneira assíncrona, defina o parâmetro do tipo de invocação como `Event`.

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

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

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

O arquivo de saída (`response.json`) não contém informações, mas continuará sendo criado quando esse comando for executado. Se o Lambda não conseguir adicionar o evento a uma fila, a mensagem de erro será exibida na saída do comando.

O Lambda gerencia a fila de eventos assíncronos da função e realiza novas tentativas em casos de erro. Se a função retornar um erro, o Lambda tentará executá-la mais duas vezes, com um intervalo de um minuto entre a primeira e a segunda tentativa, e um intervalo de dois minutos entre a segunda e a terceira. Os erros de função incluem erros retornados pelo código e pelo runtime da função, como o tempo ser atingido.

Se a função não tiver simultaneidade suficiente disponível para processar todos os eventos, as solicitações adicionais serão limitadas. Para erros de controle de utilização (429) e de sistema (série 500), o Lambda retorna o evento para a fila e tenta executar a função novamente por até 6 horas. O intervalo de repetição aumenta exponencialmente de 1 segundo após a primeira tentativa para no máximo 5 minutos. Se a fila contém muitas entradas, o Lambda aumenta o intervalo de repetição e reduz a taxa em que lê eventos da fila.

Mesmo que a função não retorne um erro, é possível que ela receba o mesmo evento do Lambda várias vezes, porque a própria fila se tornará consistente. Se a função não conseguir acompanhar os eventos recebidos, é possível que eventos sejam excluídos da fila sem serem enviados para a

função. Certifique-se de que seu código de função lide corretamente com eventos duplicados, e de que você tenhas simultaneidade suficiente disponível para lidar com todas as invocações.

Quando a fila é muito longa, novos eventos podem expirar antes de o Lambda ter a chance de enviá-los para sua função. Quando há falha em todas as tentativas de processamento de um evento ou ele expira, o Lambda o descarta. É possível configurar o tratamento de erros para uma função a fim de reduzir o número de tentativas que o Lambda executa ou descartar eventos não processados mais rapidamente.

Também é possível configurar o Lambda para enviar um registro de invocação para outro serviço. O Lambda suporta os destinos a seguir para invocação assíncrona. Observe que as filas do SQS FIFO e os tópicos do SNS FIFO não são compatíveis.

- Amazon SQS— Uma fila SQS padrão.
- Amazon SNS: um padrão do tópico SNS.
- AWS Lambda: uma função do Lambda.
- Amazon EventBridge: u barramento de eventos do EventBridge.

O registro de invocação contém detalhes sobre a solicitação e a resposta no formato JSON. É possível configurar destinos separados para eventos que são processados com êxito e eventos em que há falha no processamento de todas as tentativas. Como alternativa, é possível configurar uma fila padrão do Amazon SQS ou um tópico padrão do Amazon SNS como uma fila de mensagens não entregues para os eventos descartados. Para filas de mensagens mortas, o Lambda envia somente o conteúdo do evento, sem detalhes sobre a resposta.

Se o Lambda não conseguir enviar um registro para um destino que você configurou, ele enviará uma métrica `DestinationDeliveryFailures` para o Amazon CloudWatch. Isso poderá acontecer se sua configuração incluir um tipo de destino incompatível, como uma fila do Amazon SQS FIFO ou um tópico do Amazon SNS FIFO. Os erros de entrega também podem ocorrer devido a erros de permissão e limites de tamanho. Para obter mais informações sobre as métricas de invocação do Lambda, consulte Métricas de invocação.

> **Note**
>
> Para evitar que uma função seja acionada, você pode definir a simultaneidade reservada da função como zero. Quando você define a simultaneidade reservada como zero para uma função invocada de forma assíncrona, o Lambda começa a enviar novos eventos

para a fila de mensagens não entregues configurada ou para o destino do evento em caso de falha, sem novas tentativas. Para processar eventos que foram enviados enquanto a simultaneidade reservada estava definida como zero, você precisa consumir os eventos da fila de mensagens não entregues ou do destino do evento em caso de falha.

Configurar o tratamento de erros para invocação assíncrona

Use o console do Lambda para definir as configurações de tratamento de erros em uma função, uma versão ou um alias.

Como configurar o tratamento de erros

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e, em seguida, Asynchronous invocation (Invocação assíncrona).
4. Em Asynchronous invocation (Invocação assíncrona), escolha Edit (Editar).
5. Configure as definições a seguir.
   - Maximum age of event (Idade máxima do evento): a quantidade máxima de tempo que o Lambda retém um evento na fila de eventos assíncronos, até 6 horas.
   - Retry attempts (Tentativas de repetição): o número de vezes que o Lambda tenta novamente quando a função retorna um erro, entre 0 e 2.
6. Escolha Salvar.

Quando um evento de invocação exceder a idade máxima ou falhar em todas as tentativas de repetição, o Lambda o descartará. Para reter uma cópia de eventos descartados, configure um destino de evento com falha.

Configurar destinos para invocação assíncrona

Para reter registros de invocações assíncronas, adicione um destino à função. Você pode optar por enviar invocações com êxito ou com falha para um destino. Cada função pode ter vários destinos para que você possa configurar destinos separados para eventos com êxito e com falha. Cada registro enviado ao destino é um documento JSON com detalhes sobre a invocação. Da mesma

forma que as configurações de tratamento de erros, é possível configurar destinos em uma função, em uma versão de função ou em um alias.

> **Note**
>
> Você também pode reter registros de invocações com falha para os seguintes tipos de mapeamento de origem de eventos: Amazon Kinesis, Amazon DynamoDB, Apache Kafka autogerenciado e Amazon MSK.

A tabela a seguir lista os destinos aceitos para registros de invocação assíncrona. Para que o Lambda envie registros com êxito ao destino escolhido, certifique-se de que o perfil de execução da função também contenha as permissões relevantes. A tabela também descreve como cada tipo de destino recebe o registro de invocação JSON.

Tipos de destino	Permissão obrigatória	Formato JSON específico para o destino
Fila do Amazon SQS	sqs:SendMessage	O Lambda passa o registro de invocação como `Message` para o destino.
Tópico do Amazon SNS	sns:Publish	O Lambda passa o registro de invocação como `Message` para o destino.
Função do Lambda	InvokeFunction	O Lambda passa o registro de invocação como a carga útil para a função.
EventBridge	events:PutEvents	• O Lambda passa o registro de invocação como `detail` na chamada PutEvents. • O valor para o campo do evento `source` é `lambda`. • O valor para o campo de evento `detail-type` é “Lambda Function Invocatio

Tipos de destino	Permissão obrigatória	Formato JSON específico para o destino
		n Result - Success” ou “Lambda Function Invocatio n Result - Failure”. • O campo do evento `resource` contém nomes do recurso da Amazon (ARN) da função e do destino. • Para outros campos de eventos, consulte os eventos do Amazon EventBridge.

O exemplo a seguir mostra um registro de invocação para um evento que teve três falhas de tentativa de processamento devido a um erro de função. O registro de invocação contém detalhes sobre o evento, a resposta e o motivo pelo qual o registro foi enviado.

```
{
    "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"
    }
}
```

As etapas a seguir descrevem como configurar um destino para uma função usando o console do Lambda.

Configurar um destino para registros de invocação assíncrona

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Em Function overview (Visão geral da função), escolha Add destination (Adicionar destino).
4. Em Source (Origem), escolha Asynchronous invocation (Invocação assíncrona).
5. Em Condition (Condição), escolha uma das seguintes opções:
   - On failure (Em caso de falha): envie um registro quando o evento falhar em todas as tentativas de processamento ou exceder a idade máxima.
   - On success (Em caso de êxito): envie um registro quando a função processa com êxito uma invocação assíncrona.
6. Em Destination type (Tipo de destino), escolha o tipo de recurso que recebe o registro da invocação.
7. Em Destination (Destino), escolha um recurso.
8. Escolha Salvar.

Quando uma invocação corresponde à condição, o Lambda envia um documento JSON com detalhes sobre a invocação para o destino.

Formato JSON específico para o destino

- No Amazon SQS e no Amazon SNS (`SnsDestination` e `SqsDestination`), o registro de invocação é passado como `Message` para o destino.

- No Lambda (`LambdaDestination`), o registro de invocação é passado como carga útil para a função.
- No EventBridge (`EventBridgeDestination`), o registro de invocação é passado como `detail`na chamada PutEvents. O valor para o campo do evento `source` é `lambda`. O valor para o campo de evento `detail-type` é Lambda Function Invocation Result – Success (Resultado da invocação da função Lambda: sucesso) ou Lambda Function Invocation Result – Failure (Resultado da invocação da função Lambda: falha). O campo do evento `resource` contém nomes do recurso da Amazon (ARN) da função e do destino. Para outros campos de eventos, consulte os eventos do Amazon EventBridge.

O exemplo a seguir mostra um registro de invocação para um evento que teve três falhas de tentativa de processamento devido a um erro de função.

Example registro de invocação

```
{
    "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"
    }
```

```
}
```

O registro de invocação contém detalhes sobre o evento, a resposta e o motivo pelo qual o registro foi enviado.

Rastreamento de solicitações até destinos

Você pode usar o AWS X-Ray para exibir uma visualização conectada de cada solicitação à medida que ela é adicionada à fila, processada por uma função do Lambda e passada para o serviço de destino. Quando você ativa o rastreamento com X-Ray para uma função ou um serviço que invoca uma função, o Lambda adiciona um cabeçalho X-Ray à solicitação e passa o cabeçalho para o serviço de destino. Os rastreamentos dos serviços de upstream são vinculados automaticamente aos rastreamentos de funções do Lambda e serviços de downstream, o que cria uma visão completa de toda a aplicação. Para obter mais informações sobre rastreamento, consulte Visualizar as invocações da função do Lambda usando o AWS X-Ray.

API de configuração de invocação assíncrona

Para gerenciar configurações de invocação assíncrona com a AWS CLI ou o AWS SDK, use as operações de API a seguir.

- PutFunctionEventInvokeConfig
- GetFunctionEventInvokeConfig
- UpdateFunctionEventInvokeConfig
- ListFunctionEventInvokeConfigs
- DeleteFunctionEventInvokeConfig

Para configurar a invocação assíncrona com a AWS CLI, use o comando `put-function-event-invoke-config`. O exemplo a seguir configura uma função com uma idade máxima de evento de 1 hora e nenhuma repetição.

```
aws lambda put-function-event-invoke-config --function-name error \
--maximum-event-age-in-seconds 3600 --maximum-retry-attempts 0
```

A seguinte saída deverá ser mostrada:

```
{
    "LastModified": 1573686021.479,
```

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

O comando `put-function-event-invoke-config` substitui qualquer configuração existente na função, versão ou alias. Para configurar uma opção sem redefinir outras, use `update-function-event-invoke-config`. O exemplo a seguir configura o Lambda para enviar um registro a uma fila padrão do SQS denominada `destination` quando um evento não puder ser processado.

```
aws lambda update-function-event-invoke-config --function-name error \
--destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-
east-2:123456789012:destination"}}'
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
        }
    }
}
```

Filas de mensagens mortas

Como alternativa a um destino em caso de falha, é possível configurar a função com uma fila de mensagens mortas para salvar eventos descartados para processamento adicional. Uma fila de mensagens mortas age da mesma forma que um destino em caso de falha na medida em que é usado quando há falha em todas as tentativas de processamento de um evento ou ele expira sem ser processado. No entanto, uma fila de mensagens mortas faz parte da configuração específica

da versão de uma função, portanto ela é bloqueada quando você publica uma versão. Destinos em caso de falha também oferecem suporte a destinos adicionais e incluem detalhes sobre a resposta da função no registro de invocação.

Para reprocessar eventos em uma fila de mensagens mortas, você pode defini-la como uma fonte do evento para a função do Lambda. Você também pode recuperar os eventos manualmente.

É possível escolher uma fila padrão do Amazon SQS ou um tópico padrão do Amazon SNS para a sua fila de mensagens não entregues. As filas FIFO e os tópicos do Amazon SNS FIFO são incompatíveis. Se você não tiver uma fila ou um tópico, crie um. Escolha o tipo de destino que corresponde ao seu caso de uso.

- Fila do Amazon SQS: uma fila suspende os eventos falhos até serem recuperados. Escolha uma fila padrão do Amazon SQS caso espere que uma única entidade, como uma função do Lambda ou um alarme do CloudWatch, processe o evento que falhou. Para ter mais informações, consulte Usar o Lambda com o Amazon SQS.

  Crie uma fila no console do Amazon SQS.
- Tópico do Amazon SNS: um tópico retransmite os eventos com falha para um ou mais destinos. Escolha um tópico padrão do Amazon SNS caso espere que várias entidades atuem em um evento com falha. Por exemplo, você pode configurar um tópico para enviar eventos a um endereço de email, uma função do Lambda e/ou um endpoint HTTP. Para ter mais informações, consulte Invocar funções do Lambda com notificações do Amazon SNS.

  Crie um tópico do SNS usando o console do Amazon SNS.

Sua função precisa de permissões adicionais para enviar eventos a uma fila ou um tópico. Adicione uma política com as permissões necessárias à função de execução de sua função.

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

Se a fila ou o tópico de destino for criptografado com uma chave gerenciada pelo cliente, a função de execução também deverá ser um usuário na política baseada em recursos da chave.

Depois de criar o destino e atualizar a função de execução de sua função, adicione a fila de mensagens mortas à função. Você pode configurar várias funções para enviarem eventos ao mesmo destino.

Como configurar uma fila de mensagens mortas

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e, em seguida, Asynchronous invocation (Invocação assíncrona).
4. Em Asynchronous invocation (Invocação assíncrona), escolha Edit (Editar).
5. Definarecurso DLQ para Amazon SQS ou Amazon SNS.
6. Escolha a fila ou o tópico de destino.
7. Escolha Salvar.

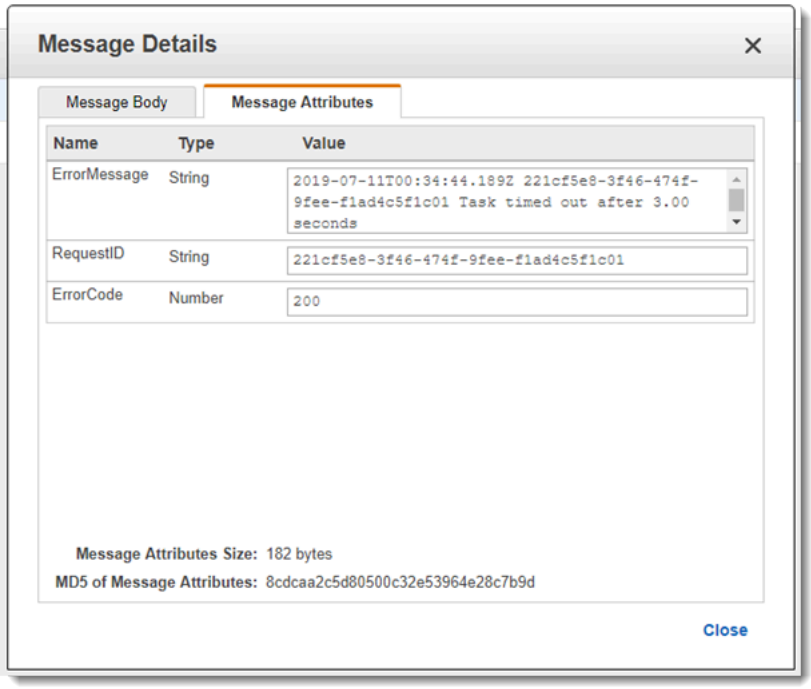
Para configurar uma fila de mensagens mortas com o AWS CLI, use o comando `update-function-configuration`.

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

O Lambda envia o evento para a fila de mensagens mortas no mesmo estado, mais informações adicionais nos atributos. Você pode usar essas informações para identificar o erro retornado pela função, ou correlacionar o evento com logs ou um rastreamento do AWS X-Ray.

Atributos de mensagens da fila de mensagens mortas

- RequestID (String): o ID de invocação da solicitação. Os IDs de solicitação aparecem nos logs de função. O X-Ray SDK também pode ser usado para registrar o ID de solicitação em um atributo no rastreamento. Em seguida, os rastreamentos podem ser procurados por ID de solicitação no console do X-Ray.
- ErrorCode (Número): o código de status HTTP.
- ErrorMessage (String): o primeiro 1 KB da mensagem de erro.



Se o Lambda não puder enviar uma mensagem para a fila de mensagens mortas, ele excluirá o evento e emitirá a métrica DeadLetterErrors. Isso pode acontecer por causa de falta de permissões, ou se o tamanho total da mensagem exceder o limite da fila ou do tópico de destino. Por exemplo, digamos que uma notificação do Amazon SNS com um tamanho de quase 256 KB acione uma função que resulte em erro. Nesse caso, os dados do evento incluídos pelo Amazon SNS, combinados aos atributos adicionados pelo Lambda, podem fazer com que a mensagem exceda o tamanho máximo permitido na fila de mensagens não entregues.

Se você está usando o Amazon SQS como uma fonte de eventos, configure uma fila de mensagens mortas na própria fila do Amazon SQS e não na função do Lambda. Para ter mais informações, consulte Usar o Lambda com o Amazon SQS.

Como o Lambda processa registros de origens de eventos baseadas em fluxos e filas

Um mapeamento de origem de evento é um recurso no Lambda que lê itens de serviços baseados em fluxo ou fila e invoca uma função com lotes de registros. Os serviços a seguir usam mapeamentos de origem de eventos para invocar funções do Lambda:

- Amazon DynamoDB
- Amazon Kinesis
- Amazon MQ
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Apache Kafka autogerenciado
- Amazon Simple Queue Service (Amazon SQS)
- Amazon DocumentDB (compatível com MongoDB) (Amazon DocumentDB)

> **Warning**
>
> Os mapeamentos da origem do evento do Lambda processam cada evento ao menos uma vez, podendo haver o processamento duplicado de registros. Para evitar possíveis problemas relacionados a eventos duplicados, é altamente recomendável tornar o código da função idempotente. Para saber mais, consulte Como tornar minha função do Lambda idempotente no Centro de Conhecimentos da AWS.

Como os mapeamentos de origem de eventos diferem dos acionadores diretos

Alguns serviços da AWS podem invocar diretamente as funções do Lambda usando acionadores. Esses serviços enviam eventos para o Lambda, e a função é invocada imediatamente quando o evento especificado ocorre. Os acionadores são adequados para eventos discretos e processamento em tempo real. Quando você cria um acionador usando o console do Lambda, o console interage com o serviço da AWS correspondente para configurar a notificação de eventos nesse serviço. Na verdade, o acionador é armazenado e gerenciado pelo serviço que gera os eventos, não pelo Lambda. Aqui estão alguns exemplos de serviços que usam acionadores para invocar funções do Lambda:

- Amazon Simple Storage Service (Amazon S3): invoca uma função quando um objeto é criado, excluído ou modificado em um bucket. Para ter mais informações, consulte Tutorial: Usar um acionador do Amazon S3 para invocar uma função do Lambda.
- Amazon Simple Notification Service (Amazon SNS): invoca uma função quando uma mensagem é publicada em um tópico do SNS. Para ter mais informações, consulte Como usar oAWS LambdaCom o Amazon Simple Notification Service.
- Amazon API Gateway: invoca uma função quando uma solicitação de API é feita para um endpoint específico. Para ter mais informações, consulte Invocar uma função do Lambda usando um endpoint do Amazon API Gateway.

Os mapeamentos de origem de eventos são recursos do Lambda criados e gerenciados dentro do serviço do Lambda. Os mapeamentos de origem de eventos são projetados para processar dados de streaming de alto volume ou mensagens de filas. O processamento de registros de um fluxo ou fila em lotes é mais eficiente do que processar registros individualmente.

Comportamento de lotes

Por padrão, um mapeamento de fonte de evento registra em lotes em uma única carga útil que o Lambda envia para sua função. Para ajustar o comportamento de lotes, configure uma janela de lotes (MaximumBatchingWindowInSeconds) e um tamanho de lote (BatchSize). A janela de lote é o tempo máximo para reunir registros em uma única carga útil. O tamanho de lote é o número máximo de registros em um único lote. O Lambda invoca a função quando um destes três critérios é atendido:

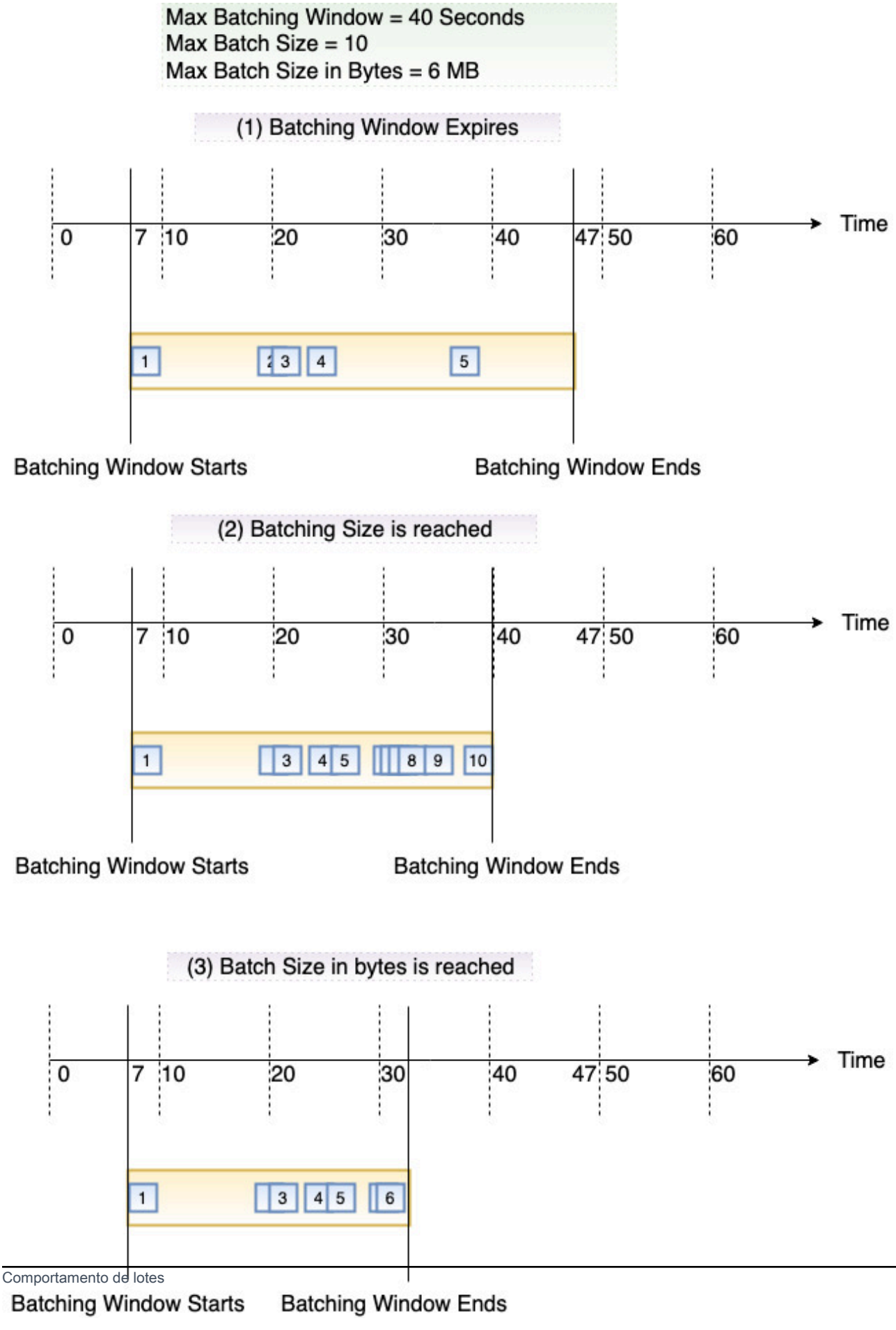
- A janela de lotes atinge o valor máximo. O comportamento padrão da janela de lote varia conforme a fonte de evento específica.
  - Para fontes de eventos do Kinesis, do DynamoDB e do Amazon SQS: a janela de lote padrão é 0 segundo. Isso significa que o Lambda envia lotes para sua função somente quando o tamanho do lote é atingido ou o limite de tamanho da carga útil é atingido. Para definir uma janela de lotes, configure `MaximumBatchingWindowInSeconds`. É possível configurar esse parâmetro para qualquer valor de 0 a 300 segundos em incrementos de 1 segundo. Se você configurar uma janela de lote, a próxima janela começará assim que a invocação de função anterior for concluída.
  - Para origens do evento do Amazon MSK, Apache Kafka autogerenciado, Amazon MQ e Amazon DocumentDB: a janela em lotes padrão é de 500 ms. É possível configurar

`MaximumBatchingWindowInSeconds` para qualquer valor de 0 a 300 segundos em incrementos de segundos. A janela de lotes começa assim que o primeiro registro chega.

> **Note**
>
> Como só é possível alterar `MaximumBatchingWindowInSeconds` em incrementos de segundos, você não pode reverter para a janela de lotes padrão de 500 ms após alterá-la. Para restaurar a janela de lotes padrão, é necessário criar um novo mapeamento de fonte de evento.

- O tamanho do lote é atendido. O tamanho mínimo do lote é 1. O tamanho do lote padrão e máximo dependem da fonte de eventos. Para obter detalhes sobre esses valores, consulte a especificação de BatchSize para a operação da API `CreateEventSourceMapping`.
- O tamanho da carga útil atinge 6 MB. Não é possível modificar esse limite.

O diagrama a seguir ilustra essas três condições. Suponha que uma janela de lotes comece em `t = 7` segundos. No primeiro caso, a janela de lotes atinge seu máximo de 40 segundos em `t = 47` segundos após acumular cinco registros. No segundo caso, como o tamanho do lote chega a dez antes que a janela de lotes expire, a janela de lotes é encerrada mais cedo. No terceiro caso, como o tamanho máximo da carga útil é atingido antes que a janela de lotes expire, a janela de lotes é encerrada mais cedo.


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
2
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 do mapeamento da fonte de eventos

Para gerenciar uma fonte de eventos com a AWS Command Line Interface (AWS CLI) ou um AWS SDK, use as seguintes operações de API:

- CreateEventSourceMapping
- ListEventSourceMappings
- GetEventSourceMapping
- UpdateEventSourceMapping
- DeleteEventSourceMapping

Usar o AWS Lambda com o Amazon DynamoDB

> **Note**
>
> Se você deseja enviar dados para um destino que não seja uma função do Lambda ou enriquecer os dados antes de enviá-los, consulte Amazon EventBridge Pipes (Pipes do Amazon EventBridge).

É possível usar uma função do AWS Lambda para processar registros em um Amazon DynamoDB Stream. Com o DynamoDB Streams, você pode acionar uma função do Lambda para executar o trabalho adicional cada vez que uma tabela do DynamoDB é atualizada.

O Lambda lê registros da transmissão e invoca sua função de maneira síncrona com um evento que contém registros de transmissão. O Lambda lê registros em lotes e invoca sua função para processar registros do lote.

Seções

- Evento de exemplo
- Fluxos de sondagem e agrupamento em lotes
- Posições iniciais de sondagem e fluxo
- Leitores simultâneos de um fragmento no DynamoDB Streams
- Permissões da função de execução
- Adicionar permissões e criar o mapeamento da origem do evento

- Tratamento de erros
- Métricas do Amazon CloudWatch
- Janelas de tempo
- Gerar relatórios de falhas de itens de lote
- Parâmetros de configuração de Fluxos do Amazon DynamoDB Streams
- Tutorial: Usar o AWS Lambda com o Amazon DynamoDB Streams
- Código de exemplo da função do
- Modelo do AWS SAM para uma aplicação do DynamoDB

Evento de exemplo

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"
    }
  ]}
```

Fluxos de sondagem e agrupamento em lotes

O Lambda sonda os fragmentos em sua transmissão do DynamoDB em busca de registros a uma taxa básica de 4 vezes por segundo. Quando os registros estão disponíveis, o Lambda invoca

a função e aguarda o resultado. Se o processamento for bem-sucedido, o Lambda continua a sondagem até que ela receba mais registros.

Por padrão, o Lambda invoca a função assim que os registros estão disponíveis. Se o lote que o Lambda lê da fonte de eventos tiver apenas um registro, o Lambda enviará apenas um registro à função. Para evitar a invocação da função com poucos registros, instrua a fonte de eventos para armazenar os registros em buffer por até cinco minutos, configurando uma janela de lotes. Antes de invocar a função, o Lambda continua a ler registros da fonte de eventos até coletar um lote inteiro, até que a janela de lote expire ou até que o lote atinja o limite de carga útil de 6 MB. Para ter mais informações, consulte Comportamento de lotes.

> **Warning**
>
> Os mapeamentos da origem do evento do Lambda processam cada evento ao menos uma vez, podendo haver o processamento duplicado de registros. Para evitar possíveis problemas relacionados a eventos duplicados, é altamente recomendável tornar o código da função idempotente. Para saber mais, consulte Como tornar minha função do Lambda idempotente no Centro de Conhecimentos da AWS.

Configure a opção ParallelizationFactor para processar um fragmento de um fluxo de dados do do DynamoDB com mais de uma invocação do Lambda simultaneamente. Você pode especificar o número de lotes simultâneos que o Lambda pesquisa de um fragmento por meio de um fator de paralelização de 1 (padrão) a 10. Quando você aumentar o número de lotes simultâneos por fragmento, o Lambda ainda garantirá o processamento por ordem no nível de item (partição e chave de classificação).

Posições iniciais de sondagem e fluxo

Esteja ciente de que a sondagem do fluxo durante a criação e as atualizações do mapeamento da origem do evento é, finalmente, consistente.

- Durante a criação do mapeamento da origem do evento, pode levar alguns minutos para a sondagem de eventos do fluxo iniciar.
- Durante as atualizações do mapeamento da origem do evento, pode levar alguns minutos para interromper e reiniciar a sondagem de eventos do fluxo.

Esse comportamento significa que, se você especificar `LATEST` como posição inicial do fluxo, o mapeamento da origem do evento pode perder eventos durante a criação ou as atualizações. Para garantir que nenhum evento seja perdido, especifique a posição inicial do fluxo como `TRIM_HORIZON`.

Leitores simultâneos de um fragmento no DynamoDB Streams

Para tabelas de região única que não são tabelas globais, você pode projetar até duas funções do Lambda para ler o mesmo fragmento do DynamoDB Streams ao mesmo tempo. Exceder esse limite pode resultar em controle de utilização de solicitação. Para tabelas globais, recomendamos que você limite o número de leitores simultâneos para 1 para evitar o controle de utilização de solicitações.

Permissões da função de execução

A política AWSLambdaDynamoDBExecutionRole gerenciada pela AWS inclui as permissões necessárias para o Lambda ler da sua fila do DynamoDB. Adicione essa política gerenciada ao perfil de execução da função.

Para enviar registros de lotes com falha para uma fila padrão do SQS ou um tópico padrão do SNS, sua função precisa de permissões adicionais. Cada serviço de destino requer uma permissão diferente, como se segue:

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

Adicionar permissões e criar o mapeamento da origem do evento

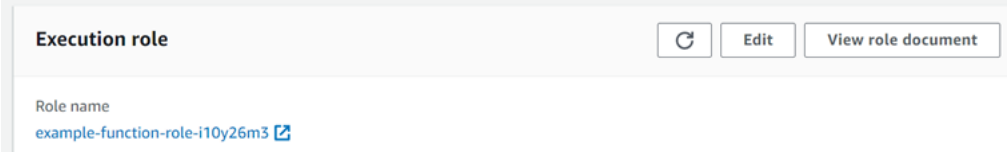
Crie um mapeamento de fontes de eventos para orientar o Lambda a enviar registros de sua transmissão para uma função do Lambda. É possível criar vários mapeamentos de origem de evento para processar os mesmos dados com várias funções do Lambda ou processar itens de vários fluxos com uma única função.

Para configurar sua função para ler do DynamoDB Streams, anexe a política AWSLambdaDynamoDBExecutionRole gerenciada pela AWS ao seu perfil de execução e, em seguida, crie um gatilho DynamoDB.

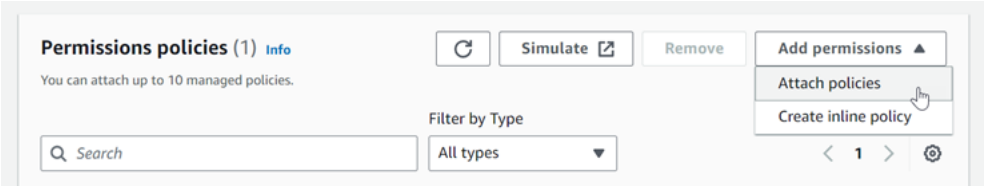
Para adicionar permissões e criar um acionador

1. Abra a página Funções do console do Lambda.

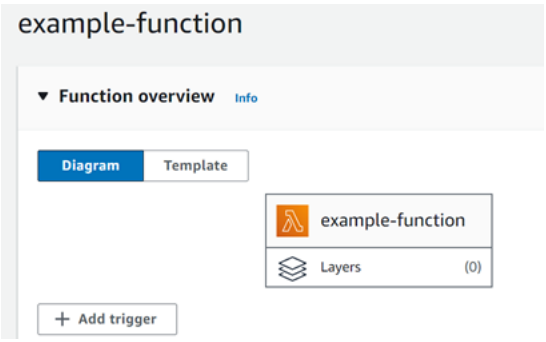
2. Escolha o nome de uma função.
3. Escolha a guia Configuration (Configuração) e, depois, Permissions (Permissões).
4. Em Nome do perfil, escolha o link para seu perfil de execução. Esse link abre o perfil no console do IAM.



5. Escolha Adicionar permissões e depois Anexar políticas.



6. No campo de pesquisa, digite `AWSLambdaDynamoDBExecutionRole`. Adicione esta política ao seu perfil de execução Essa é uma política gerenciada pela AWS que contém as permissões de que a função precisa para ler um fluxo do DynamoDB. Para obter mais informações sobre essa política, consulte AWSLambdaDynamoDBExecutionRole na Referência de política gerenciada da AWS.
7. Volte para a sua função no console do Lambda Em Visão geral da função, escolha Adicionar gatilho.



8. Escolha um tipo de acionador.
9. Configure as opções necessárias e escolha Add (Adicionar).

O Lambda é compatível com as seguintes opções de origens de eventos do DynamoDB:

Opções de fonte do evento

- DynamoDB table (Tabela do DynamoDB): a tabela do DynamoDB da qual os registros serão lidos.

- Batch size (Tamanho do lote): o número de registros a serem enviados para a função em cada lote, até 10.000. O Lambda transmite todos os registros no batch para a função em uma única chamada, enquanto o tamanho total dos eventos não exceder o limite de carga útil para invocação síncrona (6 MB).
- Batch window (Janela de lote): especifique o máximo de tempo para reunir registros antes de invocar a função, em segundos.
- Starting position (Posição inicial): processe apenas registros novos ou todos os registros existentes.
  - Latest (Mais recente): processe novos registros adicionados ao fluxo.
  - Trim horizon (Redução horizontal): processe todos os registros na transmissão.

  Depois de processar todos os registros existentes, a função é capturada e continua a processar novos registros.
- Destino na falha: uma fila padrão do SQS ou um tópico padrão do SNS para registros que não podem ser processados. Quando o Lambda descarta um lote de registros que é muito antigo ou esgotou todas as tentativas, ele envia detalhes sobre o lote à fila ou ao tópico.
- Retry attempts (Tentativas de repetição): o número máximo de vezes que o Lambda tenta novamente quando a função retorna um erro. Isso não se aplica a erros de serviço ou controles em que o lote não atingiu a função.
- Idade máxima do registro: a idade máxima de um registro que o Lambda envia para sua função.
- Dividir lote por erro: quando a função retornar um erro, o lote é dividido em dois antes de uma nova tentativa. A configuração do tamanho do lote original permanece inalterada.
- Concurrent batches per shard (Lotes simultâneos por fragmento): processa simultaneamente vários lotes do mesmo fragmento.
- Enabled (Habilitado): defina como verdadeiro para habilitar o mapeamento de fontes de eventos. Defina como falso para interromper o processamento de registros. O Lambda monitora o último registro processado e retoma o processamento a partir desse ponto quando o mapeamento é habilitado novamente.

> **Note**
>
> Você não é cobrado por chamadas da API GetRecords invocadas pelo Lambda como parte de acionadores do DynamoDB.

Para gerenciar a configuração da fonte do evento posteriormente, escolha o gatilho no designer.

Tratamento de erros

O tratamento de erros para mapeamentos de origem de eventos do DynamoDB depende se o erro ocorre antes de a função ser invocada ou durante a invocação da função:

- Antes da invocação: se um mapeamento de origem de eventos do Lambda não conseguir invocar a função devido a limitações ou a outros problemas, ele tentará novamente até que os registros expirem ou excedam a idade máxima configurada no mapeamento de origem de eventos (MaximumRecordAgeInSeconds).
- Durante a invocação: se a função for invocada, mas retornar um erro, o Lambda tentará novamente até que os registros expirem, excedam a idade máxima (MaximumRecordAgeInSeconds) ou atinjam a cota de repetição configurada (MaximumRetryAttempts). Para erros de função, também é possível pode configurar BisectBatchOnFunctionError, que divide um lote com falha em dois lotes em lotes, isolando registros com problema e evitando exceder tempos limites. A divisão de lotes não consome a cota de repetição.

Se as medidas de tratamento de erros falharem, o Lambda descartará os registros e continuará processando lotes provenientes da transmissão. Com as configurações padrão, isso significa que um registro inválido pode bloquear o processamento no fragmento afetado por até um dia. Para evitar isso, configure o mapeamento de fontes de eventos da sua função com um número razoável de tentativas e uma idade máxima de registro que se adapte ao seu caso de uso.

Configurar destinos para invocações com falha

Para reter registros de invocações de mapeamento da origem do evento com falha, adicione um destino ao mapeamento da origem de eventos da função. Cada registro enviado ao destino é um documento JSON com metadados sobre a invocação que falhou. É possível configurar qualquer tópico do Amazon SNS ou fila do Amazon SQS como destino. Sua função de execução deve ter permissões para o destino:

- Para destinos SQS: sqs:SendMessage
- Para destinos SNS: sns:Publish

Para configurar um destino em caso de falha usando o console, siga estas etapas:

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Em Function overview (Visão geral da função), escolha Add destination (Adicionar destino).
4. Em Origem, escolha Invocação do mapeamento da origem do evento.
5. Em Mapeamento da origem do evento, escolha uma origem de eventos configurada para essa função.
6. Em Condição, selecione Em caso de falha. Para invocações de mapeamento da origem de eventos, essa é a única condição aceita.
7. Em Tipo de destino, escolha o tipo de destino para o qual o Lambda envia registros de invocação.
8. Em Destination (Destino), escolha um recurso.
9. Escolha Salvar.

Também é possível configurar um destino em caso de falha usando a AWS Command Line Interface (AWS CLI). Por exemplo, o seguinte comando create-event-source-mapping adiciona um mapeamento de origem de eventos com um destino SQS em caso de falha 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"}}'
```

O comando update-event-source-mapping a seguir atualiza um mapeamento de origem de eventos para enviar registros de chamada com falha para um destino SNS após duas novas tentativas ou se os registros tiverem mais de uma hora.

```
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"}}'
```

As configurações atualizadas são aplicadas de forma assíncrona e não são refletidas na saída até que o processo seja concluído. Use o comando get-event-source-mapping para visualizar o status atual.

Para remover um destino, forneça uma string vazia como argumento para o parâmetro `destination-config`:

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

O exemplo a seguir mostra um registro de invocação de uma transmissão do DynamoDB.

Example Registro de invocação

```
{
    "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"
    }
}
```

Você pode usar essas informações para recuperar os registros afetados da transmissão para solução de problemas. Os registros reais não estão incluídos, portanto, você deve processar esses registros e recuperá-los da transmissão antes que eles expirem e sejam perdidos.

Métricas do Amazon CloudWatch

O Lambda emite a métrica `IteratorAge` quando a sua função termina de processar um lote de registros. A métrica indica a idade do último registro no lote quando o processamento foi concluído. Se a sua função estiver processando novos eventos, você poderá usar a idade do iterador para estimar a latência entre quando um registro é adicionado e quando a função o processa.

Uma tendência crescente na idade do iterador pode indicar problemas com sua função. Para ter mais informações, consulte Trabalhar com métricas de funções Lambda.

Janelas de tempo

As funções do Lambda podem executar aplicações de processamento contínuo de transmissões. Um stream representa dados não vinculados que fluem continuamente por meio de sua aplicação. Para analisar as informações dessa entrada de atualização contínua, você pode vincular os registros incluídos usando uma janela definida em termos de tempo.

As janelas de tumbling são janelas de tempo distintas que abrem e fecham em intervalos regulares. Por padrão, as invocações do Lambda são sem estado. Não é possível usá-las para processar dados ao longo de várias invocações contínuas sem um banco de dados externo. No entanto, com as janelas de tumbling, você pode manter seu estado em todas as invocações. Esse estado contém o resultado agregado das mensagens previamente processadas para a janela atual. Seu estado pode ter no máximo 1 MB por fragmento. Se exceder esse tamanho, o Lambda encerra a janela antes.

Cada registro de um fluxo pertence a uma janela específica. O Lambda processará cada registro pelo menos uma vez, mas não garantirá que cada registro seja processado apenas uma vez. Em casos raros, como tratamento de erros, alguns registros poderão ser processados mais de uma vez. Os registros são sempre processados em ordem na primeira vez. Se os registros forem processados mais de uma vez, poderão ser processados fora de ordem.

Agregação e processamento

Sua função gerenciada pelo usuário é chamada tanto para agregação quanto para processamento dos resultados finais dessa agregação. O Lambda agrega todos os registros recebidos na janela. Você pode receber esses registros em vários lotes, cada um como uma invocação separada.

Cada invocação recebe um estado. Assim, ao usar janelas de tumbling, sua resposta de função do Lambda deve conter uma propriedade de `state`. Se a resposta não contiver uma propriedade de `state`, o Lambda considerará esta uma invocação com falha. Para satisfazer essa condição, a função pode retornar um objeto do `TimeWindowEventResponse`, que tem a seguinte forma JSON:

Example Valores de **`TimeWindowEventResponse`**

```
{
    "state": {
        "1": 282,
        "2": 715
    },
    "batchItemFailures": []
}
```

> **Note**
>
> Para funções Java, recomendamos o uso de um `Map<String, String>` para representar o estado.

No final da janela, a sinalização `isFinalInvokeForWindow` é definida como `true` para indicar que esse é o estado final e que está pronto para processamento. Após o processamento, a janela é concluída e sua invocação final é concluída e, em seguida, o estado é descartado.

No final da janela, o Lambda usa o processamento final para ações sobre os resultados da agregação. Seu processamento final é invocado de forma síncrona. Após a invocação bem-sucedida, sua função define os pontos de verificação no número da sequência e o processamento de streams continua. Se a invocação não for bem-sucedida, sua função do Lambda suspenderá o processamento adicional até uma chamada bem-sucedida.

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
}
```

Configuração

Você pode configurar janelas em cascata ao criar ou atualizar um mapeamento de fonte de eventos. Para configurar uma janela em cascata, especifique a janela em segundos (TumblingWindowInSeconds). O comando de exemplo da AWS Command Line Interface (AWS CLI) a seguir cria um mapeamento de fonte de eventos em streaming com uma janela em cascata de 120 segundos. A função do Lambda definida para agregação e processamento é chamada de `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
```

O Lambda determina os limites da janela em cascata com base no horário em que os registros foram inseridos no stream. Todos os registros têm um carimbo de data/hora aproximado disponível que o Lambda usa para determinar os limites.

As agregações de janelas em cascata não são compatíveis com refragmentação. Quando o fragmento termina, o Lambda considera a janela como fechada e os fragmentos filhos iniciam suas próprias janelas em um novo estado.

As janelas em cascata são totalmente compatíveis com as políticas `maxRetryAttempts` e `maxRecordAge`.

Example Handler.py: agregação e processamento

A função do Python a seguir demonstra como agregar e, em seguida, processar seu estado final:

```
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}
```

Gerar relatórios de falhas de itens de lote

Ao consumir e processar dados de transmissão de uma fonte de eventos, o Lambda definirá checkpoints por padrão no número mais elevado na sequência de um lote somente quando o lote for um sucesso total. O Lambda trata todos os outros resultados como uma falha completa e tenta processar novamente o lote até o limite de novas tentativas. Para permitir sucessos parciais durante o processamento de lotes de um stream, ative `ReportBatchItemFailures`. Permitir sucessos parciais pode ajudar a reduzir o número de novas tentativas em um registro, embora não impeça totalmente a possibilidade de novas tentativas em um registro bem-sucedido.

Para ativar `ReportBatchItemFailures`, inclua o valor de enum **`ReportBatchItemFailures`** na lista FunctionResponseTypes. Essa lista indica quais tipos de resposta estão habilitados para sua função. Você pode configurar essa lista ao criar ou atualizar um mapeamento de origem de eventos.

Sintaxe do relatório

Ao configurar relatórios sobre falhas de itens de lote, a classe `StreamsEventResponse` é retornada com uma lista de falhas de itens de lote. É possível usar um objeto `StreamsEventResponse` para retornar o número sequencial do primeiro registro com falha no lote. Você também pode criar sua própria classe personalizada usando a sintaxe de resposta correta. A seguinte estrutura JSON mostra a sintaxe de resposta necessária:

```
{
  "batchItemFailures": [
        {
            "itemIdentifier": "<SequenceNumber>"
        }
    ]
}
```

> **Note**
>
> Se a matriz `batchItemFailures` contém vários itens, o Lambda usa o registro com o menor número de sequência como ponto de verificação. Em seguida, o Lambda repete todos os registros a partir desse ponto de verificação.

Condições de sucesso e falha

O Lambda trata um lote como um sucesso completo se você retornar qualquer um destes:

- Uma lista de `batchItemFailure` vazia
- Uma lista de `batchItemFailure` nula
- Uma `EventResponse` vazia
- Uma `EventResponse` nula

O Lambda trata um lote como uma falha absoluta se você retornar qualquer um dos seguintes:

- Uma string `itemIdentifier` vazia
- Uma `itemIdentifier` nula
- Um `itemIdentifier` com um nome de chave inválido

O Lambda faz novas tentativas após falhas com base na sua estratégia de repetição.

Dividir um lote

Se a invocação falhar e `BisectBatchOnFunctionError` estiver ativado, o lote será dividido independentemente da configuração de `ReportBatchItemFailures`.

Quando uma resposta de sucesso parcial do lote é recebida e tanto `BisectBatchOnFunctionError` quanto `ReportBatchItemFailures` estão ativados, o lote é dividido no número de sequência retornado e o Lambda tenta novamente apenas os registros restantes.

Veja alguns exemplos de código de função que retornam a lista de IDs de mensagens com falha no lote:

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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: [] };
};
```

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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 para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do DynamoDB com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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
}
```

Parâmetros de configuração de Fluxos do Amazon DynamoDB Streams

Todos os tipos de origem de evento Lambda compartilham o mesmoCreateEventSourceMappingeUpdateEventSourceMappingOperações de API do. No entanto, apenas alguns dos parâmetros se aplicam aos Fluxos do DynamoDB Streams.

Parâmetros da fonte de eventos que se aplicam aos Fluxos do DynamoDB Streams

Parâmetro	Obrigatório	Padrão	Observações
BatchSize	N	100	Máximo: 10.000.
BisectBatchOnFunctionError	N	false	
DestinationConfig	N		Fila padrão do Amazon SQS ou um destino de tópico padrão do Amazon SNS para registros descartados
Habilitado	N	verdadeiro	
EventSourceArn	Y		O ARN do fluxo de dados ou um consumidor de fluxo
FilterCriteria	N		Filtragem de eventos do Lambda
FunctionName	Y		
FunctionResponseTypes	N		Para permitir que sua função reporte falhas específicas em um lote, inclua

Parâmetro	Obrigatório	Padrão	Observações
			o valor `ReportBatchItemFailures` em `FunctionResponseTypes`. Para ter mais informações, consulte Gerar relatórios de falhas de itens de lote.
MaximumBatchingWindowInSeconds	N	0	
MaximumRecordAgeInSeconds	N	-1	-1 significa infinito: registros com falha serão repetidos até que o registro expire. O limite de retenção de dados para o DynamoDB Streams é de 24 horas. Mínimo: -1 Máximo: 604.800
MaximumRetryAttempts	N	-1	-1 significa infinito: registros com falha são repetidos até que o registro expire Mínimo: 0 Máximo: 10.000.
ParallelizationFactor	N	1	Máximo: 10

Parâmetro	Obrigatório	Padrão	Observações
StartingPosition	Y		TRIM_HORIZON
TumblingWindowinSeconds	N		Mínimo: 0 Máximo: 900

Tutorial: Usar o AWS Lambda com o Amazon DynamoDB Streams

Neste tutorial, você cria uma função do Lambda para consumir eventos do Amazon DynamoDB Streams.

Pré-requisitos

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em Criar uma função do Lambda com o console para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, a AWS Command Line Interface (AWS CLI) versão 2 será necessária. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

A seguinte saída deverá ser mostrada:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell.

> **Note**
>
> No Windows, alguns comandos da CLI do Bash que você costuma usar com o Lambda (como `zip`) não são compatíveis com os terminais integrados do sistema operacional. Para obter uma versão do Ubuntu com o Bash integrada no Windows, instale o Subsistema do Windows para Linux. Os exemplos de comandos da CLI neste guia usam a formatação

Linux. Os comandos que incluem documentos JSON em linha deverão ser reformatados se você estiver usando a CLI do Windows.

Criar a função de execução

Crie a função de execução que dá à sua função permissão para acessar recursos do AWS.

Para criar uma função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione Create role (Criar função).
3. Crie uma função com as propriedades a seguir.
   - Entidade confiável: Lambda.
   - Permissions (Permissões): AWSLambdaDynamoDBExecutionRole.
   - Role name (Nome da função): **`lambda-dynamodb-role`**.

O AWSLambdaDynamoDBExecutionRole tem as permissões necessárias para a função ler itens do DynamoDB e gravar logs no CloudWatch Logs.

Criar a função

Crie uma função do Lambda que processe seus eventos do DynamoDB. O código da função grava alguns dos dados de eventos de entrada no CloudWatch Logs.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do DynamoDB com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do DynamoDB com o Lambda usando 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)}`);
};
```

Consumir um evento do DynamoDB com o Lambda usando 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 para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do DynamoDB com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando 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(())

}
```

Para criar a função

1. Copie o código de amostra em um arquivo chamado `example.js`.
2. Crie um pacote de implantação.

```
zip function.zip example.js
```

3. Crie uma função do Lambda com o 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
```

Testar a função do Lambda

Nesta etapa, você invoca sua função do Lambda manualmente usando o comando `invoke` da CLI do AWS Lambda e o seguinte exemplo de evento do DynamoDB. Copie o seguinte para um arquivo chamado `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"
      }
   ]
}
```

Execute o seguinte comando `invoke`.

```
aws lambda invoke --function-name ProcessDynamoDBRecords \
    --cli-binary-format raw-in-base64-out \
    --payload file://input.txt outputfile.txt
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

A função retorna a string `message` no corpo da resposta.

Verifique a saída no arquivo `outputfile.txt`.

Criar uma tabela do DynamoDB com uma transmissão habilitada

Criar uma tabela do Amazon DynamoDB com uma transmissão habilitada.

Como criar uma tabela do DynamoDB

1. Abra o console do DynamoDB.
2. Escolha Create table.
3. Crie uma tabela com as configurações a seguir.
   - Table name (Nome da tabela): **`lambda-dynamodb-stream`**
   - Primary key (Chave primária): **`id`** (string)
4. Escolha Criar.

Como habilitar fluxos

1. Abra o console do DynamoDB.
2. Escolha Tables (Tabelas).
3. Escolha a tabela lambda-dynamodb-stream.
4. Em Exports and streams (Exportações e transmissões), escolha DynamoDB stream details (Detalhes da tranmissão do DynamoDB).
5. Selecione Ativar.
6. Em Tipo de exibição, escolha Somente atributos principais.
7. Escolha Ativar fluxo.

Anote o ARN do fluxo. Você precisará dele na próxima etapa quando for associar a transmissão à função do Lambda. Para obter mais informações sobre como habilitar fluxos, consulte Capturing table activity with DynamoDB Streams (Capturar atividades de tabelas com o DynamoDB Streams).

Adicionar uma fonte de eventos no AWS Lambda

Crie um mapeamento da fonte do evento no AWS Lambda. Este mapeamento de fontes de eventos associa a transmissão do DynamoDB à função do Lambda. Depois de criar esse mapeamento da fonte do evento, o AWS Lambda começa a sondar o fluxo.

Execute o comando AWS CLI a seguir da `create-event-source-mapping`. Depois que o comando for executado, anote o UUID. Você precisará deste UUID para se referir ao mapeamento da fonte do evento em todos os comandos, por exemplo, ao excluir o mapeamento da fonte do evento.

```
aws lambda create-event-source-mapping --function-name ProcessDynamoDBRecords \
    --batch-size 100 --starting-position LATEST --event-source DynamoDB-stream-arn
```

Isso cria um mapeamento entre a transmissão do DynamoDB especificado e a função do Lambda. Você pode associar uma transmissão do DynamoDB a várias funções do Lambda e associar a mesma função do Lambda a várias transmissões. No entanto, as funções do Lambda compartilharão o throughput de leitura para os fluxos dos quais compartilham.

Você pode obter a lista de mapeamentos de fontes de eventos executando o comando a seguir.

```
aws lambda list-event-source-mappings
```

Esta lista retorna todos os mapeamentos de fontes de eventos que você criou, e para cada mapeamento mostra o `LastProcessingResult`, entre outras coisas. Este campo será usado para fornecer uma mensagem informativa, se houver problemas. Valores como `No records processed` (indica que o AWS Lambda não iniciou a sondagem ou de que não há registros no fluxo) e `OK` (indica que o AWS Lambda foi bem-sucedido na leitura dos registros do fluxo e invocou a função do Lambda) indicam que não há problemas. Se houver problemas, você receberá uma mensagem de erro.

Se você tiver muitos mapeamentos da fonte do evento, use o parâmetro de nome da função para refinar os resultados.

```
aws lambda list-event-source-mappings --function-name ProcessDynamoDBRecords
```

Testar a configuração

Teste a experiência completa. À medida que você executa atualizações nas tabelas, o DynamoDB grava registros de eventos na transmissão. Quando o AWS Lambda sonda o stream, ele detecta novos registros no stream e executa a função do Lambda em seu nome, passando os eventos para a função.

1. No console do DynamoDB, adicione, atualize ou exclua itens da tabela. O DynamoDB grava registros dessas ações na transmissão.
2. O AWS Lambda sonda a transmissão e, ao detectar atualizações nela, invoca a função do Lambda passando os dados do evento que ele encontrar na transmissão.
3. A função é executada e cria logs no Amazon CloudWatch. Você pode verificar os logs relatados no console do Amazon CloudWatch.

Limpe os recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Para excluir a função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Escolha Excluir.
4. Insira o nome do perfil no campo de entrada de texto e escolha Delete (Excluir).

Para excluir uma tabela do DynamoDB

1. Abra a página Tables (Tabelas) no console do DynamoDB.
2. Selecione a tabela que você criou.
3. Escolha Excluir.
4. Digite **`delete`** na caixa de texto.
5. Selecione Delete table (Excluir tabela).

Código de exemplo da função do

O código de amostra está disponível para as seguintes linguagens.

Tópicos

- Node.js
- Java 11

- C#
- Python 3
- Go

Node.js

O exemplo a seguir processa mensagens do DynamoDB registra seu conteúdo.

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");
};
```

Cria um zip do código do exemplo para criar um pacote de implantação. Para obter instruções, consulte Implantar funções do Lambda em Node.js com arquivos .zip.

Java 11

O exemplo a seguir processa mensagens do DynamoDB e registra seu conteúdo. O `handleRequest` é o manipulador que o AWS Lambda invoca e fornece dados do evento. O manipulador usa a classe predefinida `DynamodbEvent`, que é definida na biblioteca `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 o manipulador obtém um retorno normal sem exceções, o Lambda considera que os lotes de entrada de registros foram processados com êxito e começa a ler novos registros no fluxo. Se o manipulador gera uma exceção, o Lambda considera que os lotes de entrada de registros não foram processados e invoca novamente a função com o mesmo lote de registros.

Dependências

- `aws-lambda-java-core`
- `aws-lambda-java-events`

Crie o código com as dependências da biblioteca do Lambda para criar um pacote de implantação. Para obter instruções, consulte Implantar funções do Lambda em Java com arquivos .zip ou JAR.

C#

O exemplo a seguir processa mensagens do DynamoDB e registra seu conteúdo. O `ProcessDynamoEvent` é o manipulador que o AWS Lambda invoca e fornece dados do evento. O manipulador usa a classe predefinida `DynamoDbEvent`, que é definida na biblioteca `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());
            }
        }
    }
}
```

Substitua o `Program.cs` em um projeto do .NET Core pelo exemplo acima. Para obter instruções, consulte Criar e implantar funções do Lambda em C# com arquivos .zip.

Python 3

O exemplo a seguir processa mensagens do DynamoDB registra seu conteúdo.

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'])))
```

Cria um zip do código do exemplo para criar um pacote de implantação. Para obter instruções, consulte Trabalhar com arquivos .zip para funções do Lambda em Python.

Go

O exemplo a seguir processa mensagens do DynamoDB registra seu conteúdo.

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())
            }
        }
    }
```

```
}
```

Crie o executável com o `go build` e crie um pacote de implantação. Para obter instruções, consulte Implantar funções do Lambda em Go com arquivos .zip.

Modelo do AWS SAM para uma aplicação do DynamoDB

Você pode criar esse aplicativo usando AWS SAM. Para saber mais sobre como criar modelos do AWS SAM, consulte Noções básicas de modelos do AWS SAM no Guia do desenvolvedor do AWS Serverless Application Model.

Veja abaixo um modelo de exemplo do AWS SAM para o aplicativo do tutorial. Copie o texto abaixo para um arquivo .yaml e salve-o ao lado do pacote ZIP criado previamente. Observe que os valores dos parâmetros `Handler` e `Runtime` devem corresponder àqueles usados quando você criou a função na seção anterior.

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
```

Para obter informações sobre como empacotar e implantar o aplicativo sem servidor usando os comandos de empacotamento e implantação, consulte Implantar aplicativos sem servidor no Guia do desenvolvedor do AWS Serverless Application Model.

Como o Lambda processa registros do Amazon Kinesis Data Streams

Você pode usar uma função do Lambda para processar registros em um fluxo de dados do Amazon Kinesis. É possível mapear uma função do Lambda em um consumidor de throughput compartilhado (iterador padrão) do Kinesis Data Stream ou em um consumidor de throughput dedicado com distribuição avançada. Para iteradores padrão, o Lambda sonda cada fragmento no stream do Kinesis em busca de registros usando o protocolo HTTP. O mapeamento da origem do evento compartilha a throughput de leitura com outros consumidores do fragmento.

Para obter detalhes sobre transmissões de dados do Kinesis, consulteLer dados do Amazon Kinesis Data Streams.

> **Note**
>
> O Kinesis cobra por cada fragmento e, para distribuição avançada, dados lidos da transmissão. Para obter detalhes de preço, consulte Preço do Amazon Kinesis.

Fluxos de sondagem e agrupamento em lotes

O Lambda lê registros do fluxo de dados e invoca sua função de maneira síncrona com um evento que contém registros de transmissão. O Lambda lê registros em lotes e invoca sua função para processar registros do lote. Cada lote contém registros de um único fragmento/fluxo de dados.

Por padrão, o Lambda invoca a função assim que os registros estão disponíveis. Se o lote que o Lambda lê da fonte de eventos tiver apenas um registro, o Lambda enviará apenas um registro à função. Para evitar a invocação da função com poucos registros, instrua a fonte de eventos para

armazenar os registros em buffer por até cinco minutos, configurando uma janela de lotes. Antes de invocar a função, o Lambda continua a ler registros da fonte de eventos até coletar um lote inteiro, até que a janela de lote expire ou até que o lote atinja o limite de carga útil de 6 MB. Para ter mais informações, consulte Comportamento de lotes.

> **Warning**
>
> Os mapeamentos da origem do evento do Lambda processam cada evento ao menos uma vez, podendo haver o processamento duplicado de registros. Para evitar possíveis problemas relacionados a eventos duplicados, é altamente recomendável tornar o código da função idempotente. Para saber mais, consulte Como tornar minha função do Lambda idempotente no Centro de Conhecimentos da AWS.

Configure a opção ParallelizationFactor para processar um fragmento de um fluxo de dados do Kinesis com mais de uma invocação do Lambda simultaneamente. Você pode especificar o número de lotes simultâneos que o Lambda pesquisa de um fragmento por meio de um fator de paralelização de 1 (padrão) a 10. Por exemplo, ao definir `ParallelizationFactor` como 2, você poderá ter no máximo 200 invocações simultâneas do Lambda para processar 100 fragmentos de dados do Kinesis (embora, na prática, você possa ver valores diferentes para a métrica `ConcurrentExecutions`). Isso ajuda a aumentar o throughput de processamento quando o volume de dados é volátil e o valor de `IteratorAge` é alto. Quando você aumenta o número de lotes simultâneos por fragmento, o Lambda ainda garante o processamento em ordem no nível de chave de partição.

Você também pode usar `ParallelizationFactor` com a agregação do Kinesis. O comportamento do mapeamento da origem do evento depende de você estar ou não usando fan-out avançado:

- Sem fan-out avançado: todos os eventos dentro de um evento agregado devem ter a mesma chave de partição. A chave de partição também deve ser igual à do evento agregado. Se os eventos dentro do evento agregado tiverem chaves de partição diferentes, o Lambda não poderá garantir o processamento dos eventos na ordem, por chave de partição.
- Com fan-out avançado: primeiro o Lambda decodifica o evento agregado em eventos individuais. O evento agregado pode ter uma chave de partição diferente da chave dos eventos que ele contém. Porém, os eventos que não correspondem à chave de partição são descartados e perdidos. O Lambda não processa esses eventos e não os envia para um destino de falha configurado.

Evento de exemplo

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"
        }
```

```
    ]
}
```

Processar registros do Amazon Kinesis Data Streams com o Lambda

Para processar registros do Amazon Kinesis Data Streams com o Lambda, crie um consumidor para seu fluxo e, em seguida, crie um mapeamento de origem de evento do Lambda.

Configurar o stream de dados e a função

Sua função do Lambda é uma aplicação de consumidor para seu fluxo de dados. Ele processa um lote de registros por vez de cada estilhaço. É possível mapear uma função Lambda para um consumidor de throughput compartilhado (iterador padrão) ou para um consumidor de throughput dedicado com distribuição avançada.

- Iterador padrão: o Lambda sonda cada fragmento em seu fluxo do Kinesis para identificar registros a uma taxa básica de uma vez por segundo. Quando houver mais registros disponíveis, o Lambda mantém lotes de processamento até que a função alcance o fluxo. O mapeamento da origem do evento compartilha a throughput de leitura com outros consumidores do fragmento.
- Distribuição avançada: para minimizar a latência e maximizar o throughput da leitura, crie um consumidor de fluxo de dados com distribuição avançada. Os consumidores da distribuição avançada obtêm uma conexão dedicada para cada fragmento que não afeta outros aplicativos que fazem leitura do stream. Os consumidores de fluxos usam HTTP/2 para reduzir a latência, enviando os registros para o Lambda por meio de uma conexão de longa duração e compactando cabeçalhos de solicitação. Você pode criar um consumidor de fluxo com a API RegisterStreamConsumer do Kinesis.

```
aws kinesis register-stream-consumer \
--consumer-name con1 \
--stream-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
```

A seguinte saída deverá ser mostrada:

```
{
    "Consumer": {
        "ConsumerName": "con1",
        "ConsumerARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream/
consumer/con1:1540591608",
        "ConsumerStatus": "CREATING",
```

```
        "ConsumerCreationTimestamp": 1540591608.0
    }
}
```

Para aumentar a velocidade com que sua função processa registros, adicione fragmentos ao fluxo de dados. O Lambda processa registros em cada fragmento sequencialmente. Ele deixará de processar registros adicionais em um estilhaço se sua função retornar um erro. Com mais estilhaços, há mais lotes sendo processados de uma só vez, o que diminui o impacto de erros na simultaneidade.

Se a função não conseguir se expandir para lidar com o número total de lotes simultâneos, solicite um aumento de cota ou reserve simultaneidade para a função.

Criar um mapeamento da origem do evento para invocar uma função do Lambda

Para invocar sua função do Lambda com registros do fluxo de dados, crie um mapeamento da origem do evento. É possível criar vários mapeamentos de origem de eventos para processar os mesmos dados com várias funções do Lambda ou processar itens de vários fluxos de dados com uma única função. Ao processar itens de vários fluxos, cada lote conterá registros somente de um único fragmento ou transmissão.

Você pode configurar mapeamentos da origem do evento para processar registros de um fluxo em outra Conta da AWS. Para saber mais, consulte the section called “Mapeamentos entre contas”.

Antes de criar um mapeamento da origem do evento, você precisará dar permissão à função do Lambda para ler a partir de um fluxo de dados do Kinesis. O Lambda precisa das permissões a seguir para gerenciar recursos relacionados ao seu fluxo de dados do Kinesis:

- kinesis:DescribeStream
- kinesis:DescribeStreamSummary
- kinesis:GetRecords
- kinesis:GetShardIterator
- kinesis:ListShards
- kinesis:ListStreams
- kinesis:SubscribeToShard

A política gerenciada da AWS AWSLambdaKinesisExecutionRole inclui essas permissões. Adicione essa política gerenciada à sua função, conforme descrito no procedimento a seguir.

AWS Management Console

Para adicionar permissões do Kinesis à sua função

1. Abra a página Funções do console do Lambda e selecione a função.
2. Na guia Configuração, escolha Permissões.
3. No painel Perfil de execução, em Nome do perfil, escolha o link para o perfil de execução da sua função. Esse link abre a página para esse perfil no console do IAM.
4. No painel Políticas de permissões, escolha Adicionar permissões e, em seguida, selecione Anexar políticas.
5. No campo de pesquisa, digite **`AWSLambdaKinesisExecutionRole`**.
6. Marque a caixa de seleção próxima à política e escolha Adicionar permissão.

AWS CLI

Para adicionar permissões do Kinesis à sua função

- Execute o comando da CLI a seguir para adicionar a política `AWSLambdaKinesisExecutionRole` ao perfil de execução da sua função:

```
aws iam attach-role-policy \
--role-name MyFunctionRole \
--policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaKinesisExecutionRole
```

AWS SAM

Para adicionar permissões do Kinesis à sua função

- Na definição da sua função, adicione a `Policies` propriedade, conforme mostrado no seguinte exemplo:

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./my-function/
      Handler: index.handler
      Runtime: nodejs20.x
```

```
      Policies:
        - AWSLambdaKinesisExecutionRole
```

Depois de configurar as permissões necessárias, crie o mapeamento de origem de eventos.

AWS Management Console

Para criar o mapeamento da origem do evento do Kinesis

1. Abra a página Funções do console do Lambda e selecione a função.
2. No painel Visão geral da função, escolha Adicionar gatilho.
3. Em Configuração do acionador, para a origem, selecione Kinesis.
4. Selecione o fluxo do Kinesis para o qual você deseja criar o mapeamento da origem do evento e, opcionalmente, um consumidor do seu fluxo.
5. (Opcional) edite o Tamanho do lote, a Posição inicial e a Janela do lote para o mapeamento da origem do evento.
6. Escolha Adicionar.

Ao criar seu mapeamento da origem do evento no console, seu perfil do IAM deve ter as permissões kinesis:ListStreams e kinesis:ListStreamConsumers.

AWS CLI

Para criar o mapeamento da origem do evento do Kinesis

- Execute o comando a seguir da CLI para criar um mapeamento da origem do evento do Kinesis. Escolha seu próprio tamanho de lote e posição inicial de acordo com seu caso de 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
```

Para especificar uma janela de agrupamento em lotes, adicione a opção `--maximum-batching-window-in-seconds`. Para obter mais informações sobre como este e outros parâmetros, consulte create-event-source-mapping na Referência de comandos da AWS CLI.

AWS SAM

Para criar o mapeamento da origem do evento do Kinesis

- Na definição da sua função, adicione a `KinesisEvent` propriedade, conforme mostrado no seguinte exemplo:

```
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
```

Para saber mais sobre como criar um mapeamento de origem de eventos para o Kinesis Data Streams no AWS SAM, consulte Kinesis no Guia do desenvolvedor do AWS Serverless Application Model.

Posição inicial de sondagem e fluxo

Esteja ciente de que a sondagem do fluxo durante a criação e as atualizações do mapeamento da origem do evento é, finalmente, consistente.

- Durante a criação do mapeamento da origem do evento, pode levar alguns minutos para a sondagem de eventos do fluxo iniciar.
- Durante as atualizações do mapeamento da origem do evento, pode levar alguns minutos para interromper e reiniciar a sondagem de eventos do fluxo.

Esse comportamento significa que, se você especificar `LATEST` como posição inicial do fluxo, o mapeamento da origem do evento poderá perder eventos durante a criação ou as atualizações. Para garantir que nenhum evento seja perdido, especifique a posição inicial do fluxo como `TRIM_HORIZON` ou `AT_TIMESTAMP`.

Como criar mapeamentos da origem do evento entre contas

O Amazon Kinesis Data Streams permite políticas baseadas em recursos. Por isso, você pode processar dados ingeridos em um fluxo em uma Conta da AWS com uma função do Lambda em outra conta.

Para criar um mapeamento da origem do evento para sua função do Lambda usando um fluxo do Kinesis em outra Conta da AWS, você deve configurar o fluxo usando uma política baseada em recursos para dar permissão à função do Lambda para ler itens. Para saber como configurar seu fluxo para permitir o acesso entre contas, consulte Compartilhamento de acesso com funções do AWS Lambda entre contas no Guia do desenvolvedor do Amazon Kinesis Streams.

Depois de configurar seu fluxo com uma política baseada em recursos que conceda à sua função do Lambda as permissões necessárias, crie o mapeamento da origem do evento usando qualquer um dos métodos descritos na seção anterior.

Se você optar por criar seu mapeamento da origem do evento usando o console Lambda, cole o ARN do seu fluxo diretamente no campo de entrada. Se você quiser especificar um consumidor para seu fluxo, colar o ARN do consumidor preencherá automaticamente o campo do fluxo.

Configurar a resposta em lote parcial com o Kinesis Data Streams e o Lambda

Ao consumir e processar dados de transmissão de uma fonte de eventos, o Lambda definirá checkpoints por padrão no número mais elevado na sequência de um lote somente quando o lote for um sucesso total. O Lambda trata todos os outros resultados como uma falha completa e tenta processar novamente o lote até o limite de novas tentativas. Para permitir sucessos parciais durante o processamento de lotes de um stream, ative `ReportBatchItemFailures`. Permitir sucessos parciais pode ajudar a reduzir o número de novas tentativas em um registro, embora não impeça totalmente a possibilidade de novas tentativas em um registro bem-sucedido.

Para ativar `ReportBatchItemFailures`, inclua o valor de enum **`ReportBatchItemFailures`** na lista FunctionResponseTypes. Essa lista indica quais tipos de resposta estão habilitados para sua função. Você pode configurar essa lista ao criar ou atualizar um mapeamento de origem de eventos.

Sintaxe do relatório

Ao configurar relatórios sobre falhas de itens de lote, a classe `StreamsEventResponse` é retornada com uma lista de falhas de itens de lote. É possível usar um objeto `StreamsEventResponse` para retornar o número sequencial do primeiro registro com falha no lote. Você também pode criar sua própria classe personalizada usando a sintaxe de resposta correta. A seguinte estrutura JSON mostra a sintaxe de resposta necessária:

```
{
  "batchItemFailures": [
        {
            "itemIdentifier": "<SequenceNumber>"
        }
    ]
}
```

> **Note**
>
> Se a matriz `batchItemFailures` contém vários itens, o Lambda usa o registro com o menor número de sequência como ponto de verificação. Em seguida, o Lambda repete todos os registros a partir desse ponto de verificação.

Condições de sucesso e falha

O Lambda trata um lote como um sucesso completo se você retornar qualquer um destes:

- Uma lista de `batchItemFailure` vazia
- Uma lista de `batchItemFailure` nula
- Uma `EventResponse` vazia
- Uma `EventResponse` nula

O Lambda trata um lote como uma falha absoluta se você retornar qualquer um dos seguintes:

- Uma string `itemIdentifier` vazia

- Uma `itemIdentifier` nula
- Um `itemIdentifier` com um nome de chave inválido

O Lambda faz novas tentativas após falhas com base na sua estratégia de repetição.

Dividir um lote

Se a invocação falhar e `BisectBatchOnFunctionError` estiver ativado, o lote será dividido independentemente da configuração de `ReportBatchItemFailures`.

Quando uma resposta de sucesso parcial do lote é recebida e tanto `BisectBatchOnFunctionError` quanto `ReportBatchItemFailures` estão ativados, o lote é dividido no número de sequência retornado e o Lambda tenta novamente apenas os registros restantes.

Veja alguns exemplos de código de função que retornam a lista de IDs de mensagens com falha no lote:

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do Kinesis com o Lambda usando o .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do Kinesis com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do Kinesis com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do Kinesis com o Lambda usando 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;
}
```

Relatar falhas de itens em lote do Kinesis com o Lambda usando 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 para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do Kinesis com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do Kinesis com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de item em lote do Kinesis com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relate falhas de itens em lote do Kinesis com o Lambda usando 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
}
```

Reter registros de lotes descartados para uma origem de eventos do Kinesis Data Streams no Lambda

O tratamento de erros para mapeamentos de origem de eventos do Kinesis depende se o erro ocorre antes de a função ser invocada ou durante a invocação da função:

- Antes da invocação: se um mapeamento de origem de eventos do Lambda não conseguir invocar a função devido a limitações ou a outros problemas, ele tentará novamente até que os registros expirem ou excedam a idade máxima configurada no mapeamento de origem de eventos (MaximumRecordAgeInSeconds).
- Durante a invocação: se a função for invocada, mas retornar um erro, o Lambda tentará novamente até que os registros expirem, excedam a idade máxima (MaximumRecordAgeInSeconds) ou atinjam a cota de repetição configurada (MaximumRetryAttempts). Para erros de função, também é possível pode configurar BisectBatchOnFunctionError, que divide um lote com falha em dois lotes em lotes, isolando registros com problema e evitando exceder tempos limites. A divisão de lotes não consome a cota de repetição.

Se as medidas de tratamento de erros falharem, o Lambda descartará os registros e continuará processando lotes provenientes da transmissão. Com as configurações padrão, isso significa que um registro ruim pode bloquear o processamento no fragmento afetado por até uma semana. Para evitar isso, configure o mapeamento de fontes de eventos da sua função com um número razoável de tentativas e uma idade máxima de registro que se adapte ao seu caso de uso.

Configurar destinos para invocações com falha

Para reter registros de invocações de mapeamento da origem do evento com falha, adicione um destino ao mapeamento da origem de eventos da função. Cada registro enviado ao destino é um documento JSON com metadados sobre a invocação que falhou. É possível configurar qualquer tópico do Amazon SNS ou fila do Amazon SQS como destino. Sua função de execução deve ter permissões para o destino:

- Para destinos SQS: sqs:SendMessage
- Para destinos SNS: sns:Publish

Para configurar um destino em caso de falha usando o console, siga estas etapas:

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Em Function overview (Visão geral da função), escolha Add destination (Adicionar destino).
4. Em Origem, escolha Invocação do mapeamento da origem do evento.
5. Em Mapeamento da origem do evento, escolha uma origem de eventos configurada para essa função.
6. Em Condição, selecione Em caso de falha. Para invocações de mapeamento da origem de eventos, essa é a única condição aceita.
7. Em Tipo de destino, escolha o tipo de destino para o qual o Lambda envia registros de invocação.
8. Em Destination (Destino), escolha um recurso.
9. Escolha Salvar.

Também é possível configurar um destino em caso de falha usando a AWS Command Line Interface (AWS CLI). Por exemplo, o seguinte comando create-event-source-mapping adiciona um mapeamento de origem de eventos com um destino SQS em caso de falha 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"}}'
```

O comando update-event-source-mapping a seguir atualiza um mapeamento de origem de eventos para enviar registros de chamada com falha para um destino SNS após duas novas tentativas ou se os registros tiverem mais de uma hora.

```
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"}}'
```

As configurações atualizadas são aplicadas de forma assíncrona e não são refletidas na saída até que o processo seja concluído. Use o comando get-event-source-mapping para visualizar o status atual.

Para remover um destino, forneça uma string vazia como argumento para o parâmetro `destination-config`:

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

O exemplo a seguir mostra o que é enviado pelo Lambda para uma fila do SQS ou tópico do SNS em caso de falha na invocação da origem de eventos do Kinesis. Como o Lambda envia somente os metadados para esses tipos de destino, use os campos `streamArn`, `shardId`, `startSequenceNumber` e `endSequenceNumber` para obter o registro original 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"
    }
}
```

Você pode usar essas informações para recuperar os registros afetados da transmissão para solução de problemas. Os registros reais não estão incluídos, portanto, você deve processar esses registros e recuperá-los da transmissão antes que eles expirem e sejam perdidos.

Implementando o processamento com estado do Kinesis Data Streams no Lambda

As funções do Lambda podem executar aplicações de processamento contínuo de transmissões. Um stream representa dados não vinculados que fluem continuamente por meio de sua aplicação. Para analisar as informações dessa entrada de atualização contínua, você pode vincular os registros incluídos usando uma janela definida em termos de tempo.

As janelas de tumbling são janelas de tempo distintas que abrem e fecham em intervalos regulares. Por padrão, as invocações do Lambda são sem estado. Não é possível usá-las para processar dados ao longo de várias invocações contínuas sem um banco de dados externo. No entanto, com as janelas de tumbling, você pode manter seu estado em todas as invocações. Esse estado contém o resultado agregado das mensagens previamente processadas para a janela atual. Seu estado pode ter no máximo 1 MB por fragmento. Se exceder esse tamanho, o Lambda encerra a janela antes.

Cada registro de um fluxo pertence a uma janela específica. O Lambda processará cada registro pelo menos uma vez, mas não garantirá que cada registro seja processado apenas uma vez. Em casos raros, como tratamento de erros, alguns registros poderão ser processados mais de uma vez. Os registros são sempre processados em ordem na primeira vez. Se os registros forem processados mais de uma vez, poderão ser processados fora de ordem.

Agregação e processamento

Sua função gerenciada pelo usuário é chamada tanto para agregação quanto para processamento dos resultados finais dessa agregação. O Lambda agrega todos os registros recebidos na janela. Você pode receber esses registros em vários lotes, cada um como uma invocação separada. Cada invocação recebe um estado. Assim, ao usar janelas de tumbling, sua resposta de função do Lambda deve conter uma propriedade de `state`. Se a resposta não contiver uma propriedade de `state`, o Lambda considerará esta uma invocação com falha. Para satisfazer essa condição, a função pode retornar um objeto do `TimeWindowEventResponse`, que tem a seguinte forma JSON:

Example Valores de **`TimeWindowEventResponse`**

```
{
    "state": {
        "1": 282,
        "2": 715
    },
    "batchItemFailures": []
}
```

> **Note**
>
> Para funções Java, recomendamos o uso de um `Map<String, String>` para representar o estado.

No final da janela, a sinalização `isFinalInvokeForWindow` é definida como `true` para indicar que esse é o estado final e que está pronto para processamento. Após o processamento, a janela é concluída e sua invocação final é concluída e, em seguida, o estado é descartado.

No final da janela, o Lambda usa o processamento final para ações sobre os resultados da agregação. Seu processamento final é invocado de forma síncrona. Após a invocação bem-sucedida, sua função define os pontos de verificação no número da sequência e o processamento de streams continua. Se a invocação não for bem-sucedida, sua função do Lambda suspenderá o processamento adicional até uma chamada bem-sucedida.

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
}
```

Configuração

Você pode configurar janelas em cascata ao criar ou atualizar um mapeamento de fonte de eventos. Para configurar uma janela em cascata, especifique a janela em segundos (TumblingWindowInSeconds). O comando de exemplo da AWS Command Line Interface (AWS CLI) a seguir cria um mapeamento de fonte de eventos em streaming com uma janela em cascata de 120 segundos. A função do Lambda definida para agregação e processamento é chamada de `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
```

O Lambda determina os limites da janela em cascata com base no horário em que os registros foram inseridos no stream. Todos os registros têm um carimbo de data/hora aproximado disponível que o Lambda usa para determinar os limites.

As agregações de janelas em cascata não são compatíveis com refragmentação. Quando um fragmento termina, o Lambda considera a janela como fechada e os fragmentos filhos iniciam suas próprias janelas em um novo estado. Quando nenhum novo registro está sendo adicionado à janela atual, o Lambda aguarda até 2 minutos antes de assumir que a janela terminou. Isso ajuda a garantir que a função leia todos os registros na janela atual, mesmo que os registros sejam adicionados de forma intermitente.

As janelas em cascata são totalmente compatíveis com as políticas `maxRetryAttempts` e `maxRecordAge`.

Example Handler.py: agregação e processamento

A função do Python a seguir demonstra como agregar e, em seguida, processar seu estado final:

```
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}
```

Parâmetros do Lambda para mapeamento de origem de eventos do Amazon Kinesis Data Streams.

Todos os mapeamentos de origem de evento Lambda compartilham as mesmas operações de API CreateEventSourceMapping e UpdateEventSourceMapping. No entanto, apenas alguns dos parâmetros se aplicam ao Kinesis.

Parâmetros de fonte de evento que se aplicam ao Kinesis

Parâmetro	Obrigatório	Padrão	Observações
BatchSize	N	100	Máximo: 10.000.
BisectBatchOnFunctionError	N	false	
DestinationConfig	N		Uma fila do Amazon SQS ou um destino de tópico do Amazon SNS para registros descartados. Para ter mais informações, consulte Configurar destinos para invocações com falha.
Ativado	N	verdadeiro	
EventSourceArn	Y		O ARN do fluxo de dados ou um consumidor de fluxo
FunctionName	Y		

Parâmetro	Obrigatório	Padrão	Observações
FunctionResponseTypes	N		Para permitir que sua função reporte falhas específicas em um lote, inclua o valor `ReportBatchItemFailures` em `FunctionResponseTypes`. Para ter mais informações, consulte Configurar a resposta em lote parcial com o Kinesis Data Streams e o Lambda.
MaximumBatchingWindowInSeconds	N	0	
MaximumRecordAgeInSeconds	N	-1	-1 significa infinito: o Lambda não descarta registros (as configurações de retenção de dados do Kinesis Data Streams ainda se aplicam) Mínimo: -1 Máximo: 604.800

Parâmetro	Obrigatório	Padrão	Observações
MaximumRetryAttempts	N	-1	-1 significa infinito: registros com falha são repetidos até que o registro expire Mínimo: -1 Máximo: 10.000.
ParallelizationFactor	N	1	Máximo: 10
StartingPosition	Y		AT_TIMESTAMP, TRIM_HORIZON, ou LATEST
StartingPositionTimestamp	N		Válido somente se StartingPosition estiver definido como AT_TIMESTAMP. O tempo a partir do qual iniciar a leitura, em segundos no horário do Unix
TumblingWindowInSeconds	N		Mínimo: 0 Máximo: 900

Tutorial: Usar o Lambda com o Kinesis Data Streams

Neste tutorial, você criará uma função do Lambda para consumir eventos de um fluxo de dados do Amazon Kinesis.

1. O aplicativo personalizado grava registros no fluxo.
2. O AWS Lambda sonda a transmissão e, quando detecta novos registros, invoca sua função do Lambda.

3. O AWS Lambda executa a função do Lambda assumindo a função de execução especificada no momento da criação da função do Lambda.

Pré-requisitos

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em Criar uma função do Lambda com o console para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, a AWS Command Line Interface (AWS CLI) versão 2 será necessária. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

A seguinte saída deverá ser mostrada:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell.

> **Note**
>
> No Windows, alguns comandos da CLI do Bash que você costuma usar com o Lambda (como `zip`) não são compatíveis com os terminais integrados do sistema operacional. Para obter uma versão do Ubuntu com o Bash integrada no Windows, instale o Subsistema do Windows para Linux. Os exemplos de comandos da CLI neste guia usam a formatação Linux. Os comandos que incluem documentos JSON em linha deverão ser reformatados se você estiver usando a CLI do Windows.

Criar a função de execução

Crie a função de execução que dá à sua função permissão para acessar recursos do AWS.

Para criar uma função de execução

1. Abra a página Roles (Funções) no console do IAM.

2. Selecione Create role (Criar função).
3. Crie uma função com as propriedades a seguir.
   - Trusted entity (Entidade confiável): AWS Lambda.
   - Permissões: AWSLambdaKinesisExecutionRole.
   - Role name (Nome da função): **`lambda-kinesis-role`**.

A política AWSLambdaKinesisExecutionRole tem as permissões necessárias para a função ler itens do Kinesis e gravar logs no CloudWatch Logs.

Criar a função

Crie uma função do Lambda que processe suas mensagens do Kinesis. O código da função registra o ID do evento e os dados do evento do registro do Kinesis no CloudWatch Logs.

Este tutorial usa o runtime do Node.js 18.x, mas também fornecemos exemplos de arquivos em outras linguagens de runtime. Você pode selecionar a guia na caixa a seguir para ver o código do runtime do seu interesse. O código JavaScript que você usará nesta etapa é o primeiro exemplo mostrado na guia JavaScript.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando 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;
}
```

Consumir um evento do Kinesis com o Lambda usando 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 para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consuma um evento do Kinesis com o Lambda usando 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
}
```

Para criar a função

1. Crie um diretório para o projeto e depois mude para esse diretório.

```
mkdir kinesis-tutorial
cd kinesis-tutorial
```

2. Copie o código JavaScript de amostra em um novo arquivo denominado `index.js`.
3. Crie um pacote de implantação.

```
zip function.zip index.js
```

4. Crie uma função do Lambda com o 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
```

Testar a função do Lambda

Invoque sua função do Lambda manualmente usando o comando `invoke` da CLI do AWS Lambda e um evento do Kinesis de amostra.

Para testar a função do Lambda

1. Copie o JSON a seguir em um arquivo e salve-o como `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. Use o comando `invoke` para enviar o evento para a função.

```
aws lambda invoke --function-name ProcessKinesisRecords \
--cli-binary-format raw-in-base64-out \
--payload file://input.txt outputfile.txt
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

A resposta é salva no `out.txt`.

Criar uma transmissão do Kinesis

Use o comando `create-stream` para criar um fluxo.

```
aws kinesis create-stream --stream-name lambda-stream --shard-count 1
```

Execute o comando `describe-stream` a seguir para obter o ARN do stream.

```
aws kinesis describe-stream --stream-name lambda-stream
```

A seguinte saída deverá ser mostrada:

```
{
    "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
    }
}
```

Você usa o ARN do stream na próxima etapa para associar o stream à sua função do Lambda.

Adicionar uma fonte de eventos no AWS Lambda

Execute o comando AWS CLI a seguir da `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
```

Observe o ID do mapeamento para uso posterior. Você pode obter uma lista de mapeamentos de origem de evento executando o comando `list-event-source-mappings` a seguir.

```
aws lambda list-event-source-mappings --function-name ProcessKinesisRecords \
--event-source arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream
```

Na resposta, pode verificar que o valor do status é `enabled`. Os mapeamentos de origem de eventos podem ser desabilitados para pausar a pesquisa temporariamente, sem perder nenhum registro.

Testar a configuração

Para testar o mapeamento da origem do evento, adicione registros de eventos ao seu stream do Kinesis. O valor `--data` é uma string que a CLI codifica em base64 antes de enviá-la ao Kinesis. Você pode executar o mesmo comando mais de uma vez para adicionar vários registros ao fluxo.

```
aws kinesis put-record --stream-name lambda-stream --partition-key 1 \
--data "Hello, this is a test."
```

O Lambda usa a função de execução para ler registros da transmissão. Em seguida, ele invoca sua função do Lambda, transmitindo lotes de registros. A função decodifica os dados de cada registro e os registra, enviando a saída para CloudWatch Logs. Visualize os logs no console do CloudWatch.

Limpe os recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Para excluir a função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Escolha Excluir.
4. Insira o nome do perfil no campo de entrada de texto e escolha Delete (Excluir).

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Para excluir a transmissão do Kinesis

1. Faça login no AWS Management Console e abra o console do Kinesis em https://console.aws.amazon.com/kinesis.

2. Selecione a transmissão criada.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto.
5. Escolha Excluir.

Usar o Lambda com o Amazon MQ

> **Note**
>
> Se você deseja enviar dados para um destino que não seja uma função do Lambda ou enriquecer os dados antes de enviá-los, consulte Amazon EventBridge Pipes (Pipes do Amazon EventBridge).

O Amazon MQ é um serviço gerenciado de agente de mensagem para o Apache ActiveMQ e o RabbitMQ. Um agente de mensagens habilita aplicações de software e componentes para se comunicarem usando várias linguagens de programação, sistemas operacionais e protocolos de mensagens formais por meio de destinos de eventos de tópico ou fila.

O Amazon MQ também pode gerenciar instâncias do Amazon Elastic Compute Cloud (Amazon EC2) em seu nome instalando agentes do ActiveMQ ou RabbitMQ e fornecendo diferentes topologias de rede e outras necessidades de infraestrutura.

Você pode usar uma função do Lambda para processar registros do seu agente de mensagens do Amazon MQ. O Lambda invoca sua função por meio de um mapeamento de fontes de eventos, um recurso do Lambda que lê mensagens de seu agente e invoca a função de modo síncrono.

> **Warning**
>
> Os mapeamentos da origem do evento do Lambda processam cada evento ao menos uma vez, podendo haver o processamento duplicado de registros. Para evitar possíveis problemas relacionados a eventos duplicados, é altamente recomendável tornar o código da função idempotente. Para saber mais, consulte Como tornar minha função do Lambda idempotente no Centro de Conhecimentos da AWS.

O mapeamento de fontes de eventos do Amazon MQ tem as seguintes restrições de configuração:

- Simultaneidade: as funções do Lambda que usam um mapeamento de origem do evento do Amazon MQ têm uma configuração padrão de simultaneidade máxima. Para o ActiveMQ, o serviço Lambda limita o número de ambientes de execução simultânea a cinco. Para o RabbitMQ, o número de ambientes de execução simultânea é limitado a 1. Mesmo que você altere as configurações de simultaneidade reservada ou provisionada da função, o serviço Lambda não disponibilizará mais ambientes de execução. Para solicitar um aumento na simultaneidade máxima padrão, entre em contato com o AWS Support.
- Contas cruzadas: o Lambda não é compatível com o processamento de contas cruzadas. Não é possível usar o Lambda para processar registros de um agente de mensagens do Amazon MQ que esteja em uma Conta da AWS diferente.
- Autenticação: para o ActiveMQ, somente o SimpleAuthenticationPlugin do ActiveMQ é compatível. Para RabbitMQ, somente oPLAINMecanismo de autenticação é compatível. Os usuários devem usar o AWS Secrets Manager para gerenciar suas credenciais. Para obter mais informações sobre a autenticação do ActiveMQ, consulteIntegração de corretores ActiveMQ com LDAPnoGuia do desenvolvedor do Amazon MQ.
- Cota de conexão: os agentes têm um número máximo de conexões permitidas por protocolo de nível de transmissão de dados. Essa cota é baseada no tipo de instância do agente. Para obter mais informações, consulte o .Operadoresseção doCotas no Amazon MQnoGuia do desenvolvedor do Amazon MQ.
- Conectividade: você pode criar agentes em uma Virtual Private Cloud (VPC) pública ou privada. Para VPCs privadas, sua função do Lambda precisa acessar a VPC para receber mensagens. Para obter mais informações, consulte the section called “Configuração de rede” mais adiante neste tópico.
- Destinos de eventos: somente destinos de fila são compatíveis. No entanto, você pode usar um tópico virtual, que se comporta como um tópico internamente enquanto interage com o Lambda como uma fila. Para obter mais informações, consulte Virtual Destinations no site do Apache ActiveMQ e Virtual Hosts no site do RabbitMQ.
- Topologia de rede: para o ActiveMQ, somente um agente de instância única ou em espera é aceito por mapeamento de fontes de eventos. Para o RabbitMQ, apenas um agente de instância única ou implantação de cluster é aceito por mapeamento de fonte de eventos. Os agentes de instância única requerem um endpoint de failover. Para obter mais informações sobre esses modos de implantação do agente, consulteArquitetura de operador do MQeArquitetura de MQ do RabbitnoGuia do desenvolvedor do Amazon MQ.
- Protocolos — Os protocolos suportados dependem do tipo de integração do Amazon MQ.

- Para integrações do ActiveMQ, o Lambda consome mensagens usando o protocolo OpenWire/Java Message Service (JMS). Nenhum outro protocolo é compatível para o consumo de mensagens. Dentro do protocolo JMS, somente `TextMessage` e `BytesMessage` são compatíveis. O Lambda também é compatível com propriedades personalizadas do JMS. Para obter mais informações sobre o protocolo OpenWire, consulte OpenWire no site do Apache ActiveMQ.
- Para integrações RabbitMQ, o Lambda consome mensagens usando o protocolo AMQP 0-9-1. Nenhum outro protocolo é compatível para o consumo de mensagens. Para obter mais informações sobre a implementação do protocolo AMQP 0-9-1 pelo RabbitMQ, consulteAMQP 0-9-1 Guia de referência completono site do RabbitMQ.

O Lambda oferece suporte automaticamente às versões mais recentes do ActiveMQ e RabbitMQ que não têm suporte no Amazon MQ. Para obter as versões compatíveis mais recentes, consulteNotas de versão do Amazon MQnoGuia do desenvolvedor do Amazon MQ.

> **Note**
>
> Por padrão, o Amazon MQ tem uma janela de manutenção semanal para agentes. Durante essa janela de tempo, os agentes não estão disponíveis. Para agentes sem espera, o Lambda não é possível processar nenhuma mensagem durante essa janela.

Seções

Grupo de consumidores do Lambda

Para interagir com o Amazon MQ, o Lambda cria um grupo de consumidores que pode ler a partir de seus agentes do Amazon MQ. O grupo de consumidores é criado com o mesmo ID que um UUID de mapeamento da fonte de eventos.

Para fontes de evento do Amazon MQ, o Lambda divide os registros em lotes e os envia para sua função em uma única carga útil. Para controlar o comportamento, é necessário configurar a janela de lotes e o tamanho do lote. O Lambda extrai mensagens até processar o tamanho máximo da carga útil de 6 MB, a janela de lotes expirar ou o número de registros atingir o tamanho total do lote. Para ter mais informações, consulte Comportamento de lotes.

O grupo de consumidores recupera as mensagens como um BLOB de bytes, as codifica em base64 para uma única carga útil JSON e depois invoca a função. Se sua função retorna um erro para qualquer uma das mensagens em um lote, o Lambda tenta novamente todo o lote de mensagens até que o processamento seja bem-sucedido ou as mensagens expiram.

> **Note**
>
> Embora as funções do Lambda normalmente tenham um limite máximo de tempo de 15 minutos, os mapeamentos da origem dos eventos para o Amazon MSK, o Apache Kafka autogerenciado, o Amazon DocumentDB e o Amazon MQ para ActiveMQ e RabbitMQ são compatíveis somente com funções com limites máximos de tempo limite de 14 minutos. Essa restrição garante que o mapeamento da origem do evento possa solucionar adequadamente os erros de função e repetições.

Você pode monitorar o uso da simultaneidade de uma determinada função usando a métrica `ConcurrentExecutions` no Amazon CloudWatch. Para obter mais informações sobre a simultaneidade, consulte the section called “Configurar a simultaneidade reservada”.

Example Eventos de registro do 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**
>
> No exemplo RabbitMQ,`pizzaQueue`é o nome da fila do RabbitMQ e`/`é o nome do host virtual. Ao receber mensagens, a origem do evento lista as mensagens em`pizzaQueue::/`.

Permissões da função de execução

Para ler registros de um agente do Amazon MQ, sua função do Lambda precisa das seguintes permissões adicionadas à respectiva função de execução:

- mq:DescribeBroker
- secretsmanager:GetSecretValue
- ec2:CreateNetworkInterface
- ec2:DeleteNetworkInterface
- ec2:DescribeNetworkInterfaces
- ec2:DescribeSecurityGroups
- ec2:DescribeSubnets

- ec2:DescribeVpcs
- logs:CreateLogGroup
- logs:CreateLogStream
- logs:PutLogEvents

> **Note**
>
> Ao usar uma chave criptografada gerenciada pelo cliente, adicione a permissão `kms:Decrypt` também.

Configuração de rede

Para dar ao Lambda acesso total ao seu agente por meio do mapeamento da origem do evento, seu agente deve usar um endpoint público (endereço IP público) ou você deve fornecer acesso à Amazon VPC, na qual o agente foi criado.

Por padrão, quando você cria um agente do Amazon MQ, o sinalizador `PubliclyAccessible` é definido como false. Para que seu agente receba um endereço IP público, você deve definir o sinalizador `PubliclyAccessible` como true.

A prática recomendada para usar o Amazon MQ com o Lambda é utilizar endpoints da VPC do AWS PrivateLink e permitir que a função do Lambda tenha acesso à VPC do seu agente. Implemente um endpoint para o Lambda e, exclusivamente para o ActiveMQ, um endpoint para AWS Security Token Service (AWS STS). Se o seu agente usar autenticação, implante também um endpoint para AWS Secrets Manager. Para saber mais, consulte the section called “Trabalhar com endpoints da VPC”.

Como alternativa, configure um gateway NAT em cada sub-rede pública na VPC que contém seu agente do Amazon MQ. Para ter mais informações, consulte the section called “Acesso à Internet para funções da VPC”.

Quando você cria um mapeamento da origem do evento para um agente do Amazon MQ, o Lambda verifica se as interfaces de rede elástica (ENIs) já estão presentes nas sub-redes e nos grupos de segurança da VPC do seu agente. Se o Lambda encontrar ENIs existentes, ele tentará reutilizá-las. Caso contrário, o Lambda criará novas ENIs para se conectar à origem do evento e invocar sua função.

> **Note**
>
> As funções do Lambda sempre são executadas em VPCs de propriedade do serviço Lambda. Essas VPCs recebem manutenção automática do serviço e não são visíveis para os clientes. Você também pode conectar sua função a uma Amazon VPC. Em ambos os casos, a configuração de VPC da sua função não afetará o mapeamento da origem do evento. Somente a configuração da VPC da origem de eventos determina o modo de conexão do Lambda à sua origem de eventos.

Regras de grupos de segurança da VPC

Configure com as seguintes regras (no mínimo) os grupos de segurança da Amazon VPC que contêm seu cluster:

- Regras de entrada: permitem todo o tráfego na porta do agente para os grupos de segurança especificados para sua origem de eventos de seu próprio grupo de segurança. O ActiveMQ usa a porta 61617 por padrão e o RabbitMQ usa a porta 5671 por padrão.
- Regras de saída: permitir todo o tráfego na porta 443 para todos os destinos. Permita que todo o tráfego na porta do agente esteja dentro de seu próprio grupo de segurança. O ActiveMQ usa a porta 61617 por padrão e o RabbitMQ usa a porta 5671 por padrão.
- Se você usar endpoints da VPC em vez de um gateway NAT, os grupos de segurança associados aos endpoints da VPC deverão permitir todo o tráfego de entrada na porta 443 dos grupos de segurança da origem de evento.

Trabalhar com endpoints da VPC

Quando você usa endpoints da VPC, as chamadas de API para invocar sua função são roteadas por esses endpoints usando as ENIs. A entidade principal do serviço Lambda precisa chamar `lambda:InvokeFunction` para quaisquer funções que usem essas ENIs. Além disso, para o ActiveMQ, a entidade principal do serviço Lambda precisa chamar `sts:AssumeRole` em perfis que usem os ENIs.

Por padrão, os endpoints da VPC têm políticas do IAM que são abertas. A prática recomendada é restringir essas políticas a fim de permitir somente que entidades principais específicas executem as ações necessárias usando esse endpoint. Para garantir que seu mapeamento da origem do evento seja capaz de invocar sua função do Lambda, a política de endpoint da VPC deve

permitir que a entidade principal do serviço Lambda chame `lambda:InvokeFunction` e, para o ActiveMQ, `sts:AssumeRole`. A restrição de suas políticas de endpoint da VPC para apenas permitir chamadas de API originadas em sua organização impedirá o funcionamento adequado do mapeamento da origem do evento.

O seguinte exemplo de políticas de endpoint da VPC mostra como conceder o acesso necessário ao AWS STS e aos endpoints do Lambda.

Example Política de endpoint da VPC - endpoint do AWS STS (somente ActiveMQ)

```
{
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Example Política de endpoint da VPC: endpoint do Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Se o seu agente do Amazon MQ usar autenticação, você também poderá restringir a política de endpoint da VPC para o endpoint do Secrets Manager. Para chamar a API do Secrets Manager, o Lambda usa seu perfil de função, e não a entidade principal de serviço do Lambda. O exemplo a seguir mostra uma política de endpoint do Secrets Manager.

Example Política de endpoint da VPC: endpoint do Secrets Manager.

```
{
    "Statement": [
        {
            "Action": "secretsmanager:GetSecretValue",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "customer_function_execution_role_arn"
                ]
            },
            "Resource": "customer_secret_arn"
        }
    ]
}
```

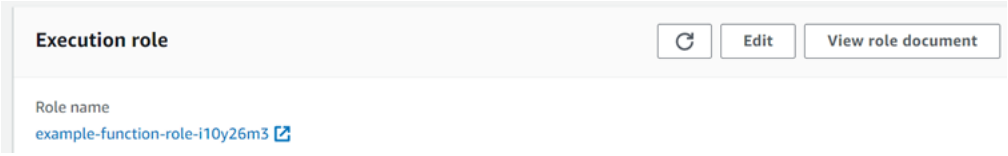
Adicionar permissões e criar o mapeamento da origem do evento

Crie um mapeamento de fontes de eventos para instruir o Lambda a enviar registros de um agente do Amazon MQ para uma função do Lambda. É possível criar vários mapeamentos de origem de evento para processar os mesmos dados com várias funções ou processar itens de várias fontes com uma única função.

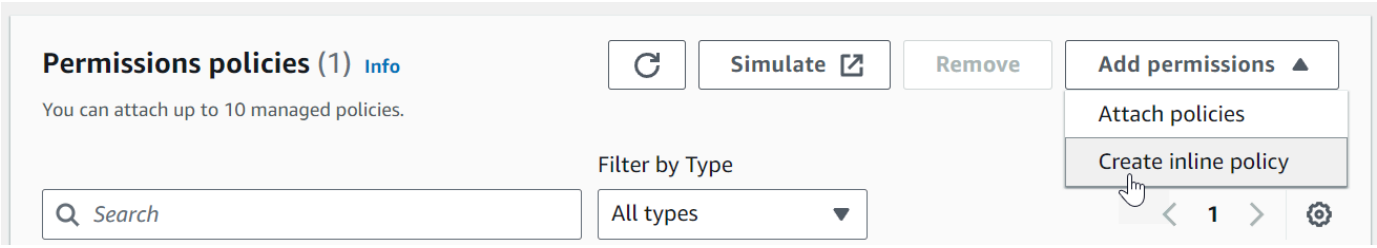
Para configurar sua função para ler do Amazon MQ, adicione as permissões necessárias e crie um acionador do MQ no console do Lambda.

Para adicionar permissões e criar um acionador

1. Abra a página Funções do console do Lambda.
2. Escolha o nome de uma função.
3. Escolha a guia Configuration (Configuração) e, depois, Permissions (Permissões).
4. Em Nome do perfil, escolha o link para seu perfil de execução. Esse link abre o perfil no console do IAM.



5. Escolha Adicionar permissões e, em seguida, Criar política em linha.

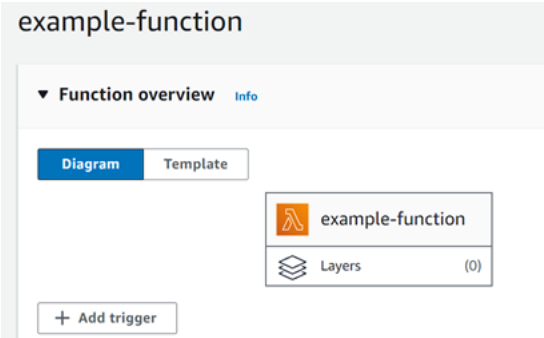


6. Na seção Editor de políticas, escolha JSON. Insira a seguinte política. A função precisa dessas permissões para ler de um agente do Amazon MQ.

```
{
  "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**
>
> Ao usar uma chave gerenciada pelo cliente criptografada, também é necessário adicionar a permissão `kms:Decrypt`.

7. Escolha Próximo. Digite um nome para a política e escolha Criar política.
8. Volte para a sua função no console do Lambda Em Visão geral da função, escolha Adicionar gatilho.



9. Selecione o tipo de acionador MQ.
10. Configure as opções necessárias e escolha Add (Adicionar).

O Lambda oferece suporte às seguintes opções de fontes de eventos do Amazon MQ:

- Agente do MQ— Selecione um corretor do Amazon MQ.
- Batch size (Tamanho do lote): defina o número máximo de mensagens a serem recuperadas em um único lote.
- Queue name (Nome da fila): digite a fila do Amazon MQ a ser consumida.
- Configuração do acesso à fonte— Insira as informações do host virtual e o segredo do Secrets Manager que armazena as credenciais do agente.
- Enable trigger (Habilitar acionador): desabilite o acionador para interromper o processamento de registros.

Para habilitar ou desabilitar o trigger (ou excluí-lo), selecione o trigger do MQ no designer. Para reconfigurar o trigger, use as operações da API de mapeamento de fontes de eventos.

Atualizar o mapeamento da origem do evento

Use o comando `update-event-source-mapping` para atualizar um mapeamento de origem de evento. O exemplo de comando a seguir atualiza um mapeamento de fonte de eventos para ter um tamanho de lote igual a 2.

```
aws lambda update-event-source-mapping \
--uuid 91eaeb7e-c976-1234-9451-8709db01f137 \
--batch-size 2
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
}
```

O Lambda atualiza essas configurações de forma assíncrona. A saída não refletirá as alterações até que esse processo seja concluído. Use o comando `get-event-source-mapping` para visualizar o status atual do seu recurso.

```
aws lambda get-event-source-mapping \
--uuid 91eaeb7e-c976-4939-9451-8709db01f137
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
}
```

Erros de mapeamento da fonte de eventos

Quando uma função do Lambda encontra um erro irrecuperável, o consumidor do Amazon MQ interrompe o processamento de registros. Qualquer outro consumidor pode continuar o processamento, contanto que não encontre o mesmo erro. Para determinar a possível causa de um consumidor interrompido, verifique o campo `StateTransitionReason` nos detalhes de devolução do `EventSourceMapping` para obter um dos seguintes códigos:

`ESM_CONFIG_NOT_VALID`

A configuração do mapeamento da fonte de eventos não é válida.

`EVENT_SOURCE_AUTHN_ERROR`

O Lambda falhou ao autenticar a fonte de eventos.

`EVENT_SOURCE_AUTHZ_ERROR`

O Lambda não tem as permissões necessárias para acessar a fonte de eventos.

`FUNCTION_CONFIG_NOT_VALID`

A configuração da função não é válida.

Os registros também não serão processados se o Lambda os descartar devido ao seu tamanho. O limite de tamanho para registros do Lambda é de 6 MB. Para entregar mensagens novamente após um erro da função, use uma Dead Letter Queue (DLQ – Fila de mensagens mortas). Para obter mais informações, consulte Message Redelivery and DLQ Handling no site do Apache ActiveMQ e Reliability Guide no site do RabbitMQ.

> **Note**
>
> O Lambda não oferece suporte às políticas de reentrega personalizadas. Em vez disso, o Lambda usa uma política com os valores padrão da página Política de reentrega no site do Apache ActiveMQ, com `maximumRedeliveries` definido como 6.

Parâmetros de configuração do Amazon MQ e RabbitMQ

Todos os tipos de origem de evento Lambda compartilham o mesmoCreateEventSourceMappingeUpdateEventSourceMappingOperações de API do. No entanto, apenas alguns dos parâmetros se aplicam ao Amazon MQ e RabbitMQ.

Parâmetros de fonte de evento que se aplicam ao Amazon MQ e RabbitMQ

Parâmetro	Obrigatório	Padrão	Observações
BatchSize	N	100	Máximo: 10.000.
Habilitado	N	verdadeiro	
FunctionName	Y		
FilterCriteria	N		Filtragem de eventos do Lambda
MaximumBatchingWindowInSeconds	N	500 ms	Comportamento de lotes
Filas	N		O nome da fila de destino do agente do Amazon MQ a ser consumido.
SourceAccessConfigurations	N		Para credencia is ActiveMQ, BASIC_AUTH. Para RabbitMQ, pode conter credencia is BASIC_AUTH e informações de VIRTUAL_HOST.

Usar o Lambda com o Amazon MSK

> **Note**
>
> Se você deseja enviar dados para um destino que não seja uma função do Lambda ou enriquecer os dados antes de enviá-los, consulte Amazon EventBridge Pipes (Pipes do Amazon EventBridge).

O Amazon Managed Streaming for Apache Kafka (Amazon MSK) é um serviço totalmente gerenciado que você pode usar para criar e executar aplicações que usam o Apache Kafka para processar dados em streaming. O Amazon MSK simplifica a configuração, o dimensionamento e o gerenciamento de clusters que executam o Kafka. O Amazon MSK também facilita a configuração de seu aplicativo para várias zonas de disponibilidade e para segurança comAWS Identity and Access Management(IAM). O Amazon MSK é compatível com várias versões de código aberto do Kafka.

O Amazon MSK como uma origem de evento funciona de forma semelhante ao uso do Amazon Simple Queue Service (Amazon SQS) ou do Amazon Kinesis. O Lambda pesquisa internamente por novas mensagens da origem do evento e, em seguida, chama de forma síncrona a função do Lambda de destino. O Lambda lê as mensagens em lotes e fornece estas para a sua função como uma carga de eventos. O tamanho máximo do lote é configurável (o padrão é 100 mensagens). Para ter mais informações, consulte Comportamento de lotes.

> **Note**
>
> Embora as funções do Lambda normalmente tenham um limite máximo de tempo de 15 minutos, os mapeamentos da origem dos eventos para o Amazon MSK, o Apache Kafka autogerenciado, o Amazon DocumentDB e o Amazon MQ para ActiveMQ e RabbitMQ são compatíveis somente com funções com limites máximos de tempo limite de 14 minutos. Essa restrição garante que o mapeamento da origem do evento possa solucionar adequadamente os erros de função e repetições.

O Lambda lê as mensagens sequencialmente para cada partição. Uma única carga do Lambda pode conter mensagens de várias partições. Depois que o Lambda processa cada lote, ele confirma os deslocamentos das mensagens nesse lote. Se sua função retorna um erro para qualquer uma das mensagens em um lote, o Lambda tenta novamente todo o lote de mensagens até que o processamento seja bem-sucedido ou as mensagens expiram.

> **Warning**
>
> Os mapeamentos da origem do evento do Lambda processam cada evento ao menos uma vez, podendo haver o processamento duplicado de registros. Para evitar possíveis problemas relacionados a eventos duplicados, é altamente recomendável tornar o código da função idempotente. Para saber mais, consulte Como tornar minha função do Lambda idempotente no Centro de Conhecimentos da AWS.

Para obter um exemplo de como configurar o Amazon MSK como uma origem de evento, consulteUsar o Amazon MSK como uma origem de evento para o AWS Lambda no blog AWS Compute. Para obter um tutorial completo, consulte Amazon MSK Lambda Integration (Integração do Lambda no Amazon MSK) no Amazon MSK Labs.

Tópicos

Tutorial: Usar um mapeamento de origem de evento do Amazon MSK para invocar uma função do Lambda

Neste tutorial, você fará o seguinte:

- Criar uma função do Lambda na mesma conta da AWS que um cluster do Amazon MSK existente.
- Configurar a rede e a autenticação para que o Lambda se comunique com o Amazon MSK.
- Configurar um mapeamento de origem de evento do Amazon MSK do Lambda, o qual executa sua função do Lambda quando os eventos aparecem no tópico.

Depois de concluir essas etapas, quando os eventos forem enviados para o Amazon MSK, você poderá configurar uma função do Lambda para processar esses eventos automaticamente com seu próprio código do Lambda personalizado.

O que você pode fazer com esse recurso?

Exemplo de solução: use um mapeamento de origem de evento do MSK para fornecer pontuações ao vivo para seus clientes.

Considere o seguinte cenário: sua empresa hospeda uma aplicação Web em que seus clientes podem ver informações sobre eventos ao vivo, como competições esportivas. As atualizações de informações do jogo são fornecidas à sua equipe por meio de um tópico do Kafka no Amazon MSK. Você deseja criar uma solução que consuma atualizações do tópico do MSK para fornecer uma visão atualizada do evento ao vivo aos clientes dentro de uma aplicação desenvolvida por você. Você optou pela seguinte abordagem de design: suas aplicações clientes se comunicarão com um back-end sem servidor hospedado na AWS. Os clientes se conectarão por meio de sessões de websocket usando a API de WebSocket do Amazon API Gateway.

Nessa solução, você precisa de um componente que leia eventos do MSK, execute uma lógica personalizada para preparar esses eventos para a camada de aplicação e, em seguida, encaminhe essas informações para a API do API Gateway. Você pode implementar esse componente com o AWS Lambda, fornecendo sua lógica personalizada em uma função do Lambda e, em seguida, chamando-a com um mapeamento de origem de eventos do Amazon MSK do AWS Lambda.

Para obter mais informações sobre a implementação de soluções usando a API de WebSocket do Amazon API Gateway, consulte os tutoriais da API de WebSocket na documentação do API Gateway.

Pré-requisitos

Uma conta da AWS com os seguintes recursos pré-configurados:

Para cumprir esses pré-requisitos, recomendamos seguir a seção Conceitos básicos do uso do Amazon MSK na documentação do Amazon MSK.

- Um cluster do Amazon MSK. Consulte Criar um cluster do Amazon MSK em Conceitos básicos do uso do Amazon MSK.
- A seguinte configuração:
  - Certifique-se de que a autenticação baseada em perfis do IAM esteja habilitada nas configurações de segurança do cluster. Isso melhora sua segurança ao limitar sua função do Lambda para acessar somente os recursos necessários do Amazon MSK. Esse comportamento é habilitado por padrão em todos os novos clusters do Amazon MSK.
  - Certifique-se de que o acesso público esteja desativado nas configurações de rede do cluster. Restringir o acesso do cluster do Amazon MSK à Internet melhora sua segurança ao limitar o número de intermediários que lidam com seus dados. Esse comportamento é habilitado por padrão em todos os novos clusters do Amazon MSK.
- Um tópico do Kafka no cluster do Amazon MSK para usar nesta solução. Consulte Criar um tópico em Conceitos básicos do uso do Amazon MSK.
- Um host administrativo do Kafka configurado para recuperar informações do cluster do Kafka e enviar eventos do Kafka para seu tópico para teste, como uma instância do Amazon EC2 com a CLI de administração do Kafka e a biblioteca do IAM do Amazon MSK instalada. Consulte Criar uma máquina cliente em Conceitos básicos do uso do Amazon MSK.

Depois de configurar esses recursos, reúna as seguintes informações da sua conta da AWS para confirmar que está tudo pronto para continuar.

- O nome do cluster do Amazon MSK. Essas informações podem ser encontradas no console do Amazon MSK.
- O UUID do cluster, parte do ARN do seu cluster do Amazon MSK, o qual pode ser encontrado no console do Amazon MSK. Siga os procedimentos em Listar clusters na documentação do Amazon MSK para encontrar essas informações.
- Os grupos de segurança associados ao cluster do Amazon MSK. Essas informações podem ser encontradas no console do Amazon MSK. Nas etapas a seguir, faça referência a elas como *`clusterSecurityGroups`*.
- O ID da Amazon VPC contendo seu cluster do Amazon MSK. É possível encontrar essas informações identificando sub-redes associadas ao seu cluster do Amazon MSK no console do Amazon MSK e, em seguida, identificando a Amazon VPC associada à sub-rede no console da Amazon VPC.
- O nome do tópico do Kafka usado na solução. Você pode encontrar essas informações chamando o cluster do Amazon MSK com a CLI de `topics` do Kafka no host administrativo do Kafka.

Para obter mais informações sobre a CLI de tópicos, consulte Adicionar e remover tópicos na documentação do Kafka.

- O nome de um grupo de consumidores para seu tópico do Kafka, adequado para uso por sua função do Lambda. Esse grupo pode ser criado automaticamente pelo Lambda, então não é necessário criá-lo com a CLI do Kafka. Se você precisar gerenciar seus grupos de consumidores, para saber mais sobre a CLI de grupos de consumidores, consulte Gerenciar grupos de consumidores na documentação do Kafka.

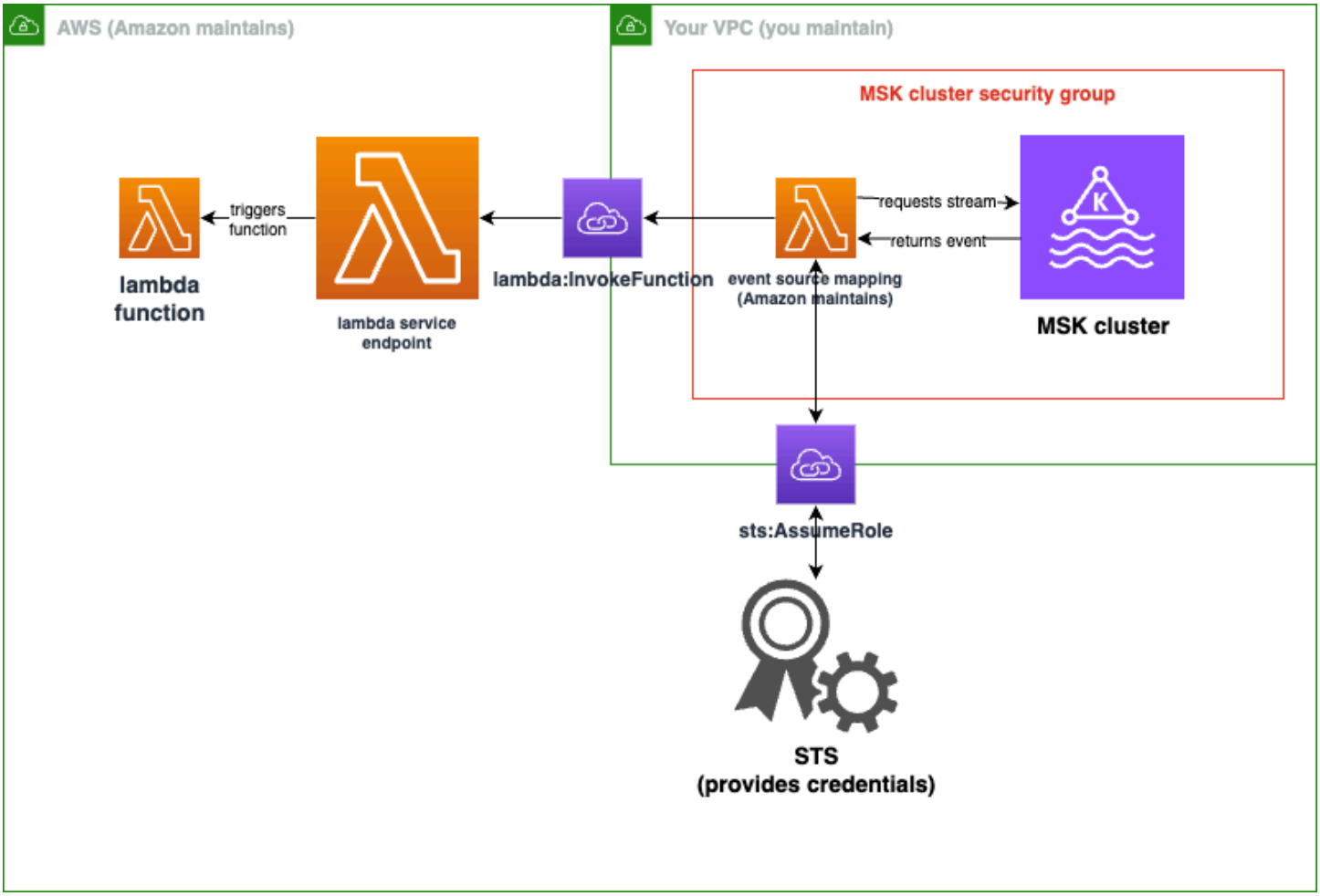
As seguintes permissões em sua conta da AWS:

- Permissão para criar e gerenciar uma função do Lambda
- Permissão para criar políticas do IAM e associá-las à sua função do Lambda.
- Permissão para criar endpoints da Amazon VPC e alterar a configuração de rede na Amazon VPC que hospeda seu cluster do Amazon MSK.

Configurar a conectividade de rede para que o Lambda se comunique com o Amazon MSK

Use AWS PrivateLink para conectar o Lambda e o Amazon MSK. Isso pode ser feito com a criação de endpoints da Amazon VPC de interface no console do Amazon VPC. Para obter mais informações sobre a configuração de rede, consulte the section called “Configuração de rede”.

Quando um mapeamento de origem de evento do Amazon MSK é executado em nome de uma função do Lambda, ele assume o perfil de execução da função do Lambda. Essa função do IAM autoriza o mapeamento a acessar recursos protegidos pelo IAM, como seu cluster do Amazon MSK. Embora os componentes compartilhem uma função de execução, o mapeamento do Amazon MSK e sua função do Lambda têm requisitos de conectividade separados para suas respectivas tarefas, conforme mostrado no diagrama a seguir.



Seu mapeamento da origem do evento pertence ao seu grupo de segurança do cluster do Amazon MSK. Nesta etapa de rede, crie endpoints do Amazon VPC da VPC do cluster do Amazon MSK para conectar o mapeamento da origem do evento aos serviços do Lambda e STS. Proteja esses endpoints para aceitar tráfego do seu grupo de segurança de cluster do Amazon MSK. Em seguida, ajuste os grupos de segurança do cluster do Amazon MSK para permitir que o mapeamento da origem do evento se comunique com o cluster do Amazon MSK.

Você pode configurar as etapas a seguir usando o AWS Management Console.

Para configurar os endpoints da Amazon VPC de interface para conectar o Lambda e o Amazon MSK

1. Crie um grupo de segurança para os endpoints da Amazon VPC de interface, *`endpointSecurityGroup`*, que permite tráfego TCP de entrada na porta 443 proveniente de *`clusterSecurityGroups`*. Siga o procedimento em Criar um grupo de segurança na documentação do Amazon EC2 para criar um grupo de segurança. Em seguida, siga o

procedimento em Adicionar regras a um grupo de segurança na documentação do Amazon EC2 para adicionar as regras apropriadas.

Crie um grupo de segurança com as seguintes informações:

Ao adicionar suas regras de entrada, crie uma regra para cada grupo de segurança em *clusterSecurityGroups*. Para cada regra:

- Em Tipo, selecione HTTPS.
- Em Origem, selecione um dos *clusterSecurityGroups*.

2. Crie um endpoint conectando o serviço do Lambda à Amazon VPC contendo seu cluster do Amazon MSK. Siga o procedimento em Criar um endpoint de interface.

   Crie um endpoint de interface com as seguintes informações:

   - Em Nome do serviço, selecione `com.amazonaws.`*regionName*`.lambda`, onde *regionName* hospeda sua função do Lambda.
   - Para VPC, selecione a Amazon VPC que contém seu cluster do Amazon MSK.
   - Em Grupos de segurança, selecione o *endpointSecurityGroup* que você criou anteriormente.
   - Para Sub-redes, selecione as sub-redes que hospedam seu cluster do Amazon MSK.
   - Para Política, forneça o documento de política a seguir, que protege o endpoint para uso pela entidade principal do serviço do Lambda para a ação `lambda:InvokeFunction`.

     ```
     {
         "Statement": [
             {
                 "Action": "lambda:InvokeFunction",
                 "Effect": "Allow",
                 "Principal": {
                     "Service": [
                         "lambda.amazonaws.com"
                     ]
                 },
                 "Resource": "*"
             }
         ]
     }
     ```

   - Certifique-se de que a opção Habilitar nome de DNS permaneça selecionada.

3. Crie um endpoint conectando o serviço do AWS STS à Amazon VPC que contém seu cluster do Amazon MSK. Siga o procedimento em Criar um endpoint de interface.

   Crie um endpoint de interface com as seguintes informações:

   - Em Nome do serviço, selecione AWS STS.
   - Para VPC, selecione a Amazon VPC que contém seu cluster do Amazon MSK.
   - Em Grupos de segurança, selecione *endpointSecurityGroup*.
   - Para Sub-redes, selecione as sub-redes que hospedam seu cluster do Amazon MSK.
   - Para Política, forneça o documento de política a seguir, que protege o endpoint para uso pela entidade principal do serviço do Lambda para a ação `sts:AssumeRole`.

   ```
   {
       "Statement": [
           {
               "Action": "sts:AssumeRole",
               "Effect": "Allow",
               "Principal": {
                   "Service": [
                       "lambda.amazonaws.com"
                   ]
               },
               "Resource": "*"
           }
       ]
   }
   ```

   - Certifique-se de que a opção Habilitar nome de DNS permaneça selecionada.

4. Para cada grupo de segurança associado ao seu cluster do Amazon MSK, ou seja, em *clusterSecurityGroups*, permita o seguinte:

   - Permita todo o tráfego TCP de entrada e saída na porta 9098 para todos os *clusterSecurityGroups*, inclusive dentro dele mesmo.
   - Permita todo o tráfego TCP de saída na porta 443.

   Parte desse tráfego é permitido por regras padrão do grupo de segurança. Portanto, se seu cluster estiver vinculado a um único grupo de segurança e esse grupo tiver regras padrão, regras adicionais não serão necessárias. Para adicionar regras do grupo de segurança, siga

o procedimento em Adicionar regras a um grupo de segurança na documentação do Amazon EC2.

Adicione regras aos seus grupos de segurança com as seguintes informações:

- Para cada regra de entrada ou regra de saída para a porta 9098, forneça
  - Em Tipo, selecione TCP personalizado.
  - Em Intervalo de portas, forneça 9098.
  - Em Origem, forneça um dos *`clusterSecurityGroups`*.
- Para cada regra de entrada da porta 443, em Tipo, selecione HTTPS.

Crie um perfil do IAM para que o Lambda leia seu tópico do Amazon MSK

Identifique os requisitos de autenticação para que o Lambda leia seu tópico do Amazon MSK e, em seguida, defina-os em uma política. Crie uma função, *`lambdaAuthRole`*, que autorize o Lambda a usar essas permissões. Autorize ações no seu cluster Amazon MSK usando ações do IAM do `kafka-cluster`. Em seguida, autorize o Lambda a realizar as ações do Amazon MSK `kafka` e do Amazon EC2 necessárias para descobrir e se conectar ao seu cluster do Amazon MSK, bem como as ações do CloudWatch para que o Lambda possa registrar em log o que fez.

Para descrever os requisitos de autenticação para que o Lambda leia a partir do Amazon MSK

1. Escreva um documento de política do IAM (um documento JSON), *`clusterAuthPolicy`*, que permita que o Lambda leia seu tópico do Kafka no seu cluster do Amazon MSK usando seu grupo de consumidores do Kafka. O Lambda exige que um grupo de consumidores do Kafka seja definido durante a leitura.

   Altere o modelo a seguir para se alinhar aos seus pré-requisitos:

```
{
    "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"
            ]
        }
    ]
}
```

Para obter mais informações, consulte the section called “Autenticação baseada em função do IAM”: Ao redigir sua política:

- Em *região* e *ID da conta,* forneça aqueles que hospedam seu cluster do Amazon MSK.
- Em *mskClusterName*, forneça o nome do seu cluster do Amazon MSK.
- Em *cluster-uuid*, forneça o UUID no ARN para seu cluster do Amazon MSK.
- Em *mskTopicName*, forneça o nome do seu tópico do Kafka.
- Em *mskGroupName*, forneça o nome do seu grupo de consumidores do Kafka.

2. Identifique as permissões do Amazon MSK, do Amazon EC2 e do CloudWatch necessárias para que o Lambda descubra e conecte seu cluster Amazon MSK e registre esses eventos no log.

   A política gerenciada `AWSLambdaMSKExecutionRole` define de forma permissiva as permissões necessárias. Use-a nas etapas a seguir.

   Em um ambiente de produção, avalie `AWSLambdaMSKExecutionRole` para restringir sua política de perfil de execução com base no princípio do privilégio mínimo e, em seguida, escreva uma política para seu perfil que substitua essa política gerenciada.

Para obter detalhes sobre a linguagem de políticas do IAM, consulte a Documentação do IAM.

Agora que você escreveu seu documento de política, crie uma política do IAM para poder anexá-la ao seu perfil. É possível fazer isso usando o console com o procedimento a seguir.

Para criar uma política do IAM a partir do seu documento de política

1. Faça login no AWS Management Console e abra o console do IAM em https://console.aws.amazon.com/iam/.
2. No painel de navegação à esquerda, escolha Políticas.
3. Escolha Criar política.
4. Na seção Editor de políticas, escolha a opção JSON.
5. Cole *clusterAuthPolicy*.
6. Quando terminar de adicionar as permissões à política, escolha Avançar.
7. Na página Revisar e criar, digite um nome de política e uma descrição (opcional) para a política que você está criando. Revise Permissões definidas nessa política para ver as permissões que são concedidas pela política.
8. Escolha Criar política para salvar sua nova política.

Para obter mais informações, consulte Criar políticas do IAM na documentação do IAM.

Agora que você tem as políticas apropriadas do IAM, crie uma perfil e anexe-o a ela. É possível fazer isso usando o console com o procedimento a seguir.

Como criar uma função de execução no console do IAM

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione Create role (Criar função).
3. Em Tipo de entidade confiável, selecione Serviço da AWS.
4. Em Use case (Caso de uso), escolha Lambda.
5. Escolha Próximo.
6. Selecione as políticas a seguir:
   - *clusterAuthPolicy*
   - `AWSLambdaMSKExecutionRole`
7. Escolha Próximo.
8. Em Nome do perfil, insira *lambdaAuthRole* e, em seguida, escolha Criar perfil.

Para ter mais informações, consulte the section called “Perfil de execução (permissões para funções acessarem outros recursos)”.

Crie uma função do Lambda para ler do seu tópico do Amazon MSK

Crie uma função do Lambda configurada para usar seu perfil do IAM. Você pode criar sua função do Lambda usando o console.

Para criar uma função do Lambda usando sua configuração de autenticação

1. Abra o console do Lambda e selecione Criar função no cabeçalho.
2. Selecione Criar do zero.
3. Em Nome da função, forneça um nome apropriado de sua escolha.
4. Em Runtime, escolha a versão Mais recente com suporte do `Node.js` para usar o código fornecido neste tutorial.
5. Escolha Alterar perfil de execução padrão.
6. Selecione Usar perfil existente.
7. Em Perfil existente, selecione *`lambdaAuthRole`*.

Em um ambiente de produção, geralmente é necessário adicionar mais políticas ao perfil de execução da sua função do Lambda para processar de forma significativa seus eventos do Amazon MSK. Para obter mais informações sobre como adicionar políticas ao seu perfil, consulte Adicionar ou remover permissões de identidade na documentação do IAM.

Criar um mapeamento da origem do evento para sua função do Lambda

Seu mapeamento de origem de eventos do Amazon MSK fornece ao serviço do Lambda as informações necessárias para invocar seu Lambda quando eventos apropriados do Amazon MSK ocorrem. Você pode criar um mapeamento do Amazon MSK usando o console. Crie um acionador do Lambda e, em seguida, o mapeamento de origem de eventos será configurado automaticamente.

Para criar um acionador do Lambda (e mapeamento de origem de eventos)

1. Navegue até a página de visão geral da sua função do Lambda.
2. Na seção de visão geral da função, selecione Adicionar acionador no canto inferior esquerdo.
3. No menu suspenso Selecionar uma origem, selecione Amazon MSK.
4. Não defina a autenticação.
5. Em Cluster do MSK, selecione o nome do seu cluster.
6. Em Tamanho do lote, insira 1. Essa etapa facilita o teste desse recurso, mas não é um valor ideal em ambientes de produção.

7. Em Nome do tópico, forneça o nome do seu tópico do Kafka.
8. Para ID do grupo de consumidores, forneça o ID do seu grupo de consumidores do Kafka.

Atualize a função do Lambda para ler seus dados de streaming

O Lambda fornece informações sobre eventos do Kafka por meio do parâmetro de método de evento. Para obter um exemplo de estrutura de um evento do Amazon MSK, consulte the section called “ Evento de exemplo”. Depois de entender como interpretar os eventos do Amazon MSK encaminhados pelo Lambda, você pode alterar o código da função do Lambda para usar as informações fornecidas por eles.

Forneça o seguinte código à sua função do Lambda para registrar em log o conteúdo de um evento do Amazon MSK do Lambda para fins de teste:

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)
      })
    }
}
```

Você pode fornecer código de função para o Lambda usando o console.

Para atualizar o códig da função do Lambda

1. Navegue até a página de visão geral da sua função do Lambda.
2. Escolha a guia Código.
3. Insira o código fornecido no IDE da Origem do código.
4. Na barra de navegação de Origem do código, escolha Implantar.

Teste sua função do Lambda para verificar se ela está conectada ao seu tópico do Amazon MSK

Agora você pode verificar se seu Lambda está sendo invocado pela origem do evento inspecionando os logs de eventos do CloudWatch.

Para verificar se sua função do Lambda está sendo invocada

1. Use seu host de administração do Kafka para gerar eventos do Kafka usando a CLI do `kafka-console-producer`. Para obter mais informações, consulte Escrever alguns eventos no tópico na documentação do Kafka. Envie eventos suficientes para preencher o lote definido pelo tamanho do lote para seu mapeamento de origem de eventos definido na etapa anterior, caso contrário, o Lambda aguardará a invocação de mais informações.
2. Se sua função for executada, o Lambda escreverá o que aconteceu com o CloudWatch. No console, navegue até a página de detalhes da função do Lambda.
3. Selecione a guia Configuration (Configuração).
4. Na barra lateral, selecione Ferramentas de monitoramento e operações.
5. Identifique o Grupo de logs do CloudWatch em Configuração de log. O grupo de logs deve começar com `/aws/lambda`. Escolha o link para o grupo de logs.
6. No console do CloudWatch, inspecione os Eventos de logs em busca dos eventos de log que o Lambda enviou para o fluxo de logs. Identifique se há eventos de log contendo a mensagem do seu evento do Kafka, como na imagem a seguir. Se houver, você conectou com sucesso uma função do Lambda ao Amazon MSK com um mapeamento de origem de eventos do Lambda.

Evento de exemplo

O Lambda envia o lote de mensagens no parâmetro de evento quando ele chama sua função. O payload do evento contém uma matriz de mensagens. Cada item de array contém detalhes do tópico do Amazon MSK e do identificador de partição, juntamente com um carimbo de data/hora e uma mensagem codificada em 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
                  ]
               }
            ]
         }
      ]
   }
}
```

Autenticação de cluster do MSK

O Lambda precisa de permissão para acessar o cluster do Amazon MSK, recuperar registros e executar outras tarefas. O Amazon MSK oferece suporte a várias opções para controlar o acesso do cliente ao cluster do MSK.

Opções de acesso ao cluster

- Acesso não autenticado
- Autenticação SASL/SCRAM
- Autenticação baseada em função do IAM
- Autenticação TLS mútua
- Configurar o segredo de mTLS
- Como o Lambda escolhe um operador de bootstrap

Acesso não autenticado

Se nenhum cliente acessar o cluster pela Internet, você poderá usar o acesso não autenticado.

Autenticação SASL/SCRAM

O Amazon MSK é compatível com autenticação Simple Authentication e Security Layer/Salted Challenge Response Authentication Mechanism (SASL/SCRAM) com criptografia Transport Layer Security (TLS). Para que o Lambda se conecte ao cluster, é necessário armazenar as credenciais de autenticação (nome de usuário e senha) em um segredo do AWS Secrets Manager.

Para obter mais informações sobre o uso do Secrets Manager, consulte Autenticação de nome de usuário e senha com o AWS Secrets Manager, no Guia do Desenvolvedor do Amazon Managed Streaming para Apache Kafka.

O Amazon MSK não oferece suporte a autenticação SASL/PLAIN.

Autenticação baseada em função do IAM

Use o IAM para autenticar a identidade dos clientes que se conectam ao cluster do MSK. Se a autenticação do IAM estiver ativa no cluster do MSK, e você não fornecer um segredo para autenticação, o Lambda usará automaticamente a autenticação do IAM por padrão. Para criar e implantar políticas baseadas em perfil ou usuário, use o console ou a API do IAM. Para obter mais

informações, consulte IAM access control (Controle de acesso do IAM) no Guia do desenvolvedor do Amazon Managed Streaming for Apache Kafka.

Para permitir que o Lambda se conecte ao cluster do MSK, leia registros e execute outras ações necessárias, adicione as permissões a seguir à função de execução da função.

```
{
    "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"
            ]
        }
    ]
}
```

É possível definir o escopo dessas permissões para abranger clusters, tópicos e grupos específicos. Para obter mais informações, consulte Amazon MSK Kafka actions (Ações do Amazon MSK Kafka) no Guia do desenvolvedor Amazon Managed Streaming for Apache Kafka.

Autenticação TLS mútua

O TLS mútuo (mTLS) fornece autenticação bidirecional entre o cliente e o servidor. O cliente envia um certificado ao servidor para que o servidor verifique o cliente, e o servidor envia um certificado ao cliente para que o cliente verifique o servidor.

No Amazon MSK, o Lambda atua como cliente. Configure um certificado de cliente (como um segredo no Secrets Manager) para autenticar o Lambda com os agentes no cluster do MSK. O

certificado do cliente deve ser assinado por uma autoridade de certificação no armazenamento de confiança do servidor. O cluster do MSK envia um certificado de servidor ao Lambda para autenticar os agentes com o Lambda. O certificado do servidor deve ser assinado por uma autoridade de certificação (CA) no armazenamento de confiança da AWS.

Para obter instruções sobre como gerar um certificado de cliente, consulte Introducing mutual TLS authentication for Amazon MSK as an event source (Introdução à autenticação TLS mútua do Amazon MSK como fonte de eventos).

O Amazon MSK não oferece suporte a certificados de servidor autoassinados, pois todos os agentes no Amazon MSK usam certificados públicos assinados por CAs do Amazon Trust Services, nos quais o Lambda confia por padrão.

Para obter mais informações sobre o mTLS para o Amazon MSK, consulte Mutual TLS Authentication (Autenticação TLS mútua) no Guia do desenvolvedor Amazon Managed Streaming for Apache Kafka.

Configurar o segredo de mTLS

O segredo CLIENT_CERTIFICATE_TLS_AUTH requer um campo de certificado e um campo de chave privada. Para uma chave privada criptografada, o segredo requer uma senha de chave privada. Tanto o certificado como a chave privada devem estar no formato PEM.

> **Note**
>
> O Lambda oferece suporte a algoritmos de criptografia de chave privada PBES1 (mas não PBES2).

O campo certificate (certificado) deve conter uma lista de certificados, começando pelo certificado do cliente, seguido por quaisquer certificados intermediários e terminando com o certificado raiz. Cada certificado deve iniciar em uma nova linha com a seguinte estrutura:

```
-----BEGIN CERTIFICATE-----
        <certificate contents>
-----END CERTIFICATE-----
```

O Secrets Manager oferece suporte a segredos de até 65.536 bytes, que é espaço suficiente para cadeias de certificados longas.

A chave privada deve estar no formato PKCS #8, com a seguinte estrutura:

```
-----BEGIN PRIVATE KEY-----
        <private key contents>
-----END PRIVATE KEY-----
```

Para uma chave privada criptografada, use a seguinte estrutura:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
         <private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

O exemplo a seguir exibe o conteúdo de um segredo para autenticação mTLS usando uma chave privada criptografada. Para uma chave privada criptografada, inclua a senha da chave privada no segredo.

```
{
 "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-----"
}
```

Como o Lambda escolhe um operador de bootstrap

O Lambda escolhe um operador de bootstrap com base nos métodos de autenticação disponíveis no cluster e dependendo de você fornecer um segredo para autenticação. Se você fornecer um segredo

para mTLS ou SASL/SCRAM, o Lambda escolherá esse método de autenticação automaticamente. Se você não fornecer um segredo, o Lambda selecionará o método de autenticação mais forte que estiver ativo no seu cluster. Veja a seguir a ordem de prioridade na qual o Lambda seleciona um operador, da autenticação mais forte para a mais fraca:

- mTLs (segredo fornecido para mTLs)
- SASL/SCRAM (segredo fornecido para SASL/SCRAM)
- SASL IAM (nenhum segredo fornecido, e a autenticação do IAM está ativa)
- TLS não autenticado (nenhum segredo fornecido, e a autenticação do IAM não está ativa)
- Texto simples (nenhum segredo fornecido, e a autenticação do IAM e o TLS não autenticado não estão ativos)

> **Note**
>
> Se o Lambda não conseguir se conectar ao tipo de operador mais seguro, o Lambda não tentará se conectar a um tipo de operador diferente (mais fraco). Se quiser que o Lambda escolha um tipo de operador mais fraco, desative todos os métodos de autenticação mais fortes no seu cluster.

Gerenciar acesso e permissões de API

Além de acessar o cluster do Amazon MSK, a função Lambda precisa de permissões para executar várias ações de API do Amazon MSK. Adicione essas permissões à função de execução da função. Se os usuários precisarem acessar qualquer ação da API do Amazon MSK, adicione as permissões necessárias à política de identidade para o usuário ou o perfil.

É possível adicionar manualmente cada uma das permissões a seguir ao perfil de execução. Ou então, você pode anexar a política gerenciada pela AWS AWSLambdaMSKExecutionRole ao perfil de execução. A política `AWSLambdaMSKExecutionRole` contém todas as ações de API e permissões da VPC necessárias listadas abaixo.

Permissões necessárias da função de execução da função do Lambda

Para criar e armazenar logs em um grupo de logs do Amazon CloudWatch Logs, sua função do Lambda deve ter as seguintes permissões na função de execução:

- logs:CreateLogGroup

- logs:CreateLogStream
- logs:PutLogEvents

Para que o Lambda acesse o cluster do Amazon por você, sua função do Lambda deve ter as seguintes permissões no perfil de execução:

- kafka:DescribeCluster
- kafka:DescribeClusterV2
- kafka:GetBootstrapBrokers
- kafka:DescribeVpcConnection: só é necessário para mapeamentos de origem de eventos entre contas.
- kafka:ListVpcConnections: não é obrigatório no perfil de execução, mas é obrigatório para uma entidade principal do IAM que estiver criando um mapeamento da origem de eventos entre contas.

Você só precisa adicionar `kafka:DescribeCluster` ou `kafka:DescribeClusterV2`. Para clusters do MSK provisionados, ambas as permissões funcionam. Para clusters do MSK com tecnologia sem servidor, é necessário usar `kafka:DescribeClusterV2`.

> **Note**
>
> O Lambda planeja remover posteriormente a permissão `kafka:DescribeCluster` desta política gerenciada `AWSLambdaMSKExecutionRole` associada. Se usar essa política, você deve migrar aplicações usando `kafka:DescribeCluster` para usar `kafka:DescribeClusterV2` no lugar.

Permissões da VPC

Se somente usuários dentro de uma VPC puderem acessar seu cluster do Amazon MSK, a função do Lambda deverá ter permissão para acessar seus recursos da Amazon VPC. Esses recursos incluem sua VPC, sub-redes, security groups e interfaces de rede. Para acessar esses recursos, o perfil de execução da função precisa ter as permissões a seguir. Essas permissões estão incluídas na política gerenciada pela AWS AWSLambdaMSKExecutionRole.

- ec2:CreateNetworkInterface
- ec2:DescribeNetworkInterfaces

- ec2:DescribeVpcs
- ec2:DeleteNetworkInterface
- ec2:DescribeSubnets
- ec2:DescribeSecurityGroups

Permissões de função do Lambda opcionais

Sua função Lambda também pode precisar dessas permissões para:

- Acesse o segredo SCRAM, se estiver usando a autenticação SASL/SCRAM.
- Descreva o segredo do Secrets Manager.
- Acessar a chave gerenciada pelo cliente AWS Key Management Service (AWS KMS)
- Enviar registros de invocações com falha para um destino

Secrets Manager e permissões do AWS KMS

Conforme o tipo de controle de acesso que você está configurando para os agentes do Amazon MSK, a função do Lambda pode precisar de permissão para acessar seu segredo do SCRAM (se usar autenticação SASL/SCRAM) ou o segredo do Secrets Manager para descriptografar sua chave do AWS KMS gerenciada pelo cliente. Para acessar esses recursos, a função de execução da função precisa ter as seguintes permissões:

- kafka:ListScramSecrets
- secretsmanager:GetSecretValue
- kms:Decrypt

Adicionar permissões à sua função de execução

Siga estas etapas para adicionar a política gerenciada pela AWS AWSLambdaMSKExecutionRole ao perfil de execução usando o console do IAM.

Para adicionar uma política gerenciada da AWS

1. Abra a página Policies (Políticas) do console do IAM.
2. Na caixa de pesquisa, insira o nome da política (`AWSLambdaMSKExecutionRole`).

3. Selecione a política na lista e, em seguida, escolha Policy actions (Ações de política), Attach (Associar).
4. NoAnexar política, selecione sua função de execução na lista e escolhaAnexar política.

Conceder acesso aos usuários com uma política do IAM

Por padrão, usuários e perfis não têm permissão para executar operações de API do Amazon MSK. Para conceder acesso a usuários em sua organização ou conta, é possível adicionar ou atualizar uma política baseada em identidades. Para obter mais informações, consulte Exemplos de políticas baseadas em identidade do Amazon MSK no Guia do desenvolvedor do Amazon Managed Streaming for Apache Kafka.

Erros de autenticação e autorização

Se alguma das permissões necessárias para consumir dados do cluster do Amazon MSK estiver ausente, o Lambda exibirá uma mensagem de erro no mapeamento da fonte de eventos em LastProcessingResult.

Mensagens de erro

- O cluster falhou ao autorizar o Lambda
- SASL authentication failed (Falha na autenticação SASL)
- Server failed to authenticate Lambda (Falha ao autenticar o Lambda no servidor)
- Provided certificate or private key is invalid (O certificado ou a chave privada fornecida é inválida)

O cluster falhou ao autorizar o Lambda

Para SASL/SCRAM ou mTLS, esse erro indica que o usuário fornecido não tem todas estas permissões da lista de controle de acesso (ACL) do Kafka necessárias:

- Cluster DescribeConfigs
- Descrever grupo
- Ler grupo
- Descrever tópico
- Ler tópico

Para controle de acesso do IAM, a função de execução da função não tem uma ou mais permissões necessárias para acessar o grupo ou tópico. Confira a lista de permissões necessárias em the section called “Autenticação baseada em função do IAM”.

Ao criar ACLs do Kafka ou uma política do IAM com as permissões necessárias do cluster do Kafka, é necessário especificar o tópico e o grupo como recursos. O nome do tópico deve corresponder ao tópico no mapeamento da fonte de eventos. O nome do grupo deve corresponder ao UUID do mapeamento da fonte de eventos.

Depois de adicionar as permissões necessárias à função de execução, poderá levar vários minutos para que as alterações entrem em vigor.

SASL authentication failed (Falha na autenticação SASL)

Para SASL/SCRAM, esse erro indica que o nome de usuário e a senha fornecidos são inválidos.

Para controle de acesso do IAM, a função de execução não tem a permissão de `kafka-cluster:Connect` para o cluster do MSK. Adicione essa permissão à função e especifique o nome do recurso da Amazon (ARN) do cluster como recurso.

É possível que esse erro ocorra de modo intermitente. O cluster rejeitará conexões depois que o número de conexões de TCP exceder a cota de serviço do Amazon MSK. O Lambda recua e tenta novamente até que uma conexão seja bem-sucedida. Depois que o Lambda se conecta ao cluster e sonda por registros, o último resultado de processamento é alterado para `OK`.

Server failed to authenticate Lambda (Falha ao autenticar o Lambda no servidor)

Esse erro indica que não foi possível autenticar os agentes do Kafka do Amazon MSK com o Lambda. Esse erro pode ocorrer por estes motivos:

- Você não forneceu um certificado de cliente para autenticação mTLS.
- Você forneceu um certificado de cliente, mas os agentes não estão configurados para usar mTLS.
- Um certificado de cliente não é confiável para os agentes.

Provided certificate or private key is invalid (O certificado ou a chave privada fornecida é inválida)

Esse erro indica que o consumidor do Amazon MSK não conseguiu usar o certificado ou a chave privada fornecida. Verifique se o certificado e a chave usam o formato PEM e que a criptografia de chave privada usa um algoritmo PBES1.

Configuração de rede

Para que o Lambda use seu cluster do Kafka como uma origem de eventos, ele precisa de acesso à Amazon VPC na qual o cluster reside. Recomendamos implantar endpoints da VPC do AWS PrivateLink para o Lambda acessar sua VPC. Implante endpoints para Lambda e AWS Security Token Service (AWS STS). Se o agente usar autenticação, implante também um endpoint da VPC para o Secrets Manager. Se você tiver configurado um destino em caso de falha, implante também um endpoint da VPC para o serviço de destino.

Alternativamente, certifique-se de que a VPC associada ao cluster do Kafka inclua um gateway NAT por sub-rede pública. Para ter mais informações, consulte the section called “Acesso à Internet para funções da VPC”.

Se você usa endpoints da VPC, também deve configurá-los para habilitar nomes DNS privados.

Quando você cria um mapeamento da origem do evento para um cluster do MSK, o Lambda verifica se as interfaces de rede elástica (ENIs) já estão presentes nas sub-redes e nos grupos de segurança da VPC do seu cluster. Se o Lambda encontrar ENIs existentes, ele tentará reutilizá-las. Caso contrário, o Lambda criará novas ENIs para se conectar à origem do evento e invocar sua função.

> **Note**
>
> As funções do Lambda sempre são executadas em VPCs de propriedade do serviço Lambda. Essas VPCs recebem manutenção automática do serviço e não são visíveis para os clientes. Você também pode conectar sua função a uma Amazon VPC. Em ambos os casos, a configuração de VPC da sua função não afetará o mapeamento da origem do evento. Somente a configuração da VPC da origem de eventos determina o modo de conexão do Lambda à sua origem de eventos.

Sua configuração da Amazon VPC pode ser detectada pela API do Amazon MSK. Não é necessário defini-la durante a configuração usando o comando create-event-source-mapping.

Para obter mais informações sobre como configurar a rede, consulteConfigurar o AWS Lambda com um cluster do Apache Kafka em uma VPC no blog AWS Compute.

Regras de grupos de segurança da VPC

Configure com as seguintes regras (no mínimo) os grupos de segurança da Amazon VPC que contêm seu cluster:

- Regras de entrada: permitir todo o tráfego na porta do agente do Amazon MSK (9092 para texto sem formatação, 9094 para TLS, 9096 para SASL, 9098 para IAM) para os grupos de segurança especificados para a fonte de eventos.
- Regras de saída: permitir todo o tráfego na porta 443 para todos os destinos. Permitir todo o tráfego na porta do agente do Amazon MSK (9092 para texto sem formatação, 9094 para TLS, 9096 para SASL, 9098 para IAM) para os grupos de segurança especificados para a fonte de eventos.
- Se você estiver usando endpoints da VPC em vez de um gateway NAT, os grupos de segurança associados aos endpoints da VPC deverão permitir todo o tráfego de entrada na porta 443 dos grupos de segurança da fonte de eventos.

Trabalhar com endpoints da VPC

Quando você usa endpoints da VPC, as chamadas de API para invocar sua função são roteadas por esses endpoints usando as ENIs. A entidade principal do serviço Lambda precisa chamar `sts:AssumeRole` e `lambda:InvokeFunction` para quaisquer perfis e funções que usem essas ENIs.

Por padrão, os endpoints da VPC têm políticas do IAM que são abertas. A prática recomendada é restringir essas políticas a fim de permitir que somente entidades principais específicas executem as ações necessárias usando esse endpoint. Para garantir que seu mapeamento da origem do evento seja capaz de invocar sua função do Lambda, a política de endpoint da VPC deve permitir que a entidade principal do serviço Lambda chame `sts:AssumeRole` e `lambda:InvokeFunction`. A restrição de suas políticas de endpoint da VPC para permitir apenas chamadas de API originadas em sua organização impedirá o funcionamento adequado do mapeamento da origem do evento.

O seguinte exemplo de políticas de endpoint da VPC mostra como conceder o acesso necessário à entidade principal do serviço Lambda para os endpoints do AWS STS e Lambda.

Example Política de endpoint da VPC: endpoint do AWS STS

```
{
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
```

```
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Example Política de endpoint da VPC: endpoint do Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Se o seu agente do Kafka usar autenticação, você também poderá restringir a política de endpoint da VPC para o endpoint do Secrets Manager. Para chamar a API do Secrets Manager, o Lambda usa seu perfil de função, e não a entidade principal de serviço do Lambda. O exemplo a seguir mostra uma política de endpoint do Secrets Manager.

Example Política de endpoint da VPC: endpoint do Secrets Manager.

```
{
    "Statement": [
        {
            "Action": "secretsmanager:GetSecretValue",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "customer_function_execution_role_arn"
                ]
            },
```

```
            "Resource": "customer_secret_arn"
        }
    ]
}
```

Se você tiver um destino em caso de falha configurado, o Lambda também usará o perfil de sua função para chamar `s3:PutObject`, `sns:Publish` ou `sqs:sendMessage` usando as ENIs gerenciadas pelo Lambda.

Adicionar o Amazon MSK como uma origem de evento

Para criar um mapeamento de origem de evento, adicione o Amazon MSK como um acionador de função do Lambda usando o console do Lambda, um AWS SDK ou a AWS Command Line Interface (AWS CLI). Observe que, quando você adiciona o Amazon MSK como gatilho, o Lambda assume as configurações de VPC do cluster do Amazon MSK, e não as configurações de VPC da função do Lambda.

Esta seção descreve como criar um mapeamento de origens de eventos usando o console do Lambda e oAWS CLI.

Pré-requisitos

- Um cluster do Amazon MSK e um tópico do Kafka. Para obter mais informações, consulte Conceitos básicos do uso do Amazon MSK no Guia do desenvolvedor do Amazon Managed Streaming for Apache Kafka.
- Uma função de execução com permissão para acessar os recursos da AWS que seu cluster do MSK usa.

ID de grupo de consumidores personalizável

Ao configurar o Kafka como uma origem de eventos, você pode especificar um ID de grupo de consumidores. Esse ID de grupo de consumidores é um identificador existente para o grupo de consumidores do Kafka no qual você deseja que a função do Lambda ingresse. Você pode usar esse recurso para migrar facilmente qualquer configuração de processamento em andamento de registros do Kafka de outros consumidores para o Lambda.

Se você especificar um ID de grupo de consumidores e houver outros pesquisadores ativos dentro desse grupo de consumidores, o Kafka distribuirá mensagens entre todos os consumidores. Em outras palavras, o Lambda não receberá todas as mensagens para o tópico do Kafka. Se você quiser

que o Lambda gerencie todas as mensagens do tópico, desative todos os outros pesquisadores desse grupo de consumidores.

Além disso, se você especificar um ID de grupo de consumidores e o Kafka encontrar um grupo de consumidores válido já existente com o mesmo ID, o Lambda ignorará o parâmetro `StartingPosition` no mapeamento de origem de eventos. Em vez disso, o Lambda começará a processar registros de acordo com o deslocamento confirmado do grupo de consumidores. Se você especificar um ID de grupo de consumidores e o Kafka não conseguir encontrar um grupo de consumidores existente, o Lambda configurará a origem de eventos com a `StartingPosition` especificada.

O ID do grupo de consumidores que você especificar deverá ser exclusivo entre todas as origens de eventos do Kafka. Após criar um mapeamento de origem de eventos do Kafka com o ID do grupo de consumidores especificado, você não poderá atualizar esse valor.

Adicionar um gatilho do Amazon MSK (console)

Siga estas etapas para adicionar seu cluster do Amazon MSK e um tópico do Kafka como um acionador para sua função do Lambda.

Para adicionar um acionador do Amazon MSK à sua função do Lambda (console)

1. Abra a página Functions (Funções) no console do Lambda.
2. Escolha o nome da função do Lambda.
3. Em Visão geral da função, escolha Adicionar gatilho.
4. Em Trigger configuration (Configuração do acionador), faça o seguinte:
    a. Selecione o tipo de acionador MSK.
    b. Em Cluster do MSK, selecione seu cluster.
    c. Em Batch size (Tamanho do lote), insira o número máximo de mensagens a serem recebidas em um único lote.
    d. Em Batch window (Janela de lote), insira o tempo máximo em segundos usado pelo Lambda para coletar os registros antes de invocar a função.
    e. Em Topic name (Nome do tópico), insira um nome para o tópico do Kafka.
    f. (Opcional) em Consumer group ID (ID do grupo de consumidores), insira o ID de um grupo de consumidores do Kafka no qual ingressar.
    g. (Opcional) Em Posição inicial, escolha Mais recente para começar a realizar a leitura do fluxo a partir do registro mais recente, Horizonte de corte para começar no registro mais

antigo disponível ou No carimbo de data e hora para especificar um carimbo de data e hora para começar a realizar a leitura.

h. (Opcional) Em Authentication (Autenticação), escolha a chave secreta para autenticar com os agentes no cluster do MSK.

i. Para criar o gatilho em um estado desativado para teste (recomendado), desmarqueAtivar gatilho. Ou, para habilitar o gatilho imediatamente, selecioneAtivar gatilho.

5. Para criar o acionador, selecione Add (Adicionar).

Adicionando um gatilho do Amazon MSK (AWS CLI)

Use os comandos de exemplo a seguir da AWS CLI para criar e visualizar um acionador do Amazon MSK para sua função do Lambda.

Criar um trigger usando a AWS CLI

Example : criar um mapeamento da origem de eventos para o cluster que usa autenticação do IAM

O exemplo a seguir usa o comando `create-event-source-mapping` da AWS CLI para mapear uma função do Lambda chamada `my-kafka-function` em um tópico do Kafka chamado `AWSKafkaTopic`. A posição inicial do tópico está definida como `LATEST`. Quando o cluster usa a autenticação baseada em perfil do IAM, você não precisa de um objeto SourceAccessConfiguration. Exemplo:

```
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 : criar um mapeamento da origem de eventos para o cluster que usa autenticação SASL/SCRAM

Se o cluster usar autenticação SASL/SCRAM, você precisa incluir um objeto SourceAccessConfiguration que especifique `SASL_SCRAM_512_AUTH` e um ARN secreto do 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 : criar um mapeamento da origem de eventos para o cluster que usa autenticação mTLS

Se o cluster usar autenticação mTLS, você precisa incluir um objeto SourceAccessConfiguration que especifique `CLIENT_CERTIFICATE_TLS_AUTH` e um ARN secreto do 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"}]'
```

Para obter mais informações, consulte o .CreateEventSourceMappingDocumentação de referência da API.

Visualizar o status usando a AWS CLI

O exemplo a seguir usa o comando `get-event-source-mapping` da AWS CLI para descrever o status do mapeamento de fontes de eventos que você criou.

```
aws lambda get-event-source-mapping \
  --uuid 6d9bce8e-836b-442c-8070-74e77903c815
```

Criar mapeamentos da origem do evento entre contas

Você pode usar a conectividade privada com várias VPCs para conectar uma função do Lambda a um cluster DO MSK provisionado em outra Conta da AWS. A conectividade com várias VPCs usa AWS PrivateLink, que mantém todo o tráfego na rede da AWS.

> **Note**
>
> Você não pode criar mapeamentos da origem do evento entre contas para clusters do MSK sem servidor.

Para criar um mapeamento da origem de eventos entre contas, primeiro você precisa configurar a conectividade com várias VPCs para o cluster do MSK. Ao criar o mapeamento da origem de eventos, use o ARN da conexão VPC gerenciada, em vez do ARN do cluster, conforme mostrado nos exemplos a seguir. A operação CreateEventSourceMapping também difere dependendo do tipo de autenticação usado pelo cluster do MSK.

Example : criar um mapeamento da origem de eventos entre contas para o cluster que usa autenticação do IAM

Quando o cluster usa a autenticação baseada em perfil do IAM, você não precisa de um objeto SourceAccessConfiguration. Exemplo:

```
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 : criar um mapeamento da origem de eventos entre contas para o cluster que usa autenticação SASL/SCRAM

Se o cluster usar autenticação SASL/SCRAM, você precisa incluir um objeto SourceAccessConfiguration que especifique `SASL_SCRAM_512_AUTH` e um ARN secreto do Secrets Manager.

Há duas maneiras de usar segredos para mapeamentos da origem do evento entre contas do Amazon MSK com autenticação SASL/SCRAM:

- Crie um segredo na conta da função do Lambda e sincronize-o com o segredo do cluster. Crie uma rotação para manter os dois segredos sincronizados. Essa opção permite controlar o segredo pela conta da função.
- Use o segredo associado ao cluster do MSK. Esse segredo deve permitir o acesso entre contas à conta da função do Lambda. Para obter mais informações, consulte Permissões para segredos do AWS Secrets Manager para usuários em uma conta diferente.

```
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 : criar um mapeamento da origem de eventos entre contas para o cluster que usa autenticação mTLS

Se o cluster usar autenticação mTLS, você precisa incluir um objeto SourceAccessConfiguration que especifique `CLIENT_CERTIFICATE_TLS_AUTH` e um ARN secreto do Secrets Manager. O segredo pode ser armazenado na conta do cluster ou na conta da função do 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"}]'
```

Destinos em caso de falha

Para reter registros de invocações de mapeamento da origem do evento com falha, adicione um destino ao mapeamento da origem de eventos da função. Cada registro enviado ao destino é um documento JSON com metadados sobre a invocação que falhou. É possível configurar qualquer tópico do Amazon SNS, fila do Amazon SQS ou bucket do S3 como destino. Sua função de execução deve ter permissões para o destino:

- Para destinos SQS: sqs:SendMessage
- Para destinos SNS: sns:Publish
- Para destinos de bucket do S3: s3:PutObject e s3:ListBuckets

Além disso, se você configurou uma chave do KMS no seu destino, o Lambda precisará das seguintes permissões, dependendo do tipo de destino:

- Se você habilitou a criptografia com sua própria chave do KMS para um destino do S3, kms:GenerateDataKey é necessário. Se a chave do KMS e o destino do bucket do S3 estiverem

em uma conta diferente da função do Lambda e do perfil de execução, configure a chave do KMS para confiar no perfil de execução para permitir kms:GenerateDataKey.

- Se você habilitou a criptografia com sua própria chave do KMS para um destino do SQS, kms:Decrypt e kms:GenerateDataKey são necessários. Se a chave do KMS e o destino da fila do SQS estiverem em uma conta diferente da função do Lambda e do perfil de execução, configure a chave do KMS para confiar no perfil de execução para permitir kms:Decrypt, kms:GenerateDataKey, kms:DescribeKey e kms:ReEncrypt.
- Se você habilitou a criptografia com sua própria chave do KMS para um destino do SNS, kms:Decrypt e kms:GenerateDataKey são necessários. Se a chave do KMS e o destino do tópico do SNS estiverem em uma conta diferente da função do Lambda e do perfil de execução, configure a chave do KMS para confiar no perfil de execução para permitir kms:Decrypt, kms:GenerateDataKey, kms:DescribeKey e kms:ReEncrypt.

Para configurar um destino em caso de falha usando o console, siga estas etapas:

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Em Function overview (Visão geral da função), escolha Add destination (Adicionar destino).
4. Em Origem, escolha Invocação do mapeamento da origem do evento.
5. Em Mapeamento da origem do evento, escolha uma origem de eventos configurada para essa função.
6. Em Condição, selecione Em caso de falha. Para invocações de mapeamento da origem de eventos, essa é a única condição aceita.
7. Em Tipo de destino, escolha o tipo de destino para o qual o Lambda envia registros de invocação.
8. Em Destination (Destino), escolha um recurso.
9. Escolha Salvar.

Você também pode configurar um destino em caso de falha usando a AWS CLI. Por exemplo, o seguinte comando create-event-source-mapping adiciona um mapeamento de origem de eventos com um destino SQS em caso de falha 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"}}'
```

O seguinte comando update-event-source-mapping adiciona um destino S3 em caso de falha à origem de eventos associada à entrada `uuid`:

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": "arn:aws:s3:::dest-bucket"}}'
```

Para remover um destino, forneça uma string vazia como argumento para o parâmetro `destination-config`:

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

Exemplo de registro de invocação do SNS e do SQS

O exemplo a seguir mostra o que é enviado pelo Lambda para o destino de tópico do SNS ou de fila do SQS em caso de falha na invocação da origem de eventos do Kafka. Cada uma das chaves abaixo de `recordsInfo` contém o tópico e a partição do Kafka separados por um hífen. Por exemplo, para a chave `"Topic-0"`, `Topic` é o tópico do Kafka, e `0` é a partição. Para cada tópico e partição, você pode usar os dados de desvios e do carimbo de data/hora para encontrar os registros de invocação originais.

```
{
    "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",
            }
        }
    }
}
```

Exemplo de registro de invocação de destino S3

Para destinos do S3, o Lambda envia todo o registro da invocação junto com os metadados para o destino. O exemplo a seguir mostra o que é enviado pelo Lambda para o destino de bucket do S3 em caso de falha na invocação da origem de eventos do Kafka. Além de todos os campos do exemplo anterior para destinos do SQS e do SNS, o campo `payload` contém o registro de invocação original como uma string JSON com 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**
>
> Recomendamos habilitar o versionamento do S3 no bucket de destino.

Auto scaling da origem do evento do Amazon MSK

Quando você cria inicialmente uma fonte de eventos do Amazon MSK, o Lambda aloca um consumidor para processar todas as partições no tópico do Kafka. Cada consumidor conta com vários processadores em execução em paralelo para lidar com um aumento de workloads. O Lambda aumenta ou reduz a escala na vertical automaticamente do número de consumidores com base na workload. Para preservar a ordenação de mensagens em cada partição, o número máximo de consumidores é um consumidor por partição no tópico.

A cada um minuto, o Lambda avalia o atraso de compensação do consumidor de todas as partições do tópico. Se o atraso for muito alto, a partição está recebendo mensagens mais rápido do que o Lambda pode processá-las. Se necessário, o Lambda adiciona ou remove os consumidores do tópico. O processo de escalabilidade de adicionar ou remover consumidores ocorre em até três minutos após a avaliação.

Se a função do Lambda de destino sofrer um controle de utilização, o Lambda reduzirá o número de consumidores. Essa ação reduz a workload na função, reduzindo o número de mensagens que os consumidores podem recuperar e enviar para a função.

Para monitorar o throughput do tópico do Kafka, veja a Métrica de atraso de deslocamento que o Lambda emite enquanto sua função processa registros.

Para verificar quantas invocações de função ocorrem em paralelo, você também pode monitorar ométricas de simultaneidadePara sua função.

Posições iniciais de sondagem e fluxo

Esteja ciente de que a sondagem do fluxo durante a criação e as atualizações do mapeamento da origem do evento é, finalmente, consistente.

- Durante a criação do mapeamento da origem do evento, pode levar alguns minutos para a sondagem de eventos do fluxo iniciar.
- Durante as atualizações do mapeamento da origem do evento, pode levar alguns minutos para interromper e reiniciar a sondagem de eventos do fluxo.

Esse comportamento significa que, se você especificar `LATEST` como posição inicial do fluxo, o mapeamento da origem do evento poderá perder eventos durante a criação ou as atualizações. Para garantir que nenhum evento seja perdido, especifique a posição inicial do fluxo como `TRIM_HORIZON` ou `AT_TIMESTAMP`.

Métricas do Amazon CloudWatch

O Lambda emite a métrica `OffsetLag` enquanto sua função processa registros. O valor dessa métrica é a diferença de deslocamento entre o último registro gravado no tópico da origem de eventos do Kafka e o último registro que o grupo de consumidores da função processou. Você pode usar `OffsetLag` para estimar a latência entre o momento em que um registro é adicionado e o momento em que o grupo de consumidores o processa.

Uma tendência crescente em `OffsetLag` pode indicar problemas com pesquisadores no grupo de consumidores da função. Para ter mais informações, consulte Trabalhar com métricas de funções Lambda.

Parâmetros de configuração do Amazon MSK

Todos os tipos de origem de evento Lambda compartilham o mesmoCreateEventSourceMappingeUpdateEventSourceMappingOperações de API do. No entanto, apenas alguns dos parâmetros se aplicam ao Amazon MSK.

Parâmetros de origem de evento que se aplicam ao Amazon MSK

Parâmetro	Obrigatório	Padrão	Observações
AmazonManagedKafkaEventSourceConfig	N	Contém o campo ConsumerGroupId, que assume, por padrão, um valor exclusivo.	Pode definir apenas em Criar
BatchSize	N	100	Máximo: 10.000.
Habilitado	N	Habilitado	
EventSourceArn	Y		Pode definir apenas em Criar
FunctionName	Y		
FilterCriteria	N		Filtragem de eventos do Lambda

Parâmetro	Obrigatório	Padrão	Observações
MaximumBatchingWindowInSeconds	N	500 ms	Comportamento de lotes
SourceAccessConfigurations	N	Sem credenciais do	Credenciais de autenticação SASL/SCRAM ou CLIENT_CERTIFICATE_TLS_AUTH (MutualTLS) para a fonte do evento
StartingPosition	Y		AT_TIMESTAMP, TRIM_HORIZON, ou LATEST Pode definir apenas em Criar
StartingPositionTimestamp	N		Obrigatório se StartingPosition estiver definido como AT_TIMESTAMP
Tópicos	Y		Nome do tópico do Kafka Pode definir apenas em Criar

Usar o Lambda com o Apache Kafka autogerenciado

> **Note**
>
> Se você deseja enviar dados para um destino que não seja uma função do Lambda ou enriquecer os dados antes de enviá-los, consulte Amazon EventBridge Pipes (Pipes do Amazon EventBridge).

O Lambda suportaApache Kafkacomo umOrigem do evento. O Apache Kafka é uma plataforma de streaming de eventos de código aberto que suporta cargas de trabalho, como pipelines de dados e análises de streaming.

Você pode usar o serviço do Kafka gerenciado pela AWS Amazon Managed Streaming for Apache Kafka (Amazon MSK) ou um cluster do Kafka autogerenciado. Para obter detalhes sobre o uso do Lambda com o Amazon MSK, consulte Usar o Lambda com o Amazon MSK.

Este tópico descreve como usar o Lambda com um cluster Kafka autogerenciado. Na terminologia da AWS, um cluster autogerenciado inclui clusters do Kafka não hospedados pela AWS. Por exemplo, você pode hospedar seu cluster Kafka com um provedor de nuvem, como Confluent Cloud.

O Apache Kafka como uma fonte de eventos opera de forma semelhante ao uso do Amazon Simple Queue Service (Amazon SQS) ou do Amazon Kinesis. O Lambda pesquisa internamente por novas mensagens da origem do evento e, em seguida, chama de forma síncrona a função do Lambda de destino. O Lambda lê as mensagens em lotes e fornece estas para a sua função como uma carga de eventos. O tamanho máximo do lote é configurável. (O valor padrão é de 100 mensagens.)

> **Warning**
>
> Os mapeamentos da origem do evento do Lambda processam cada evento ao menos uma vez, podendo haver o processamento duplicado de registros. Para evitar possíveis problemas relacionados a eventos duplicados, é altamente recomendável tornar o código da função idempotente. Para saber mais, consulte Como tornar minha função do Lambda idempotente no Centro de Conhecimentos da AWS.

Para fontes de eventos baseadas em Kafka, o Lambda oferece suporte a parâmetros de controle de processamento, como janelas de lotes e tamanho do lote. Para ter mais informações, consulte Comportamento de lotes.

Para obter um exemplo de como usar o Kafka autogerenciado como uma fonte de eventos, consulteUsar o Apache Kafka auto-hospedado como uma fonte de eventos para o AWS Lambda no blog AWSCompute.

Tópicos

Evento de exemplo

O Lambda envia o lote de mensagens no parâmetro de evento quando ele invoca sua função do Lambda. O payload do evento contém uma matriz de mensagens. Cada item da matriz contém detalhes do tópico do Kafka e do identificador de partição do Kafka, juntamente com um carimbo de data/hora e uma mensagem codificada em 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
                  ]
               }
            ]
         }
      ]
   }
}
```

Autenticação de clusters do Kafka

O Lambda suporta vários métodos para autenticar com seu cluster Apache Kafka autogerenciado. Configure o cluster Kafka para utilizar um dos métodos de autenticação compatíveis. Para obter mais informações sobre a segurança do Kafka, consulte a seção Segurança da documentação do Kafka.

Acesso por VPC

Se apenas os usuários do Kafka de sua VPC acessarem seus agentes do Kafka, será necessário configurar a fonte de eventos com o acesso da Amazon Virtual Private Cloud (Amazon VPC).

Autenticação SASL/SCRAM

O Lambda oferece suporte à autenticação Simple Authentication e Security Layer/Salted Challenge Response Authentication Mechanism (SASL/SCRAM) com criptografia (`SASL_SSL`) Transport Layer

Security (TLS). O Lambda envia as credenciais criptografadas para autenticar com o cluster. O Lambda não oferece suporte a SASL/SCRAM com texto simples (`SASL_PLAINTEXT`). Para obter mais informações sobre a autenticação do SASL/SCRAM, consulte RFC 5802.

O Lambda também oferece suporte à autenticação SASL/PLAIN. Como esse mecanismo usa credenciais de texto não criptografado, a conexão com o servidor deve usar criptografia TLS para garantir que as credenciais sejam protegidas.

Para a autenticação SASL, armazene as credenciais de login como um segredo no AWS Secrets Manager. Para obter mais informações sobre o uso do Secrets Manager, consulte Tutorial: Create and retrieve a secret (Criar e recuperar um segredo) no Guia do usuário do AWS Secrets Manager.

> **Important**
>
> Para usar o Secrets Manager para autenticação, os segredos devem estar armazenados na mesma região da AWS que sua função do Lambda.

Autenticação TLS mútua

O TLS mútuo (mTLS) fornece autenticação bidirecional entre o cliente e o servidor. O cliente envia um certificado ao servidor para que o servidor verifique o cliente, e o servidor envia um certificado ao cliente para que o cliente verifique o servidor.

No Apache Kafka autogerenciado, o Lambda atua como cliente. Você configura um certificado de cliente (como um segredo no Secrets Manager) para autenticar o Lambda com os agentes do Kafka. O certificado do cliente deve ser assinado por uma autoridade de certificação no armazenamento de confiança do servidor.

O cluster do Kafka envia um certificado de servidor ao Lambda para autenticar os agentes do Kafka com o Lambda. O certificado do servidor pode ser um certificado CA público ou um certificado de CA privado/autoassinado. O certificado CA público deve ser assinado por uma autoridade de certificação (CA) que esteja no repositório de confiança do Lambda. Para um certificado CA privado/autoassinado, configure o certificado CA raiz do servidor (como um segredo no Secrets Manager). O Lambda usa o certificado raiz para verificar os agentes do Kafka.

Para obter mais informações sobre o mTLS, consulte Introducing mutual TLS authentication for Amazon MSK as an event source (Introdução à autenticação TLS mútua para o Amazon MSK como fonte de eventos).

Configurar o segredo do certificado do cliente

O segredo CLIENT_CERTIFICATE_TLS_AUTH requer um campo de certificado e um campo de chave privada. Para uma chave privada criptografada, o segredo requer uma senha de chave privada. Tanto o certificado como a chave privada devem estar no formato PEM.

> **Note**
>
> O Lambda oferece suporte a algoritmos de criptografia de chave privada PBES1 (mas não PBES2).

O campo certificate (certificado) deve conter uma lista de certificados, começando pelo certificado do cliente, seguido por quaisquer certificados intermediários e terminando com o certificado raiz. Cada certificado deve iniciar em uma nova linha com a seguinte estrutura:

```
-----BEGIN CERTIFICATE-----
        <certificate contents>
-----END CERTIFICATE-----
```

O Secrets Manager oferece suporte a segredos de até 65.536 bytes, que é espaço suficiente para cadeias de certificados longas.

A chave privada deve estar no formato PKCS #8, com a seguinte estrutura:

```
-----BEGIN PRIVATE KEY-----
         <private key contents>
-----END PRIVATE KEY-----
```

Para uma chave privada criptografada, use a seguinte estrutura:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
          <private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

O exemplo a seguir exibe o conteúdo de um segredo para autenticação mTLS usando uma chave privada criptografada. Para uma chave privada criptografada, inclua a senha de chave privada no segredo.

```
{"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-----"
}
```

Configurar o segredo do certificado CA raiz do servidor

Crie esse segredo se seus agentes do Kafka usarem criptografia TLS com certificados assinados por uma CA privada. É possível usar criptografia TLS para autenticação de VPC, SASL/SCRAM, SASL/PLAIN ou mTLS.

O segredo do certificado CA raiz do servidor requer um campo que contenha o certificado CA raiz do agente do Kafka no formato PEM. O exemplo a seguir exibe a estrutura do segredo.

```
{"certificate":"-----BEGIN CERTIFICATE-----
MIID7zCCAtegAwIBAgIBADANBgkqhkiG9w0BAQsFADCBmDELMAkGA1UEBhMCVVMx
EDAOBgNVBAgTB0FyaXpvbmExEzARBgNVBAcTClNjb3R0c2RhbGUxJTAjBgNVBAoT
HFN0YXJmaWVsZCBUZWNobm9sb2dpZXMsIEluYy4xOzA5BgNVBAMTMlN0YXJmaWVs
ZCBTZXJ2aWNlcyBSb290IENlcnRpZmljYXRlIEF1dG...
-----END CERTIFICATE-----"
}
```

Gerenciar acesso e permissões de API

Além de acessar seu cluster do Kafka autogerenciado, a função Lambda necessita de permissões para executar várias ações da API. Adicione essas permissões à função de execução da função. Se

os usuários precisarem acessar alguma ação da API, adicione as permissões necessárias à política de identidade para o usuário ou a função do AWS Identity and Access Management (IAM).

Permissões de função do Lambda necessárias

Para criar e armazenar logs em um grupo de logs do Amazon CloudWatch Logs, sua função Lambda deve ter as seguintes permissões na função de execução:

- logs:CreateLogGroup
- logs:CreateLogStream
- logs:PutLogEvents

Permissões de função do Lambda opcionais

Sua função Lambda também pode precisar dessas permissões para:

- Descreva o segredo do Secrets Manager.
- Acessar a chave gerenciada pelo cliente AWS Key Management Service (AWS KMS)
- Acesse sua Amazon VPC.
- Enviar registros de invocações com falha para um destino

Secrets Manager e permissões do AWS KMS

Conforme o tipo de controle de acesso que você está configurando para seus agentes do Kafka, sua função Lambda poderá precisar de permissão para acessar seu segredo do Secrets Manager ou descriptografar sua chave do AWS KMS gerenciada pelo cliente. Para acessar esses recursos, a função de execução da função precisa ter as seguintes permissões:

- secretsmanager:GetSecretValue
- kms:Decrypt

Permissões da VPC

Se somente usuários dentro de uma VPC puderem acessar seu cluster do Apache Kafka autogerenciado, a função Lambda deverá ter permissão para acessar seus recursos da Amazon VPC. Esses recursos incluem sua VPC, sub-redes, security groups e interfaces de rede. Para acessar esses recursos, a função de execução da função precisa ter as seguintes permissões:

- ec2:CreateNetworkInterface
- ec2:DescribeNetworkInterfaces
- ec2:DescribeVpcs
- ec2:DeleteNetworkInterface
- ec2:DescribeSubnets
- ec2:DescribeSecurityGroups

Adicionar permissões à sua função de execução

Para acessar outros serviços da AWS que seu cluster autogerenciado do Apache Kafka usa, o Lambda utiliza as políticas de permissão que você define na função de execução da função do Lambda.

Por padrão, o Lambda não pode executar as ações obrigatórias ou opcionais para um cluster do Apache Kafka autogerenciado. Você deve criar e definir essas ações em uma política de confiança do IAM e, em seguida, associar a política à sua função de execução. Este exemplo mostra como criar uma política que permita que o Lambda acesse seus recursos da Amazon VPC.

```
{
        "Version":"2012-10-17",
        "Statement":[
           {
              "Effect":"Allow",
              "Action":[
                 "ec2:CreateNetworkInterface",
                 "ec2:DescribeNetworkInterfaces",
                 "ec2:DescribeVpcs",
                 "ec2:DeleteNetworkInterface",
                 "ec2:DescribeSubnets",
                 "ec2:DescribeSecurityGroups"
              ],
              "Resource":"*"
           }
        ]
     }
```

Para obter informações sobre como criar um documento de política JSON no console do IAM, consulte Criar políticas na guia JSON no Guia do usuário do IAM.

Conceder acesso aos usuários com uma política do IAM

Por padrão, usuários e perfis não têm permissão para executar operações de API de origem de eventos. Para conceder acesso a usuários de sua organização ou conta, crie ou atualize uma política baseada em identidades. Para obter mais informações, consulte Controlar o acesso aos recursos da AWS usando políticas no Manual do usuário do IAM.

Erros de autenticação e autorização

Se alguma das permissões necessárias para consumir dados do cluster do Kafka estiver ausente, o Lambda exibirá uma das mensagens a seguir de erro no mapeamento da fonte de eventos em LastProcessingResult.

Mensagens de erro

- O cluster falhou ao autorizar o Lambda
- SASL authentication failed (Falha na autenticação SASL)
- Server failed to authenticate Lambda (Falha ao autenticar o Lambda no servidor)
- Lambda failed to authenticate server (Falha ao autenticar o servidor no Lambda)
- Provided certificate or private key is invalid (O certificado ou a chave privada fornecida é inválida)

O cluster falhou ao autorizar o Lambda

Para SASL/SCRAM ou mTLS, esse erro indica que o usuário fornecido não tem todas estas permissões da lista de controle de acesso (ACL) do Kafka necessárias:

- Cluster DescribeConfigs
- Descrever grupo
- Ler grupo
- Descrever tópico
- Ler tópico

Ao criar ACLs do Kafka com as permissões necessárias do `kafka-cluster`, especifique o tópico e o grupo como recursos. O nome do tópico deve corresponder ao tópico no mapeamento da fonte de eventos. O nome do grupo deve corresponder ao UUID do mapeamento da fonte de eventos.

Depois de adicionar as permissões necessárias à função de execução, poderá levar vários minutos para que as alterações entrem em vigor.

SASL authentication failed (Falha na autenticação SASL)

Em SASL/SCRAM ou SASL/PLAIN, esse erro indica que as credenciais de login fornecidas não são válidas.

Server failed to authenticate Lambda (Falha ao autenticar o Lambda no servidor)

Esse erro indica que o agente do Kafka não conseguiu autenticar o Lambda. Esse erro pode ocorrer por estes motivos:

- Você não forneceu um certificado de cliente para autenticação mTLS.
- Você forneceu um certificado de cliente, mas os agentes do Kafka não estão configurados para usar a autenticação mTLS.
- Um certificado de cliente não é confiável para os agentes do Kafka.

Lambda failed to authenticate server (Falha ao autenticar o servidor no Lambda)

Esse erro indica que o Lambda não conseguiu autenticar o agente do Kafka. Esse erro pode ocorrer por estes motivos:

- Os agentes do Kafka usam certificados autoassinados ou uma CA privada, mas não fornecem o certificado CA raiz do servidor.
- O certificado CA raiz do servidor não corresponde à CA raiz que assinou o certificado do agente.
- A validação do nome de host falhou porque o certificado do agente não contém o nome DNS ou o endereço IP do agente como nome alternativo do assunto.

Provided certificate or private key is invalid (O certificado ou a chave privada fornecida é inválida)

Esse erro indica que o consumidor do Kafka não pôde usar o certificado fornecido ou a chave privada. Verifique se o certificado e a chave usam o formato PEM e que a criptografia de chave privada usa um algoritmo PBES1.

Configuração de rede

Para que o Lambda use seu cluster do Kafka como uma origem de eventos, ele precisa de acesso à Amazon VPC na qual o cluster reside. Recomendamos implantar endpoints da VPC do AWS PrivateLink para o Lambda acessar sua VPC. Implante endpoints para Lambda e AWS Security

Token Service (AWS STS). Se o agente usar autenticação, implante também um endpoint da VPC para o Secrets Manager. Se você tiver configurado um destino em caso de falha, implante também um endpoint da VPC para o serviço de destino.

Alternativamente, certifique-se de que a VPC associada ao cluster do Kafka inclua um gateway NAT por sub-rede pública. Para ter mais informações, consulte the section called “Acesso à Internet para funções da VPC”.

Se você usa endpoints da VPC, também deve configurá-los para habilitar nomes DNS privados.

Quando você cria um mapeamento da origem do evento para um cluster autogerenciado do Apache Kafka, o Lambda verifica se as interfaces de rede elástica (ENIs) já estão presentes nas sub-redes e nos grupos de segurança da VPC do seu cluster. Se o Lambda encontrar ENIs existentes, ele tentará reutilizá-las. Caso contrário, o Lambda criará novas ENIs para se conectar à origem do evento e invocar sua função.

> **Note**
>
> As funções do Lambda sempre são executadas em VPCs de propriedade do serviço Lambda. Essas VPCs recebem manutenção automática do serviço e não são visíveis para os clientes. Você também pode conectar sua função a uma Amazon VPC. Em ambos os casos, a configuração de VPC da sua função não afetará o mapeamento da origem do evento. Somente a configuração da VPC da origem de eventos determina o modo de conexão do Lambda à sua origem de eventos.

Para obter mais informações sobre como configurar a rede, consulteConfigurar o AWS Lambda com um cluster do Apache Kafka em uma VPC no blog AWS Compute.

Regras de grupos de segurança da VPC

Configure com as seguintes regras (no mínimo) os grupos de segurança da Amazon VPC que contêm seu cluster:

- Regras de entrada: permitir todo o tráfego na porta do agente do Kafka para os grupos de segurança especificados para sua fonte de eventos. O Kafka usa a porta 9092 por padrão.
- Regras de saída: permitir todo o tráfego na porta 443 para todos os destinos. Permita todo o tráfego na porta do agente do Kafka para os grupos de segurança especificados para sua fonte de eventos. O Kafka usa a porta 9092 por padrão.

- Se você estiver usando endpoints da VPC em vez de um gateway NAT, os grupos de segurança associados aos endpoints da VPC deverão permitir todo o tráfego de entrada na porta 443 dos grupos de segurança da fonte de eventos.

Trabalhar com endpoints da VPC

Quando você usa endpoints da VPC, as chamadas de API para invocar sua função são roteadas por esses endpoints usando as ENIs. A entidade principal do serviço Lambda precisa chamar `sts:AssumeRole` e `lambda:InvokeFunction` para quaisquer perfis e funções que usem essas ENIs.

Por padrão, os endpoints da VPC têm políticas do IAM que são abertas. A prática recomendada é restringir essas políticas a fim de permitir que somente entidades principais específicas executem as ações necessárias usando esse endpoint. Para garantir que seu mapeamento da origem do evento seja capaz de invocar sua função do Lambda, a política de endpoint da VPC deve permitir que a entidade principal do serviço Lambda chame `sts:AssumeRole` e `lambda:InvokeFunction`. A restrição de suas políticas de endpoint da VPC para permitir apenas chamadas de API originadas em sua organização impedirá o funcionamento adequado do mapeamento da origem do evento.

O seguinte exemplo de políticas de endpoint da VPC mostra como conceder o acesso necessário à entidade principal do serviço Lambda para os endpoints do AWS STS e Lambda.

Example Política de endpoint da VPC: endpoint do AWS STS

```
{
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Example Política de endpoint da VPC: endpoint do Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Se o seu agente do Kafka usar autenticação, você também poderá restringir a política de endpoint da VPC para o endpoint do Secrets Manager. Para chamar a API do Secrets Manager, o Lambda usa seu perfil de função, e não a entidade principal de serviço do Lambda. O exemplo a seguir mostra uma política de endpoint do Secrets Manager.

Example Política de endpoint da VPC: endpoint do Secrets Manager.

```
{
    "Statement": [
        {
            "Action": "secretsmanager:GetSecretValue",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "customer_function_execution_role_arn"
                ]
            },
            "Resource": "customer_secret_arn"
        }
    ]
}
```

Se você tiver um destino em caso de falha configurado, o Lambda também usará o perfil de sua função para chamar `s3:PutObject`, `sns:Publish` ou `sqs:sendMessage` usando as ENIs gerenciadas pelo Lambda.

Adicionar um cluster do Kafka como uma origem de evento

Para criar um mapeamento de origem de evento, adicione seu cluster do Kafka como um acionador de função do Lambda usando o console do Lambda, um AWS SDK ou a AWS Command Line Interface (AWS CLI).

Esta seção descreve como criar um mapeamento de origens de eventos do usando o console do Lambda e aAWS CLI.

Pré-requisitos

- Um cluster Apache Kafka autogerenciado. O Lambda é compatível com o Apache Kafka versão 0.10.1.0 e posteriores.
- Uma função de execução com permissão para acessar os recursos da AWS que o cluster do Kafka autogerenciado usa.

ID de grupo de consumidores personalizável

Ao configurar o Kafka como uma origem de eventos, você pode especificar um ID de grupo de consumidores. Esse ID de grupo de consumidores é um identificador existente para o grupo de consumidores do Kafka no qual você deseja que a função do Lambda ingresse. Você pode usar esse recurso para migrar facilmente qualquer configuração de processamento em andamento de registros do Kafka de outros consumidores para o Lambda.

Se você especificar um ID de grupo de consumidores e houver outros pesquisadores ativos dentro desse grupo de consumidores, o Kafka distribuirá mensagens entre todos os consumidores. Em outras palavras, o Lambda não receberá todas as mensagens para o tópico do Kafka. Se você quiser que o Lambda gerencie todas as mensagens do tópico, desative todos os outros pesquisadores desse grupo de consumidores.

Além disso, se você especificar um ID de grupo de consumidores e o Kafka encontrar um grupo de consumidores válido já existente com o mesmo ID, o Lambda ignorará o parâmetro `StartingPosition` no mapeamento de origem de eventos. Em vez disso, o Lambda começará a processar registros de acordo com o deslocamento confirmado do grupo de consumidores. Se você especificar um ID de grupo de consumidores e o Kafka não conseguir encontrar um grupo de consumidores existente, o Lambda configurará a origem de eventos com a `StartingPosition` especificada.

O ID do grupo de consumidores que você especificar deverá ser exclusivo entre todas as origens de eventos do Kafka. Após criar um mapeamento de origem de eventos do Kafka com o ID do grupo de consumidores especificado, você não poderá atualizar esse valor.

Adicionando um cluster Kafka autogerenciado (console)

Siga estas etapas para adicionar seu cluster autogerenciado do Apache Kafka e um tópico do Kafka como um acionador para sua função do Lambda.

Para adicionar um acionador do Apache Kafka à sua função do Lambda (console)

1. Abra a página Functions (Funções) no console do Lambda.
2. Escolha o nome da função do Lambda.
3. Em Visão geral da função, escolha Adicionar gatilho.
4. Em Trigger configuration (Configuração do acionador), faça o seguinte:

    a. Selecione o tipo de acionadro Apache Kafka.

    b. para oServidores de bootstrap, insira o endereço de host e par de portas de um corretor Kafka em seu cluster e escolhaAdicionar. Repita para cada agente da Kafka no cluster.

    c. para oNome do tópico, insira o nome do tópico do Kafka usado para armazenar registros no cluster.

    d. (Opcional) ParaTamanho do lote, insira o número máximo de registros a serem recebidos em um único lote.

    e. Em Batch window (Janela de lote), insira o tempo máximo em segundos usado pelo Lambda para coletar os registros antes de invocar a função.

    f. (Opcional) em Consumer group ID (ID do grupo de consumidores), insira o ID de um grupo de consumidores do Kafka no qual ingressar.

    g. (Opcional) Em Posição inicial, escolha Mais recente para começar a realizar a leitura do fluxo a partir do registro mais recente, Horizonte de corte para começar no registro mais antigo disponível ou No carimbo de data e hora para especificar um carimbo de data e hora para começar a realizar a leitura.

    h. (Opcional) Em VPC, escolha a Amazon VPC para seu cluster do Kafka. Em seguida, escolha VPC subnets (Sub-redes da VPC) e VPC security groups (Grupos de segurança da VPC).

        Essa configuração será necessária se apenas os usuários da VPC acessarem seus agentes.

i. (Opcional) Em Authentication (Autenticação), escolha Add (Adicionar) e faça o seguinte:

i. Escolha o protocolo de acesso ou a autenticação dos agentes do Kafka no cluster.
- Se o agente do Kafka usar autenticação SASL/PLAIN, escolha BASIC_AUTH.
- Se seu agente usar autenticação SASL/SCRAM, escolha um dos protocolos SASL_SCRAM.
- Se estiver configurando a autenticação mTLS, escolha o protocolo CLIENT_CERTIFICATE_TLS_AUTH.

ii. Para a autenticação SASL/SCRAM ou mTLS, escolha a chave secreta do Secrets Manager que contém as credenciais de seu cluster do Kafka.

j. (Opcional) Em Encryption (Criptografia), escolha o segredo do Secrets Manager que contém o certificado CA raiz que seus agentes do Kafka usam para criptografia TLS, se seus agentes do Kafka usarem certificados assinados por uma CA privada.

Essa configuração se aplica à criptografia TLS para SASL/SCRAM ou SASL/PLAIN e à autenticação MTLs.

k. Para criar o gatilho em um estado desativado para teste (recomendado), desmarqueAtivar gatilho. Ou, para habilitar o gatilho imediatamente, selecioneAtivar gatilho.

5. Para criar o acionador, selecione Add (Adicionar).

Adicionando um cluster Kafka autogerenciado (AWS CLI)

Use os comandos de exemplo da AWS CLI a seguir para criar e visualizar um acionador autogerenciado do Apache Kafka para sua função do Lambda.

Usar SASL/SCRAM

Se usuários do Kafka acessarem seus agentes do Kafka pela Internet, especifique o segredo do Secrets Manager criado para autenticação SASL/SCRAM. O exemplo a seguir usa o comando `create-event-source-mapping` da AWS CLI para mapear uma função do Lambda chamada `my-kafka-function` em um tópico do Kafka chamado `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"]}}'
```

Usar uma VPC

Se apenas os usuários do Kafka em sua VPC acessarem seus agentes do Kafka, você deverá especificar sua VPC, suas sub-redes e seu grupo de segurança da VPC. O exemplo a seguir usa o comando `create-event-source-mapping` da AWS CLI para mapear uma função do Lambda chamada `my-kafka-function` em um tópico do Kafka chamado `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"]}}'
```

Visualizar o status usando a AWS CLI

O exemplo a seguir usa o comando `get-event-source-mapping` da AWS CLI para descrever o status do mapeamento de fontes de eventos que você criou.

```
aws lambda get-event-source-mapping
          --uuid dh38738e-992b-343a-1077-3478934hjkfd7
```

Destinos em caso de falha

Para reter registros de invocações de mapeamento da origem do evento com falha, adicione um destino ao mapeamento da origem de eventos da função. Cada registro enviado ao destino é um documento JSON com metadados sobre a invocação que falhou. É possível configurar qualquer tópico do Amazon SNS, fila do Amazon SQS ou bucket do S3 como destino. Sua função de execução deve ter permissões para o destino:

- Para destinos SQS: sqs:SendMessage
- Para destinos SNS: sns:Publish
- Para destinos de bucket do S3: s3:PutObject e s3:ListBuckets

Além disso, se você configurou uma chave do KMS no seu destino, o Lambda precisará das seguintes permissões, dependendo do tipo de destino:

- Se você habilitou a criptografia com sua própria chave do KMS para um destino do S3, kms:GenerateDataKey é necessário. Se a chave do KMS e o destino do bucket do S3 estiverem em uma conta diferente da função do Lambda e do perfil de execução, configure a chave do KMS para confiar no perfil de execução para permitir kms:GenerateDataKey.
- Se você habilitou a criptografia com sua própria chave do KMS para um destino do SQS, kms:Decrypt e kms:GenerateDataKey são necessários. Se a chave do KMS e o destino da fila do SQS estiverem em uma conta diferente da função do Lambda e do perfil de execução, configure a chave do KMS para confiar no perfil de execução para permitir kms:Decrypt, kms:GenerateDataKey, kms:DescribeKey e kms:ReEncrypt.
- Se você habilitou a criptografia com sua própria chave do KMS para um destino do SNS, kms:Decrypt e kms:GenerateDataKey são necessários. Se a chave do KMS e o destino do tópico do SNS estiverem em uma conta diferente da função do Lambda e do perfil de execução, configure a chave do KMS para confiar no perfil de execução para permitir kms:Decrypt, kms:GenerateDataKey, kms:DescribeKey e kms:ReEncrypt.

Para configurar um destino em caso de falha usando o console, siga estas etapas:

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Em Function overview (Visão geral da função), escolha Add destination (Adicionar destino).
4. Em Origem, escolha Invocação do mapeamento da origem do evento.
5. Em Mapeamento da origem do evento, escolha uma origem de eventos configurada para essa função.
6. Em Condição, selecione Em caso de falha. Para invocações de mapeamento da origem de eventos, essa é a única condição aceita.
7. Em Tipo de destino, escolha o tipo de destino para o qual o Lambda envia registros de invocação.
8. Em Destination (Destino), escolha um recurso.
9. Escolha Salvar.

Você também pode configurar um destino em caso de falha usando a AWS CLI. Por exemplo, o seguinte comando create-event-source-mapping adiciona um mapeamento de origem de eventos com um destino SQS em caso de falha 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"}}'
```

O seguinte comando update-event-source-mapping adiciona um destino S3 em caso de falha à origem de eventos associada à entrada `uuid`:

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": "arn:aws:s3:::dest-bucket"}}'
```

Para remover um destino, forneça uma string vazia como argumento para o parâmetro `destination-config`:

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

Exemplo de registro de invocação do SNS e do SQS

O exemplo a seguir mostra o que é enviado pelo Lambda para o destino de tópico do SNS ou de fila do SQS em caso de falha na invocação da origem de eventos do Kafka. Cada uma das chaves abaixo de `recordsInfo` contém o tópico e a partição do Kafka separados por um hífen. Por exemplo, para a chave `"Topic-0"`, `Topic` é o tópico do Kafka, e `0` é a partição. Para cada tópico e partição, você pode usar os dados de desvios e do carimbo de data/hora para encontrar os registros de invocação originais.

```
{
    "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",
            }
        }
    }
}
```

Exemplo de registro de invocação de destino S3

Para destinos do S3, o Lambda envia todo o registro da invocação junto com os metadados para o destino. O exemplo a seguir mostra o que é enviado pelo Lambda para o destino de bucket do S3 em caso de falha na invocação da origem de eventos do Kafka. Além de todos os campos do exemplo anterior para destinos do SQS e do SNS, o campo `payload` contém o registro de invocação original como uma string JSON com 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**
>
> Recomendamos habilitar o versionamento do S3 no bucket de destino.

Usar um cluster do Kafka como uma fonte de eventos

Quando você adiciona seu cluster Apache Kafka como um gatilho para sua função do Lambda, o cluster é usado como umOrigem do evento.

O Lambda lê os dados de eventos dos tópicos do Kafka que você especifica como `Topics` em uma solicitação CreateEventSourceMapping com base na `StartingPosition` especificada. Após o processamento bem-sucedido, seu tópico do Kafka é confirmado no cluster do Kafka.

Se você especificar `StartingPosition` como `LATEST`, o Lambda começará a ler da mensagem mais recente em cada partição pertencente ao tópico. Como pode haver algum atraso após a configuração do acionador antes de o Lambda começar a ler as mensagens, o Lambda não lerá nenhuma mensagem produzida durante a janela.

O Lambda processa registros de uma ou mais partições de tópico do Kafka especificadas e envia uma carga útil JSON à função. Quando mais registros estiverem disponíveis, o Lambda continuará processando-os em lotes, com base no valor de `BatchSize` especificado na solicitação CreateEventSourceMapping, até que a função atinja o tópico.

Se sua função retorna um erro para qualquer uma das mensagens em um lote, o Lambda tenta novamente todo o lote de mensagens até que o processamento seja bem-sucedido ou as mensagens expiram. É possível enviar registros que apresentaram falha em todas as tentativas a um destino em caso de falha para processamento posterior.

> **Note**
>
> Embora as funções do Lambda normalmente tenham um limite máximo de tempo de 15 minutos, os mapeamentos da origem dos eventos para o Amazon MSK, o Apache Kafka autogerenciado, o Amazon DocumentDB e o Amazon MQ para ActiveMQ e RabbitMQ são compatíveis somente com funções com limites máximos de tempo limite de 14 minutos. Essa restrição garante que o mapeamento da origem do evento possa solucionar adequadamente os erros de função e repetições.

Posições iniciais de sondagem e fluxo

Esteja ciente de que a sondagem do fluxo durante a criação e as atualizações do mapeamento da origem do evento é, finalmente, consistente.

- Durante a criação do mapeamento da origem do evento, pode levar alguns minutos para a sondagem de eventos do fluxo iniciar.
- Durante as atualizações do mapeamento da origem do evento, pode levar alguns minutos para interromper e reiniciar a sondagem de eventos do fluxo.

Esse comportamento significa que, se você especificar `LATEST` como posição inicial do fluxo, o mapeamento da origem do evento poderá perder eventos durante a criação ou as atualizações. Para garantir que nenhum evento seja perdido, especifique a posição inicial do fluxo como `TRIM_HORIZON` ou `AT_TIMESTAMP`.

Dimensionamento automático da origem do evento Kafka

Quando você inicialmente cria uma fonte de eventos do Apache Kafka, o Lambda aloca um consumidor para processar todas as partições no tópico do Kafka. Cada consumidor conta com vários processadores em execução em paralelo para lidar com um aumento de workloads. O Lambda aumenta ou reduz a escala na vertical automaticamente do número de consumidores com base na workload. Para preservar a ordenação de mensagens em cada partição, o número máximo de consumidores é um consumidor por partição no tópico.

A cada um minuto, o Lambda avalia o atraso de compensação do consumidor de todas as partições do tópico. Se o atraso for muito alto, a partição está recebendo mensagens mais rápido do que o Lambda pode processá-las. Se necessário, o Lambda adiciona ou remove os consumidores do tópico. O processo de escalabilidade de adicionar ou remover consumidores ocorre em até três minutos após a avaliação.

Se sua função alvo do Lambda estiver sobrecarregada, o Lambda reduz o número de consumidores. Essa ação reduz a workload na função, reduzindo o número de mensagens que os consumidores podem recuperar e enviar para a função.

Para monitorar o throughput do tópico do Kafka, você pode visualizar as métricas de consumo do Apache Kafka, como `consumer_lageconsumer_offset`. Para verificar quantas invocações de função ocorrem em paralelo, você também pode monitorar oMétricas de simultaneidadePara a função do.

Erros de mapeamento da fonte de eventos

Quando você adicionar seu cluster do Apache Kafka como uma fonte de eventos para sua função do Lambda, se ela encontrar um erro, o consumidor do Kafka interrompe o processamento de registros. Os consumidores de uma partição de tópico são aqueles que se inscrevem, leem e processam seus registros. Seus outros consumidores do Kafka podem continuar processando registros, desde que não encontrem o mesmo erro.

Para determinar a causa de um consumidor parado, verifique o campo `StateTransitionReason` na resposta do `EventSourceMapping`. A lista a seguir descreve os erros de origem do evento que você pode receber:

`ESM_CONFIG_NOT_VALID`

A configuração do mapeamento da fonte de eventos não é válida.

`EVENT_SOURCE_AUTHN_ERROR`

O Lambda não conseguiu autenticar a fonte de eventos.

`EVENT_SOURCE_AUTHZ_ERROR`

O Lambda não tem as permissões necessárias para acessar a fonte de eventos.

`FUNCTION_CONFIG_NOT_VALID`

A configuração da função não é válida.

> **Note**
>
> Se os registros de eventos do Lambda excederem o limite de tamanho permitido de 6 MB, eles poderão ficar sem processamento.

Métricas do Amazon CloudWatch

O Lambda emite a métrica `OffsetLag` enquanto sua função processa registros. O valor dessa métrica é a diferença de deslocamento entre o último registro gravado no tópico da origem de eventos do Kafka e o último registro que o grupo de consumidores da função processou. Você pode usar `OffsetLag` para estimar a latência entre o momento em que um registro é adicionado e o momento em que o grupo de consumidores o processa.

Uma tendência crescente em `OffsetLag` pode indicar problemas com pesquisadores no grupo de consumidores da função. Para ter mais informações, consulte Trabalhar com métricas de funções Lambda.

Parâmetros de configuração autogerenciados do Apache Kafka

Todos os tipos de origem de evento Lambda compartilham o mesmoCreateEventSourceMappingeUpdateEventSourceMappingOperações da API. No entanto, apenas alguns dos parâmetros se aplicam ao Apache Kafka.

Parâmetros da fonte de eventos que se aplicam ao Apache Kafka autogerenciado

Parâmetro	Obrigatório	Padrão	Observações
BatchSize	N	100	Máximo: 10.000.
Habilitado	N	Habilitado	
FunctionName	Y		
FilterCriteria	N		Filtragem de eventos do Lambda
MaximumBatchingWindowInSeconds	N	500 ms	Comportamento de lotes
SelfManageDeventSource	Y		Lista de corretores Kafka. Pode definir apenas em Criar
SelfManagedKafkaEventSourceConfig	N	Contém o campo ConsumerGroupId que assume por padrão um valor exclusivo.	Pode definir apenas em Criar
SourceAccessConfigurations	N	Nenhuma credencial	Informações da VPC ou credenciais de autenticação para o cluster

Parâmetro	Obrigatório	Padrão	Observações
			Para SASL_PLAIN, defina como BASIC_AUTH
StartingPosition	Y		AT_TIMESTAMP, TRIM_HORIZON, ou LATEST Pode definir apenas em Criar
StartingPositionTimestamp	N		Obrigatório se StartingPosition estiver definido como AT_TIMESTAMP
Tópicos	Y		Nome do tópico Pode definir apenas em Criar

Usar o Lambda com o Amazon SQS

> **Note**
>
> Se você deseja enviar dados para um destino que não seja uma função do Lambda ou enriquecer os dados antes de enviá-los, consulte Amazon EventBridge Pipes (Pipes do Amazon EventBridge).

É possível usar uma função do Lambda para processar mensagens em uma fila do Amazon Simple Queue Service (Amazon SQS). O Lambda oferece suporte a filas padrão e filas FIFO (primeiro a entrar, primeiro a sair) para mapeamentos de origem de eventos.

Entender o comportamento de sondagens e lotes para mapeamentos de origem de eventos do Amazon SQS

Com os mapeamentos de origem de eventos do Amazon SQS, o Lambda sonda a fila e invoca sua função de forma síncrona com um evento. Cada evento pode conter um lote de várias mensagens da fila. O Lambda recebe esses eventos um lote por vez e invoca sua função uma vez para cada lote. Quando sua função processa um lote com êxito, o Lambda exclui suas mensagens da fila.

Quando o Lambda recebe um lote, as mensagens permanecem na fila, mas permanecem ocultas durante o tempo limite de visibilidade da fila. Se a função processar com êxito todas as mensagens no lote, o Lambda excluirá as mensagens da fila. Por padrão, se a sua função encontrar um erro durante o processamento de um lote, todas as mensagens naquele lote se tornarão visíveis na fila novamente após o tempo limite de visibilidade expirar. Por conta disso, o código da função deve ter a capacidade de processar a mesma mensagem várias vezes sem causar efeitos colaterais não intencionais.

> **Warning**
>
> Os mapeamentos da origem do evento do Lambda processam cada evento ao menos uma vez, podendo haver o processamento duplicado de registros. Para evitar possíveis problemas relacionados a eventos duplicados, é altamente recomendável tornar o código da função idempotente. Para saber mais, consulte Como tornar minha função do Lambda idempotente no Centro de Conhecimentos da AWS.

Para evitar que o Lambda processe uma mensagem diversas vezes, é possível configurar o mapeamento de origem de eventos para incluir falhas de itens de lote na resposta da função ou usar a API DeleteMessage para remover mensagens da fila à medida que a função do Lambda as processa com êxito.

Para obter informações sobre os parâmetros de configuração aceitos pelo Lambda para mapeamentos de origem de eventos do SQS, consulte the section called “Criar um mapeamento de origem de evento do SQS”.

Exemplo de evento de mensagem de fila padrão

Example Evento de mensagem do Amazon SQS (fila padrão)

```
{
```

```
    "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"
        }
    ]
}
```

Por padrão, o Lambda pesquisará até 10 mensagens em sua fila de uma só vez e envia esse lote para sua função. Para evitar a invocação da função com poucos registros, você pode configurar a origem de eventos para armazenar os registros em buffer por até cinco minutos, configurando uma janela de lote. Antes de invocar a função, o Lambda continuará a sondar as mensagens da fila padrão até a janela de lote expirar, a cota de tamanho da carga útil de invocação ser atingida ou o tamanho de lote máximo configurado ser atingido.

Se você estiver usando uma janela em lote e sua fila do SQS contiver um volume de tráfego muito baixo, o Lambda poderá esperar até 20 segundos antes de invocar sua função. Isso será válido mesmo se você definir uma janela de lote inferior a 20 segundos.

> **Note**
>
> Em Java, talvez você observe erros de ponteiro nulo ao desserializar o JSON. Isso pode ser causado por como a caixa de “Records” e “EventSourceArn” é convertida pelo mapeador de objetos JSON.

Exemplo de evento de mensagem de fila FIFO

Para as filas FIFO, os registros contêm atributos adicionais relacionados a desduplicação e sequenciamento.

Example Evento de mensagem do Amazon SQS (fila 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"
        }
    ]
}
```

Criar e configurar um mapeamento de origem de evento do Amazon SQS

Para processar mensagens do Amazon SQS com o Lambda, configure sua fila com as configurações apropriadas e, em seguida, crie um mapeamento de origem de evento do Lambda.

Configurar uma fila para usar com o Lambda

Se você ainda não tiver uma fila do Amazon SQS, crie uma para servir como uma origem de eventos para a sua função do Lambda. Depois, configure a fila para permitir que a sua função do Lambda processe cada lote de eventos.

Para permitir que a função tenha tempo para processar cada lote de registros, defina o tempo limite de visibilidade da fila de origem para, no mínimo, seis vezes o tempo limite da configuração na sua função. Esse tempo extra permite que o Lambda repita o processo se a execução da sua função for limitada durante o processamento de um lote anterior.

Por padrão, se o Lambda se deparar com um erro em algum momento durante o processamento de um lote, todas as mensagens naquele lote retornarão para a fila. Após o tempo limite de visibilidade, as mensagens se tornam visíveis para o Lambda novamente. Você pode configurar o mapeamento de origem de evento para usar respostas parciais em lote para recolocar na fila somente as mensagens com falha. Além disso, se a sua função não conseguir processar uma mensagem várias vezes, o Amazon SQS poderá enviá-la para uma fila de mensagens não entregues. Recomendamos definir a `maxReceiveCount` na política de redirecionamento da fila de origem para pelo menos 5. Isso dá ao Lambda algumas chances de tentar novamente antes de enviar mensagens com falha diretamente para a fila de mensagens não entregues.

Configurar permissões do perfil de execução do Lambda

A política gerenciada pela AWS AWSLambdaSQSQueueExecutionRole inclui as permissões necessárias para o Lambda ler da sua fila do Amazon SQS. Você pode adicionar essa política gerenciada ao perfil de execução da função.

Opcionalmente, se você estiver usando uma fila criptografada, também precisará adicionar a seguinte permissão à sua função de execução:

- kms:Decrypt

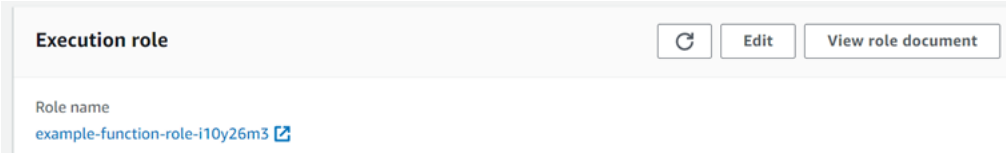
Criar um mapeamento de origem de evento do SQS

Crie um mapeamento de fontes de eventos para orientar o Lambda a enviar itens da sua fila para uma função do Lambda. Você pode criar vários mapeamentos da fonte do evento para processar itens de várias filas com uma única função. Quando o Lambda invoca a função de destino, o evento pode conter vários itens, até um tamanho de lote máximo configurável.

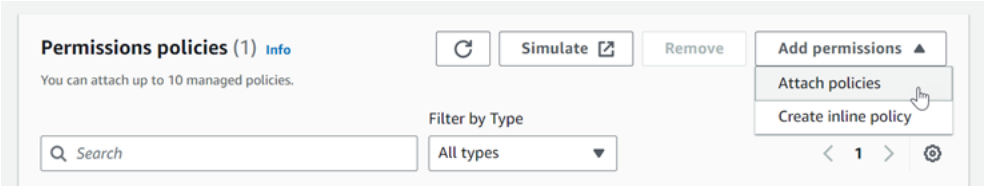
Para configurar sua função para ler do Amazon SQS, associe a política gerenciada pela AWS AWSLambdaSQSQueueExecutionRole ao seu perfil de execução. Em seguida, crie um mapeamento de origem de evento do SQS via console usando as etapas a seguir.

Para adicionar permissões e criar um acionador

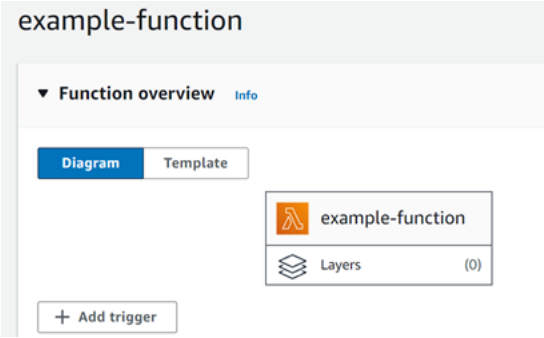
1. Abra a página Funções do console do Lambda.
2. Escolha o nome de uma função.
3. Escolha a guia Configuration (Configuração) e, depois, Permissions (Permissões).
4. Em Nome do perfil, escolha o link para seu perfil de execução. Esse link abre o perfil no console do IAM.



5. Escolha Adicionar permissões e depois Anexar políticas.



6. No campo de pesquisa, digite `AWSLambdaSQSQueueExecutionRole`. Adicione esta política ao seu perfil de execução Essa é uma política gerenciada pela AWS que contém as permissões que sua função precisa para ler de uma fila do Amazon SQS. Para obter mais informações sobre essa política, consulte AWSLambdaSQSQueueExecutionRole na Referência de políticas gerenciadas pela AWS.
7. Volte para a sua função no console do Lambda Em Visão geral da função, escolha Adicionar gatilho.



8. Escolha um tipo de acionador.
9. Configure as opções necessárias e escolha Add (Adicionar).

O Lambda oferece suporte às seguintes opções de configuração para origens de eventos do Amazon SQS:

Fila do SQS

A fila do Amazon SQS de onde os registros serão lidos.

Habilitar acionador

O status do mapeamento da origem do evento. Enable trigger (Habilitar acionador) está selecionado por padrão.

Tamanho do lote

O número máximo de registros a serem enviados para a função em cada lote. Em uma fila padrão, isso pode ser até 10.000 registros. Em uma fila FIFO, o máximo é 10. Para um tamanho de lote acima de dez, você também deve definir a janela do lote (`MaximumBatchingWindowInSeconds`) para, no mínimo, um segundo.

Configure o tempo limite da função para permitir tempo suficiente para o processamento de um lote inteiro de itens. Se os itens levarem muito tempo para serem processados, escolha um tamanho de lote menor. Um tamanho de lote grande pode melhorar a eficiência de cargas de trabalho muito rápidas ou com muita sobrecarga. Se você configurar a simultaneidade reservada na sua função, defina pelo menos cinco execuções simultâneas para reduzir as chances de erros de controle de utilização quando o Lambda invocar a sua função.

O Lambda transmite todos os registros do lote para a função em uma única chamada, desde que o tamanho total dos eventos não exceda a cota de tamanho da carga útil de invocação para invocação síncrona (6 MB). O Lambda e o Amazon SQS geram metadados para cada registro.

Esses metadados adicionais contam para o tamanho total da carga útil e podem fazer com que o total de registros enviados em um lote seja menor que o tamanho do lote configurado. Os campos de metadados enviados pelo Amazon SQS podem variar em termos de comprimento. Para saber mais sobre os campos de metadados do Amazon SQS, consulte a documentação da operação de API ReceiveMessage na Referência de APIs do Amazon Simple Queue Service.

Janela do lote

O máximo de tempo para reunir registros antes de invocar a função, em segundos. Aplicável somente a filas padrão.

Se você estiver usando uma janela de lote maior que zero segundos, deverá considerar o aumento do tempo de processamento no tempo limite de visibilidade da fila. Recomendamos definir o tempo limite de visibilidade da fila para seis vezes o tempo limite da função, acrescido do valor de `MaximumBatchingWindowInSeconds`. Isso permite que sua função do Lambda tenha tempo para processar cada lote de eventos e tentar novamente em caso de erro de controle de utilização.

Quando as mensagens ficam disponíveis, o Lambda começa a processar as mensagens em lotes. O Lambda começa a processar cinco lotes por vez com cinco invocações simultâneas de sua função. Se ainda houver mensagens disponíveis, o Lambda adicionará até 300 instâncias a mais da função por minuto, até alcançar 1.000 instâncias da função, no máximo. Para saber mais sobre a escalabilidade e simultaneidade, consulte Escalabilidade da função do Lambda.

Para processar mais mensagens, você pode otimizar sua função do Lambda para aumentar o throughput. Para obter mais informações, consulte Entender como o AWS Lambda escala com as filas padrão do Amazon SQS.

Máximo de simultaneidade

O número máximo de funções simultâneas que a origem do evento pode invocar. Para ter mais informações, consulte Configuração de simultaneidade máxima para origens de eventos do Amazon SQS.

Critérios de filtro

Adicione critérios de filtro para controlar quais eventos o Lambda enviará à função para processamento. Para ter mais informações, consulte Filtragem de eventos do Lambda.

Configurar o comportamento de escalabilidade para mapeamentos de origem de eventos do SQS

Para filas padrão, o Lambda usa sondagem longa para sondar uma fila até que ela se torne ativa. Quando as mensagens estão disponíveis, o Lambda começa a processar cinco lotes por vez com cinco invocações simultâneas da sua função. Se ainda houver mensagens disponíveis, o Lambda aumentará o número de processos de leitura de lotes em até 300 instâncias a mais por minuto. O número máximo de lotes que podem ser processados simultaneamente por um mapeamento de fontes de eventos é 1.000.

Para filas FIFO, o Lambda envia mensagens para a função na ordem em que as recebe. Ao enviar uma mensagem para uma fila do FIFO, você especifica umID do grupo de mensagens. O Amazon SQS garante que as mensagens no mesmo grupo sejam entregues ao Lambda em ordem. Quando o Lambda lê suas mensagens em lotes, cada lote poderá conter mensagens de mais de um grupo de mensagens, mas a ordem das mensagens será mantida. Se a função retornar um erro, ela fará todas as tentativas possíveis nas mensagens afetadas antes que o Lambda receba mensagens adicionais do mesmo grupo.

Configuração de simultaneidade máxima para origens de eventos do Amazon SQS

É possível usar a configuração de simultaneidade máxima para controlar o comportamento de escalabilidade de suas fontes de eventos do SQS. A configuração de simultaneidade máxima limita o número de instâncias simultâneas da função que uma origem de evento do Amazon SQS pode invocar. A simultaneidade máxima é uma configuração em nível de origem do evento. Se você tiver diversas origens de eventos do Amazon SQS mapeadas para uma função, cada origem de evento poderá ter uma configuração de simultaneidade máxima separada. É possível usar a simultaneidade máxima para evitar que uma fila use toda a simultaneidade reservada da função ou todo o restante da cota de simultaneidade da conta. Não há custos para configurar a simultaneidade máxima em uma origem de evento do Amazon SQS.

Importante, a simultaneidade máxima e a simultaneidade reservada são duas configurações independentes. Não defina a simultaneidade máxima maior que a simultaneidade reservada da função. Se você configurar a simultaneidade máxima, certifique-se de que a simultaneidade reservada da função seja maior ou igual à simultaneidade máxima total para todas as origens de eventos do Amazon SQS na função. Caso contrário, o Lambda pode aplicar um controle de utilização para as suas mensagens.

Se a simultaneidade máxima não for definida, o Lambda pode escalar sua origem de eventos do Amazon SQS até a cota total de simultaneidade da sua conta, que é 1.000 por padrão.

> **Note**
>
> Para filas FIFO, as invocações simultâneas são limitadas pelo número de IDs de grupos de mensagens (`messageGroupId`) ou pela configuração máxima de simultaneidade, o que for menor. Por exemplo, se você tiver seis IDs de grupo de mensagens e a simultaneidade máxima estiver definida como dez, sua função poderá ter no máximo seis invocações simultâneas.

Você pode configurar a simultaneidade máxima em mapeamentos da origem do evento novos e existentes do Amazon SQS.

Configuração da simultaneidade máxima usando o console do Lambda

1. Abra a página Funções do console do Lambda.
2. Escolha o nome de uma função.
3. Em Function overview (Visão geral da função), escolha SQS. Isso abre a guia Configuration (Configuração).
4. Selecione o acionador do Amazon SQS e escolha Edit (Editar).
5. Em Maximum concurrency (Simultaneidade máxima), insira um número entre dois e mil. Para desativar a simultaneidade máxima, deixe a caixa em branco.
6. Escolha Salvar.

Configuração da simultaneidade máxima usando a AWS Command Line Interface (AWS CLI)

Use o comando update-event-source-mapping com a opção `--scaling-config`. Exemplo:

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --scaling-config '{"MaximumConcurrency":5}'
```

Para desativar a simultaneidade máxima, insira um valor vazio para `--scaling-config`:

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --scaling-config "{}"
```

Configuração da simultaneidade máxima usando a API do Lambda

Use a ação CreateEventSourceMapping ou UpdateEventSourceMapping com um objeto ScalingConfig.

Tratamento de erros para uma origem de eventos do SQS no Lambda

Para lidar com erros relacionados a uma fonte de eventos do SQS, o Lambda usa automaticamente uma estratégia de repetição com uma estratégia de recuo. Você também pode personalizar o comportamento de tratamento de erros configurando o mapeamento de origem de eventos do SQS para retornar respostas parciais em lote.

Estratégia de recuo para invocações com falha

Quando uma invocação falha, o Lambda tenta repetir a invocação enquanto implementa uma estratégia de recuo. A estratégia de recuo difere ligeiramente caso o Lambda tenha encontrado a falha devido a um erro no código da função ou devido ao controle de utilização.

- Se o código da função causou o erro, o Lambda interromperá o processamento e repetirá a invocação. Enquanto isso, o Lambda recua gradualmente, reduzindo a quantidade de simultaneidade alocada ao mapeamento da origem do evento do Amazon SQS. Depois que limite de tempo de visibilidade da sua fila se esgotar, a mensagem será mostrada novamente na fila.
- Se a invocação apresentar falhas devido ao controle de utilização, o Lambda recua gradualmente as novas tentativas, reduzindo a quantidade de simultaneidade alocada para o mapeamento da origem do evento do Amazon SQS. O Lambda continuará a repetir a mensagem até que o carimbo de data/hora da mensagem exceda o tempo limite de visibilidade da fila, que corresponde ao momento em que o Lambda descartará a mensagem.

Implementar respostas parciais em lote

Quando sua função do Lambda encontra um erro ao processar um lote, todas as mensagens nesse lote tornam-se novamente visíveis na fila por padrão, incluindo mensagens que o Lambda processou com sucesso. Como resultado, a função pode acabar processando a mesma mensagem diversas vezes.

Para não precisar reprocessar todas as mensagens processadas com êxito em um lote com falha, você pode configurar o mapeamento da origem do evento para tornar visíveis novamente apenas as mensagens com falha. Isso se chama resposta parcial em lote. Para ativar respostas parciais em lote, especifique `ReportBatchItemFailures` para a ação FunctionResponseTypes ao configurar o mapeamento da origem do evento. Isso permite que a função retorne um sucesso parcial, podendo ajudar a reduzir o número de novas tentativas desnecessárias nos registros.

Quando `ReportBatchItemFailures` estiver ativado, o Lambda não reduzirá a escala verticalmente da sondagem de mensagens quando as invocações de funções falharem. Se você espera que algumas mensagens falhem, e não quer que essas falhas afetem a taxa de processamento de mensagens, use `ReportBatchItemFailures`.

> **Note**
>
> Ao usar respostas parciais em lote, tenha em mente:
>
> - Se a sua função lançar uma exceção, o lote inteiro será considerado uma falha total.
> - Se você estiver usando esse recurso com uma fila FIFO, a função deverá interromper o processamento de mensagens após a primeira falha e retornar todas as mensagens com falha e não processadas no `batchItemFailures`. Isso ajuda a preservar a ordem das mensagens na sua fila.

Para ativar o relatório parcial em lote

1. Leia as práticas recomendadas para implementar respostas parciais em lote.
2. Execute o comando a seguir para ativar `ReportBatchItemFailures` para a função. Para recuperar o UUID do mapeamento da origem do evento, execute o comando list-event-source-mappings da AWS CLI.

```
aws lambda update-event-source-mapping \
--uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
--function-response-types "ReportBatchItemFailures"
```

3. Atualize o código da função para capturar todas as exceções e retornar mensagens com falha em uma resposta `batchItemFailures` do JSON. A resposta `batchItemFailures` deve incluir uma lista de IDs de mensagens, como valores `itemIdentifier` do JSON.

   Por exemplo, suponha que você tenha um lote de cinco mensagens, com os IDs de mensagem `id1`, `id2`, `id3`, `id4` e `id5`. Sua função processa `id1`,`id3` e `id5` com sucesso. Para tornar as mensagens `id2` e `id4` novamente visíveis na fila, a função deve retornar a seguinte resposta:

```
{
  "batchItemFailures": [
        {
            "itemIdentifier": "id2"
```

```
        },
        {
            "itemIdentifier": "id4"
        }
    ]
}
```

Veja alguns exemplos de código de função que retornam a lista de IDs de mensagens com falha no lote:

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando o .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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}`);
}
```

Relatar falhas de item em lote do SQS com o Lambda usando 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 para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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 os eventos com falha não retornarem à fila, consulte Como soluciono problemas da função do Lambda do SQS ReportBatchItemFailures? no Centro de Conhecimento da AWS.

Condições de sucesso e falha

O Lambda considera um lote um sucesso total quando a função retorna qualquer um dos seguintes:

- Uma lista de `batchItemFailures` vazia
- Uma lista de `batchItemFailures` nula
- Uma `EventResponse` vazia
- Uma `EventResponse` nula

O Lambda considera um lote uma falha total quando a função retorna qualquer um dos seguintes:

- Uma resposta JSON inválida
- Uma string `itemIdentifier` vazia
- Uma `itemIdentifier` nula
- Um `itemIdentifier` com um nome de chave inválido
- Um valor `itemIdentifier` com um ID de mensagem inexistente

Métricas do CloudWatch

Para determinar se a função está reportando falhas de itens em lote corretamente, é possível monitorar as métricas do Amazon SQS `NumberOfMessagesDeleted` e `ApproximateAgeOfOldestMessage` no Amazon CloudWatch.

- `NumberOfMessagesDeleted` rastreia o número de mensagens removidas da sua fila. Se o número cair para 0, é um sinal de que a resposta da função não está retornando corretamente mensagens com falha.

- `ApproximateAgeOfOldestMessage` rastreia quanto tempo a mensagem mais antiga permaneceu na sua fila. Um aumento acentuado nessa métrica pode indicar que a função não está retornando mensagens com falha corretamente.

Parâmetros do Lambda para mapeamentos de origem de eventos do Amazon SQS

Todos os tipos de origem de evento Lambda compartilham o mesmoCreateEventSourceMappingeUpdateEventSourceMappingOperações de API do. No entanto, apenas alguns dos parâmetros se aplicam ao Amazon SQS.

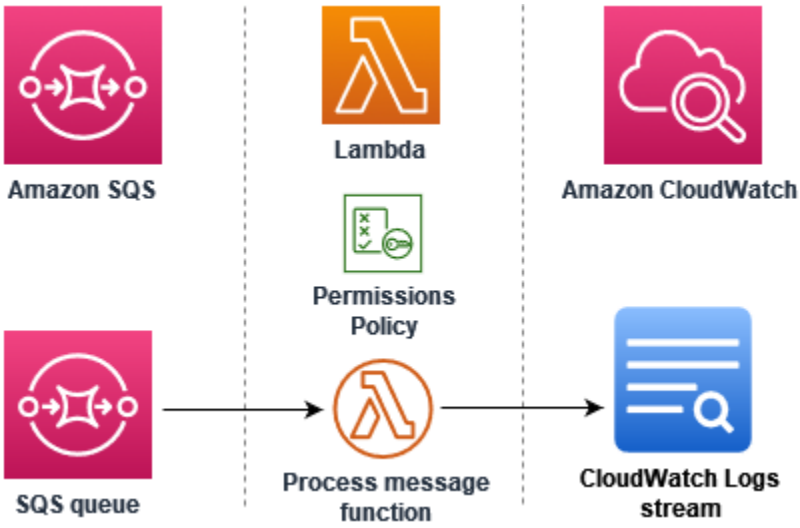
Parâmetros de fonte de evento que se aplicam ao Amazon SQS

Parâmetro	Obrigatório	Padrão	Observações
BatchSize	N	10	Para filas padrão, o máximo é 10 mil. Para filas FIFO, o máximo é dez.
Habilitado	N	verdadeiro	
EventSourceArn	Y		O ARN do fluxo de dados ou um consumidor de fluxo
FunctionName	Y		
FilterCriteria	N		Filtragem de eventos do Lambda
FunctionResponseTypes	N		Para permitir que sua função reporte falhas específicas em um lote, inclua o valor `ReportBatchItemFailures` em `FunctionResponseTypes` . Para ter

Parâmetro	Obrigatório	Padrão	Observações
			mais informações, consulte Implementar respostas parciais em lote.
MaximumBatchingWindowInSeconds	N	0	
ScalingConfig	N		Configuração de simultaneidade máxima para origens de eventos do Amazon SQS

Tutorial: usar o Lambda com o Amazon SQS

Neste tutorial, você criará uma função do Lambda que consome mensagens de uma fila do Amazon Simple Queue Service (Amazon SQS). A função do Lambda é executada sempre que uma nova mensagem é adicionada à fila. A função grava as mensagens em um fluxo do Amazon CloudWatch Logs. O diagrama a seguir mostrará os recursos da AWS que você usará para concluir o tutorial.



Para concluir este tutorial, execute as seguintes tarefas:

1. Crie uma função do Lambda que grave mensagens no CloudWatch Logs.

2. Criar uma fila do Amazon SQS.
3. Crie um mapeamento da origem do evento do Lambda. O mapeamento da origem do evento realiza a leitura da fila do Amazon SQS e invoca a função do Lambda quando uma nova mensagem é adicionada.
4. Teste a configuração ao adicionar mensagens à sua fila e ao monitorar os resultados no CloudWatch Logs.

Pré-requisitos

Cadastre-se em uma Conta da AWS

Se você ainda não tem Conta da AWS, siga as etapas a seguir para criar uma.

Para se cadastrar em uma Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

   Quando você se cadastra em uma Conta da AWS, um Usuário raiz da conta da AWS é criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como uma prática recomendada de segurança, atribua o acesso administrativo para um usuário e use somente o usuário-raiz para executar tarefas que requerem o acesso de usuário-raiz.

A AWS envia um e-mail de confirmação depois que o processo de cadastramento é concluído. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se cadastrar em uma Conta da AWS, proteja seu Usuário raiz da conta da AWS, habilite o AWS IAM Identity Center e crie um usuário administrativo para não usar o usuário raiz em tarefas cotidianas.

Proteger seu Usuário raiz da conta da AWS

1. Faça login no AWS Management Console como o proprietário da conta ao escolher a opção Usuário raiz e inserir o endereço de e-mail da Conta da AWS. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Signing in as the root user (Fazer login como usuário-raiz) no Guia do usuário do Início de Sessão da AWS.
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo MFA virtual para o usuário-raiz de sua conta da Conta da AWS (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center.
2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para obter um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso dos usuários com o Diretório do Centro de Identidade do IAM padrão no Guia do usuário do AWS IAM Identity Center.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use a URL de login que foi enviada ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

  Para obter ajuda com o login utilizando um usuário do Centro de Identidade do IAM, consulte Fazer login no portal de acesso da AWS, no Guia do usuário do Início de Sessão da AWS.

Atribuir acesso para usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center.

2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center.

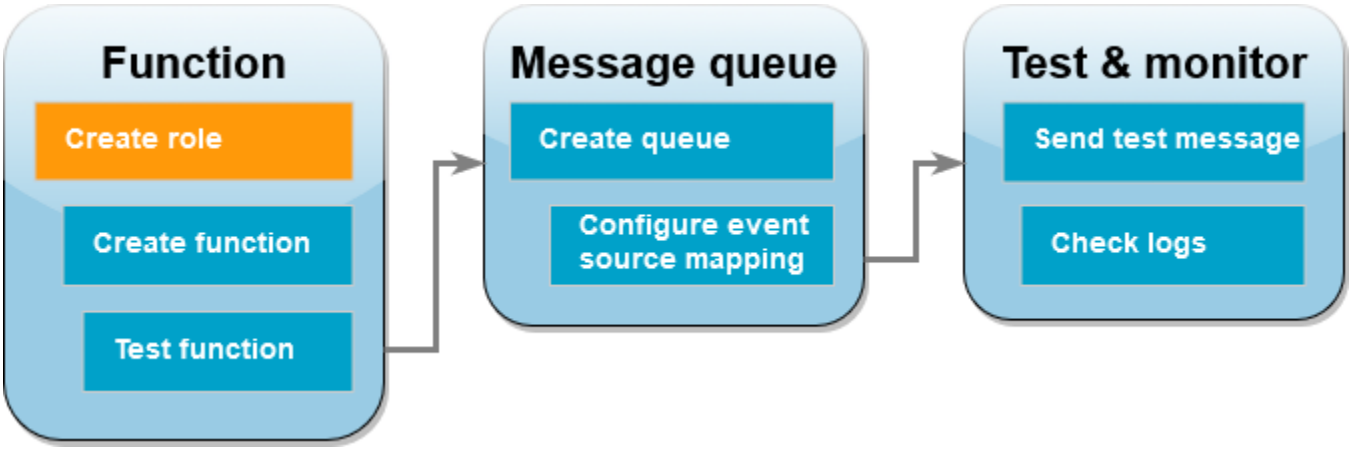
Instalar a AWS Command Line Interface

Se você ainda não instalou a AWS Command Line Interface, siga as etapas em Instalar ou atualizar a versão mais recente da AWS CLI para instalá-la.

O tutorial requer um terminal de linha de comando ou um shell para executar os comandos. No Linux e no macOS, use o gerenciador de pacotes e de shell de sua preferência.

> **Note**
>
> No Windows, alguns comandos da CLI do Bash que você costuma usar com o Lambda (como `zip`) não são compatíveis com os terminais integrados do sistema operacional. Para obter uma versão do Ubuntu com o Bash integrada no Windows, instale o Subsistema do Windows para Linux.

Criar a função de execução



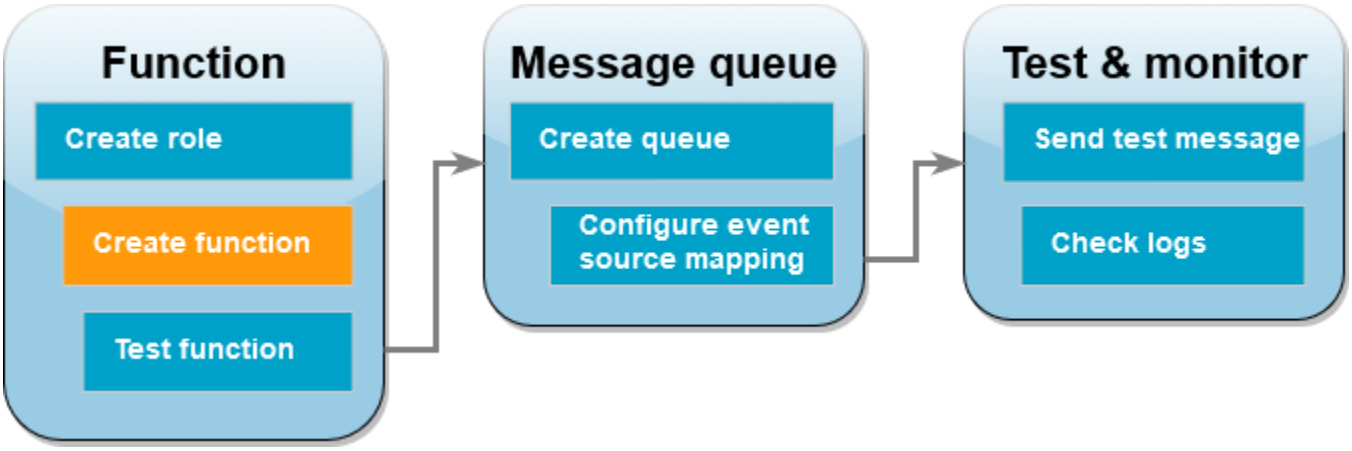
Um perfil de execução é um perfil do AWS Identity and Access Management (IAM) que concede a uma função do Lambda permissão para acessar serviços e recursos da AWS. Para permitir que a função realize a leitura de itens do Amazon SQS, anexe a política de permissões AWSLambdaSQSQueueExecutionRole.

Criar um perfil de execução e anexar uma política de permissões do Amazon SQS

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione Criar função.
3. Em Tipo de entidade confiável, escolha Serviços da AWS.
4. Em Caso de uso, escolha Lambda.
5. Escolha Próximo.
6. Na caixa de pesquisa Políticas de permissões, insira **`AWSLambdaSQSQueueExecutionRole`**.
7. Selecione a política AWSLambdaSQSQueueExecutionRole e, em seguida, escolha Próximo.
8. Em Detalhes do perfil, para Nome do perfil, insira **`lambda-sqs-role`** e, em seguida, escolha Criar perfil.

Após a criação da função, anote o nome de recurso da Amazon (ARN) do seu perfil de execução. Você precisará dele em etapas posteriores.

Criar a função



Crie uma função do Lambda queprocessesuas mensagens do Amazon SQS. O código da função registra o corpo da mensagem do Amazon SQS no CloudWatch Logs.

Este tutorial usa o runtime do Node.js 18.x, mas também fornecemos exemplos de arquivos em outras linguagens de runtime. Você pode selecionar a guia na caixa a seguir para ver o código do runtime do seu interesse. O código JavaScript que você usará nesta etapa é o primeiro exemplo mostrado na guia JavaScript.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
```

```
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consumir um evento do SQS com o Lambda usando TypeScript.

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

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consuma um evento do SQS com o Lambda usando 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
}
```

Criar uma função do Lambda em Node.js

1. Crie um diretório para o projeto e depois mude para esse diretório.

```
mkdir sqs-tutorial
cd sqs-tutorial
```

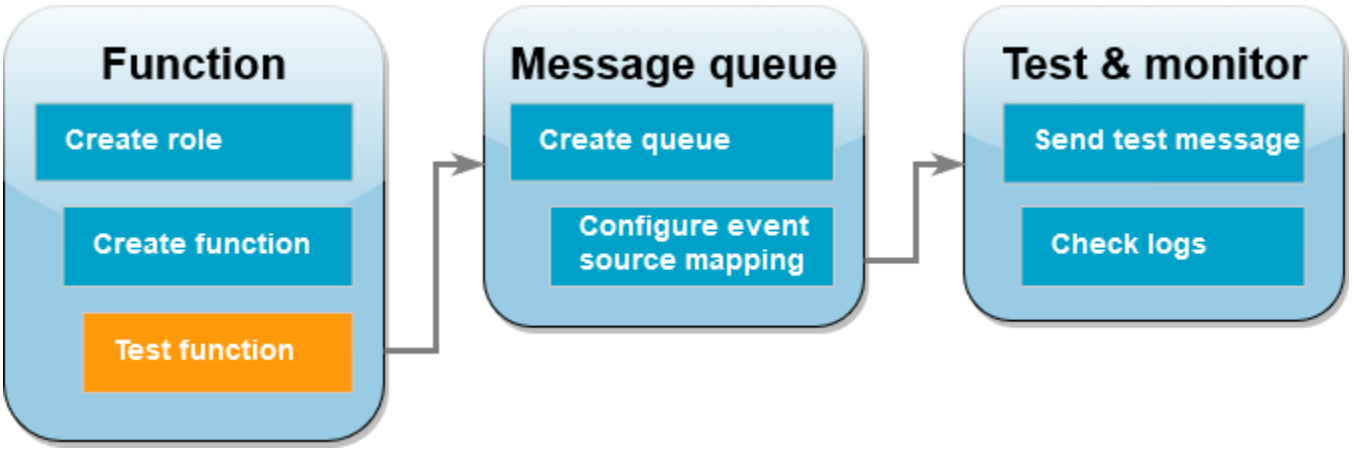
2. Copie o código JavaScript de amostra em um novo arquivo denominado `index.js`.
3. Crie um pacote de implantação usando o comando `zip` a seguir.

```
zip function.zip index.js
```

4. Crie uma função do Lambda usando o comando create-function da AWS CLI. Para o parâmetro `role`, insira o ARN da função de execução criada anteriormente.

```
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
```

Testar a função



Invoque sua função do Lambda manualmente usando o comando da `invoke` AWS CLI e um evento do Amazon SQS de amostra.

Invocar a função do Lambda com um evento de amostra

1. Salve o JSON a seguir como um arquivo denominado `input.json`. Esse JSON simula um evento que o Amazon SQS pode enviar para a função do Lambda, no qual `"body"` contém a mensagem real da fila. Neste exemplo, a mensagem é `"test"`.

   Example Evento do Amazon SQS

   Este é um evento de teste, portanto, você não precisa alterar a mensagem ou o número da conta.

```
{
    "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. Execute o comando invoke da AWS CLI apresentado a seguir. Esse comando retorna os logs do CloudWatch na resposta. Para obter mais informações sobre recuperação de logs, consulte Acesso a logs com a AWS CLI.

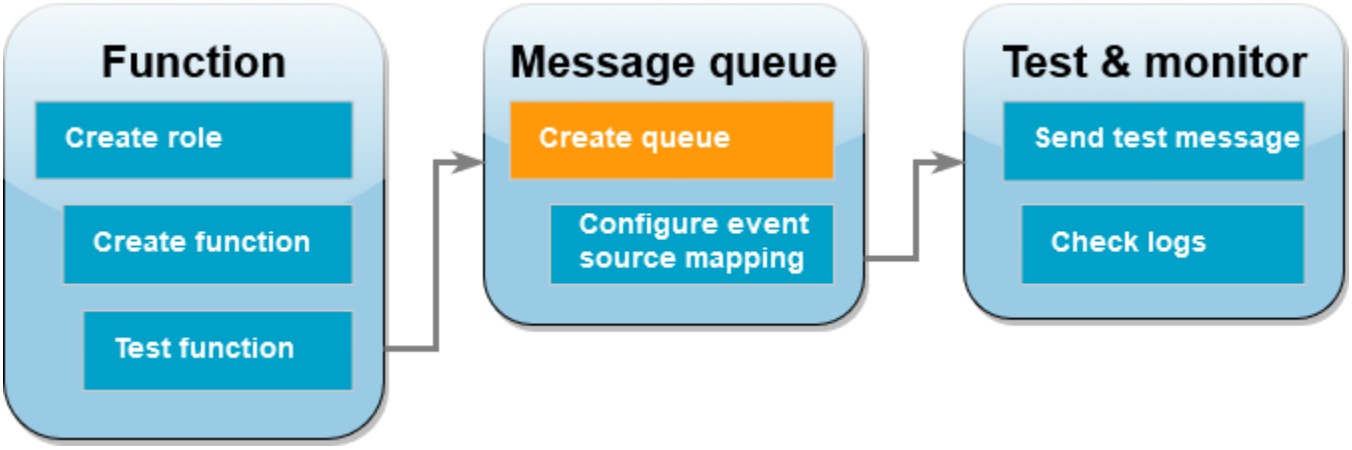
```
aws lambda invoke --function-name ProcessSQSRecord --payload file://input.json out
 --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

   A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

3. Encontre o log `INFO` na resposta. É aqui que a função do Lambda registra em log o corpo da mensagem. Você deve ver logs semelhantes a este:

```
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
```

Criar uma fila do Amazon SQS



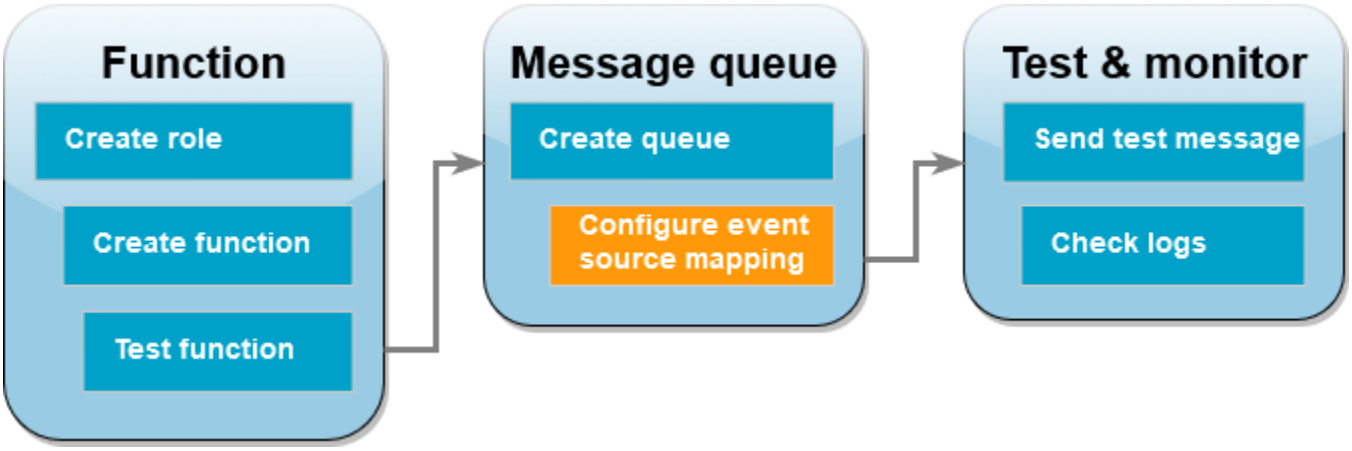
Crie uma fila do Amazon SQS que a função do Lambda possa usar como uma fonte de eventos.

Para criar uma fila

1. Abra o console do Amazon SQS.
2. Selecione Criar fila.
3. Insira um nome para a fila. Deixe todas as outras opções nas configurações padrão.
4. Selecione Criar fila.

Depois de criar a fila, anote seu ARN. Você precisará dele na próxima etapa, quando for associar a fila à função do Lambda.

Configurar a origem de evento



Conecte a fila do Amazon SQS à função do Lambda ao criar um mapeamento da origem do evento. O mapeamento da origem do evento realiza a leitura da fila do Amazon SQS e invoca a função do Lambda quando uma nova mensagem é adicionada.

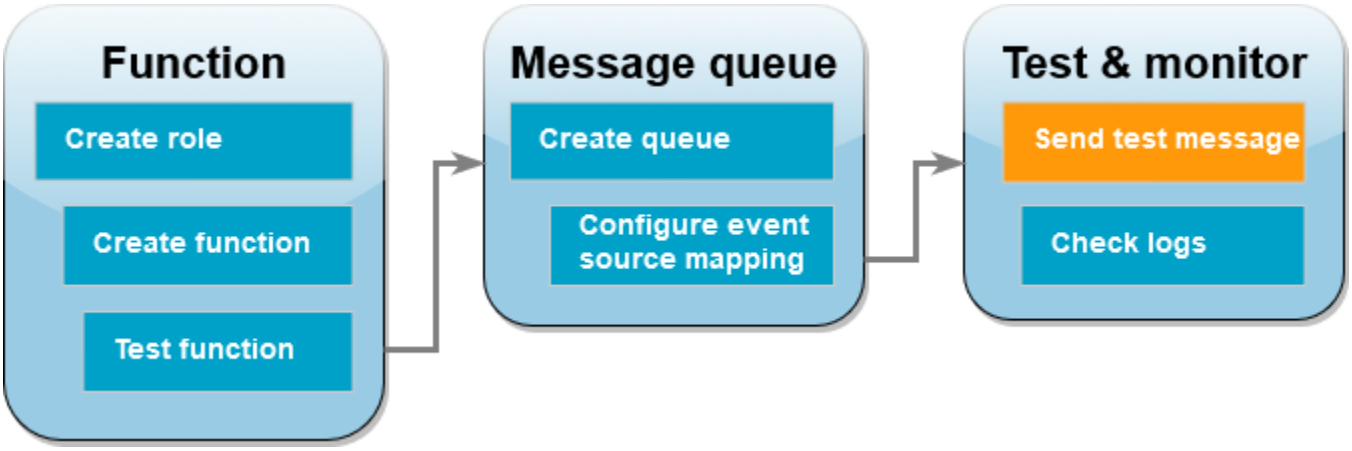
Para criar um mapeamento entre a fila do Amazon SQS e a função do Lambda, use o comando create-event-source-mapping da AWS CLI. Exemplo:

```
aws lambda create-event-source-mapping --function-name ProcessSQSRecord  --batch-size
 10 \
--event-source-arn arn:aws:sqs:us-east-1:111122223333:my-queue
```

Para obter uma lista dos mapeamentos da origem dos eventos, use o comando list-event-source-mappings. Exemplo:

```
aws lambda list-event-source-mappings --function-name ProcessSQSRecord
```

Envie uma mensagem de teste

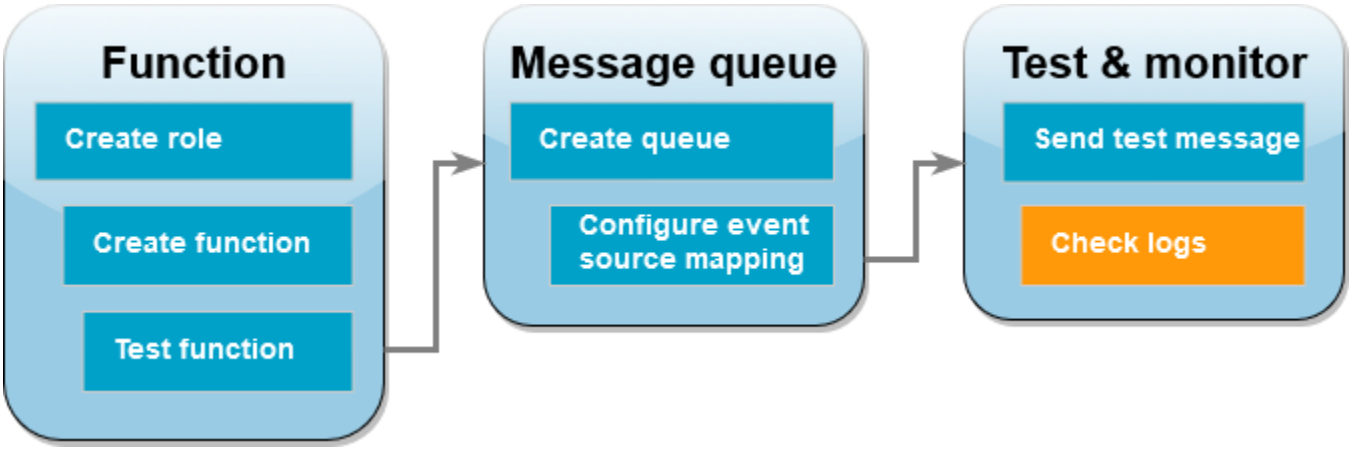


Enviar uma mensagem do Amazon SQS para a função do Lambda

1. Abra o console do Amazon SQS.
2. Escolha a fila que você criou anteriormente.
3. Escolha Send and receive messages (Enviar e receber mensagens).
4. Em Corpo da mensagem, insira uma mensagem de teste, como “esta é uma mensagem de teste”.
5. Escolha Send Message (Enviar mensagem).

O Lambda pesquisa a fila para atualizações. Quando há uma nova mensagem, o Lambda invoca a sua função com esses novos dados de evento da fila. Caso o manipulador de função retorne sem exceções, o Lambda considera a mensagem processada com êxito e começa a realizar a leitura de novas mensagens na fila. Após o processamento com sucesso, uma mensagem,o Lambda a exclui automaticamente da fila. Se o manipulador gera uma exceção, o Lambda considera que o batch de mensagens não foi processado com êxito e o Lambda invoca novamente a função com o mesmo batch de mensagens.

Verificar os logs do CloudWatch



Confirmar que a função processou a mensagem

1. Abra a página Funções do console do Lambda.
2. Escolha a função ProcessSQSRecord.
3. Escolha Monitor.
4. Escolha Visualizar logs do CloudWatch.
5. No console do CloudWatch, escolha o Fluxo de logs para a função.
6. Encontre o log `INFO`. É aqui que a função do Lambda registra em log o corpo da mensagem. Você deve visualizar a mensagem que enviou da fila do Amazon SQS. Exemplo:

```
2023-09-11T22:49:12.730Z b0c41e9c-0556-5a8b-af83-43e59efeec71 INFO Processed
 message this is a test message.
```

Limpe os recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Para excluir a função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Escolha Excluir.
4. Insira o nome do perfil no campo de entrada de texto e escolha Delete (Excluir).

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Para excluir a fila do Amazon SQS

1. Faça login no AWS Management Console e abra o console do Amazon SQS em https://console.aws.amazon.com/sqs/.
2. Selecione a fila que você criou.
3. Escolha Excluir.
4. Digite **`confirm`** no campo de entrada de texto.
5. Escolha Excluir.

Tutorial: Uso de uma fila do Amazon SQS entre contas como a origem de um evento

Neste tutorial, você criará uma função do Lambda que consome mensagens de uma fila do Amazon Simple Queue Service (Amazon SQS) em uma conta da AWS diferente. Este tutorial envolve duas contas da AWS: Conta A refere-se à conta que contém sua função do Lambda e Conta B refere-se à conta que contém a fila do Amazon SQS.

Pré-requisitos

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em Criar uma função do Lambda com o console para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, a AWS Command Line Interface (AWS CLI) versão 2 será necessária. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

A seguinte saída deverá ser mostrada:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell.

> **Note**
>
> No Windows, alguns comandos da CLI do Bash que você costuma usar com o Lambda (como `zip`) não são compatíveis com os terminais integrados do sistema operacional. Para obter uma versão do Ubuntu com o Bash integrada no Windows, instale o Subsistema do Windows para Linux. Os exemplos de comandos da CLI neste guia usam a formatação Linux. Os comandos que incluem documentos JSON em linha deverão ser reformatados se você estiver usando a CLI do Windows.

Criar a função de execução (Conta A)

Na Conta A, crie uma função de execução que dê à sua função permissão para acessar os recursos da AWS necessários.

Para criar uma função de execução

1. Abra a página Roles (Funções) no console do AWS Identity and Access Management (IAM).
2. Selecione Criar função.

3. Crie uma função com as propriedades a seguir.

   - Trusted entity (Entidade confiável): AWS Lambda.
   - Permissões: AWSLambdaSQSQueueExecutionRole.
   - Role name (Nome da função): **`cross-account-lambda-sqs-role`**

A política AWSLambdaSQSQueueExecutionRole tem as permissões necessárias para a função ler itens do Amazon SQS e para gravar logs no Amazon CloudWatch Logs.

Criar a função (Conta A)

Na Conta A, crie uma função do Lambda para processar suas mensagens do Amazon SQS. O exemplo de código Node.js 18 a seguir grava cada mensagem em um log do CloudWatch Logs.

Example index.mjs

```
export const handler = async function(event, context) {
  event.Records.forEach(record => {
    const { body } = record;
    console.log(body);
  });
  return {};
}
```

Para criar a função

> **Note**
>
> Seguir estas etapas cria uma função no Node.js 18. Para outras linguagens, as etapas são semelhantes, mas alguns detalhes são diferentes.

1. Salve o exemplo de código como um arquivo denominado `index.mjs`.
2. Crie um pacote de implantação.

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

3. Crie a função usando o comando `create-function` da 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
```

Teste a função (Conta A)

Na Conta A, invoque sua função do Lambda manualmente usando o comando `invoke` AWS CLI e um evento do Amazon SQS de exemplo.

Se o manipulador obtém um retorno normal sem exceções, o Lambda considerará a mensagem como processada com êxito e começará a ler novas mensagens na fila. Após o processamento com sucesso, uma mensagem,o Lambda a exclui automaticamente da fila. Se o manipulador gera uma exceção, o Lambda considera que o batch de mensagens que não foram processadas com êxito e o Lambda invoca novamente a função com o mesmo batch de mensagens.

1. Salve o JSON a seguir como um arquivo denominado `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"
        }
    ]
}
```

O JSON anterior simula um evento que o Amazon SQS pode enviar para a função do Lambda, onde "`body`" contém a mensagem real da fila.

2. Execute o comando `invoke` a seguir da AWS CLI.

```
aws lambda invoke --function-name CrossAccountSQSExample \
--cli-binary-format raw-in-base64-out \
--payload file://input.txt outputfile.txt
```

   A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

3. Verifique a saída no arquivo `outputfile.txt`.

Criar uma fila do Amazon SQS (Conta B)

Na Conta B, crie uma fila do Amazon SQS que a função do Lambda na Conta A possa usar como uma fonte de eventos.

Para criar uma fila

1. Abra o console do Amazon SQS.
2. Selecione Criar fila.
3. Crie uma fila com as propriedades a seguir.
   - Type (Tipo): Standard (Padrão)
   - Name (Nome): LambdaCrossAccountQueue
   - Configuration (Configuração): mantenha as configurações padrão.
   - Access policy (Política de acesso): selecione Advanced (Avançado). Cole a seguinte política do 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"
      }
    ]
}
```

Esta política concede à função de execução do Lambda na Conta A permissões para consumir mensagens desta fila do Amazon SQS.

4. Depois de criar a fila, registre o nome do recurso da Amazon (ARN). Você precisará dele na próxima etapa, quando for associar a fila à função do Lambda.

Configurar a origem do evento (Conta A)

Na Conta A, crie um mapeamento da origem do evento entre a fila do Amazon SQS na Conta B e sua função do Lambda executando o comando `create-event-source-mapping` da AWS CLI a seguir.

```
aws lambda create-event-source-mapping --function-name CrossAccountSQSExample --batch-size 10 \
--event-source-arn arn:aws:sqs:us-east-1:<AccountB_ID>:LambdaCrossAccountQueue
```

Para obter uma lista de mapeamentos de fonte do evento, execute o comando a seguir.

```
aws lambda list-event-source-mappings --function-name CrossAccountSQSExample \
--event-source-arn arn:aws:sqs:us-east-1:<AccountB_ID>:LambdaCrossAccountQueue
```

Testar a configuração

Você agora pode testar a configuração da seguinte forma:

1. Na Conta B, abra o console do Amazon SQS.
2. Escolha LambdaCrossAccountQueue criada anteriormente.
3. Escolha Send and receive messages (Enviar e receber mensagens).
4. Em Message body (Corpo da mensagem), insira uma mensagem de teste.
5. Escolha Send Message (Enviar mensagem).

Sua função do Lambda na Conta A deve receber a mensagem. O Lambda continuará a sondar a fila em busca de atualizações. Quando há uma nova mensagem, o Lambda invoca a sua função com esses novos dados de evento da fila. A função é executada e cria logs no Amazon CloudWatch. Você pode visualizar os logs no console do CloudWatch.

Limpe os recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Na Conta A, limpe sua função de execução e a função do Lambda.

Para excluir a função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Escolha Excluir.
4. Insira o nome do perfil no campo de entrada de texto e escolha Delete (Excluir).

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Na Conta B, limpe a fila do Amazon SQS.

Para excluir a fila do Amazon SQS

1. Faça login no AWS Management Console e abra o console do Amazon SQS em https://console.aws.amazon.com/sqs/.
2. Selecione a fila que você criou.
3. Escolha Excluir.
4. Digite **`confirm`** no campo de entrada de texto.

5. Escolha Excluir.

Processar eventos do Amazon DocumentDB com o Lambda

Você pode usar uma função do Lambda para processar eventos em um fluxo de alterações do Amazon DocumentDB (compatível com MongoDB) ao configurar um cluster do Amazon DocumentDB como uma origem do evento. Em seguida, você pode automatizar workloads orientadas por eventos invocando a função do Lambda sempre que os dados são alterados com o cluster do Amazon DocumentDB.

> **Note**
>
> O Lambda é compatível somente com as versões 4.0 e 5.0 do Amazon DocumentDB. O Lambda não é compatível com a versão 3.6.
> Além disso, para os mapeamentos da origem do evento, o Lambda oferece suporte somente a clusters baseados em instâncias e a clusters regionais. O Lambda não é compatível com clusters elásticos ou clusters globais. Essa limitação não se aplica ao usar o Lambda como cliente para se conectar ao Amazon DocumentDB. O Lambda pode se conectar a todos os tipos de cluster para realizar operações CRUD.

O Lambda processa eventos dos fluxos de alterações do Amazon DocumentDB sequencialmente na ordem em que chegam. Por causa disso, a função só pode processar uma invocação simultânea do DocumentDB de cada vez. Para monitorar a função, você pode rastrear as métricas de simultaneidade dela.

> **Warning**
>
> Os mapeamentos da origem do evento do Lambda processam cada evento ao menos uma vez, podendo haver o processamento duplicado de registros. Para evitar possíveis problemas relacionados a eventos duplicados, é altamente recomendável tornar o código da função idempotente. Para saber mais, consulte Como tornar minha função do Lambda idempotente no Centro de Conhecimentos da AWS.

Tópicos

- Exemplo de evento do Amazon DocumentDB

- Pré-requisitos e permissões
- Configuração de rede
- Criar um mapeamento da origem do evento do Amazon DocumentDB (console)
- Criar um mapeamento da origem do evento do Amazon DocumentDB (SDK ou CLI)
- Posições iniciais de sondagem e fluxo
- Monitorar a origem do evento do Amazon DocumentDB
- Tutorial: uso do AWS Lambda com o Amazon DocumentDB Streams

Exemplo de evento do 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"
}
```

Para obter mais informações sobre os eventos neste exemplo e suas formas, consulte Alterar eventos no site de documentação do MongoDB.

Pré-requisitos e permissões

Antes que você possa usar o Amazon DocumentDB como origem do evento a sua função do Lambda, observe os pré-requisitos a seguir. Você deve:

- Ter um cluster do Amazon DocumentDB existente na mesma Conta da AWS e Região da AWS como sua função. Se você não tiver um cluster existente, poderá criar um seguindo as etapas em Get Started with Amazon DocumentDB no Guia do desenvolvedor do Amazon DocumentDB. Como alternativa, o primeiro conjunto de etapas do Tutorial: uso do AWS Lambda com o Amazon DocumentDB Streams o guiará na criação de um cluster do DocumentDB com todos os pré-requisitos necessários.
- Permitir que o Lambda tenha acesso aos recursos da Amazon Virtual Private Cloud (Amazon VPC) associados ao cluster do Amazon DocumentDB. Para ter mais informações, consulte Configuração de rede.
- Habilitar o TLS no cluster do Amazon DocumentDB. Essa é a configuração padrão. Se você desabilitar o TLS, o Lambda não poderá se comunicar com o cluster.
- Ativar os fluxos de alterações no cluster do Amazon DocumentDB. Para obter mais informações, consulte Usar fluxos de alterações com o Amazon DocumentDB no Guia do desenvolvedor do Amazon DocumentDB.
- Fornecer ao Lambda credenciais para acessar o cluster do Amazon DocumentDB. Ao configurar a origem do evento, forneça a chave AWS Secrets Manager que contém os detalhes de autenticação (nome de usuário e senha) necessários para acessar o cluster. Para fornecer essa chave durante a configuração, execute uma das seguintes ações:
  - Se você estiver usando o console do Lambda para configurar, forneça essa chave no campo chave do Secrets Manager.
  - Se você estiver usando o AWS Command Line Interface (AWS CLI) para configuração, forneça essa chave na opção `source-access-configurations`. Você pode incluir essa opção com

o comando `create-event-source-mapping` ou o comando `update-event-source-mapping`. Por exemplo:

```
aws lambda create-event-source-mapping \
    ...
    --source-access-configurations
 '[{"Type":"BASIC_AUTH","URI":"arn:aws:secretsmanager:us-
west-2:123456789012:secret:DocDBSecret-AbC4E6"}]' \
    ...
```

- Conceder permissões ao Lambda para gerenciar recursos relacionados ao fluxo do Amazon DocumentDB. Adicione manualmente as seguintes permissões à função de execução da função:
  - rds:DescribeDBClusters
  - rds:DescribeDBClusterParameters
  - rds:DescribeDBSubnetGroups
  - ec2:CreateNetworkInterface
  - ec2:DescribeNetworkInterfaces
  - ec2:DescribeVpcs
  - ec2:DeleteNetworkInterface
  - ec2:DescribeSubnets
  - ec2:DescribeSecurityGroups
  - kms:Decrypt
  - secretsmanager:GetSecretValue
- Manter o tamanho dos eventos de fluxos de alterações do Amazon DocumentDB que você envia para o Lambda abaixo de 6 MB. O Lambda suporta apenas cargas úteis de até 6 MB. Se seu fluxo de alterações tentar enviar ao Lambda um evento maior que 6 MB, o Lambda descartará a mensagem e emitirá a métrica `OversizedRecordCount`. O Lambda emite todas as métricas com base no melhor esforço.

> **Note**
>
> Embora as funções do Lambda normalmente tenham um limite máximo de tempo de 15 minutos, os mapeamentos da origem dos eventos para o Amazon MSK, o Apache Kafka autogerenciado, o Amazon DocumentDB e o Amazon MQ para ActiveMQ e RabbitMQ são compatíveis somente com funções com limites máximos de tempo limite de 14 minutos. Essa

restrição garante que o mapeamento da origem do evento possa solucionar adequadamente os erros de função e repetições.

Configuração de rede

Para que o Lambda use seu cluster do Amazon DocumentDB como uma origem de eventos, ele precisa de acesso à Amazon VPC na qual o cluster reside. Recomendamos implantar endpoints da VPC do AWS PrivateLink para o Lambda acessar sua VPC. Implante um endpoint da VPC para o Lambda e, se o cluster usar autenticação, implante também um endpoint da VPC para o Secrets Manager.

Como alternativa, verifique se a VPC associada ao cluster do Amazon DocumentDB contém um gateway NAT por sub-rede pública. Para ter mais informações, consulte the section called “Acesso à Internet para funções da VPC”.

Se você usa endpoints da VPC, também deve configurá-los para habilitar nomes DNS privados.

Quando você cria um mapeamento da origem do evento para um cluster do Amazon DocumentDB, o Lambda verifica se as interfaces de rede elástica (ENIs) já estão presentes nas sub-redes e nos grupos de segurança da VPC do seu cluster. Se o Lambda encontrar ENIs existentes, ele tentará reutilizá-las. Caso contrário, o Lambda criará novas ENIs para se conectar à origem do evento e invocar sua função.

> **Note**
>
> As funções do Lambda sempre são executadas em VPCs de propriedade do serviço Lambda. Essas VPCs recebem manutenção automática do serviço e não são visíveis para os clientes. Você também pode conectar sua função a uma Amazon VPC. Em ambos os casos, a configuração de VPC da sua função não afetará o mapeamento da origem do evento. Somente a configuração da VPC da origem de eventos determina o modo de conexão do Lambda à sua origem de eventos.

Regras de grupos de segurança da VPC

Configure com as seguintes regras (no mínimo) os grupos de segurança da Amazon VPC que contêm seu cluster:

- Regras de entrada: permita todo o tráfego na porta do cluster do Amazon DocumentDB para os grupos de segurança especificados para sua origem do evento. O Amazon DocumentDB usa a porta 27017 por padrão.
- Regras de saída: permitir todo o tráfego na porta 443 para todos os destinos. Permita todo o tráfego na porta do cluster do Amazon DocumentDB. O Amazon DocumentDB usa a porta 27017 por padrão.
- Se você estiver usando endpoints da VPC em vez de um gateway NAT, os grupos de segurança associados aos endpoints da VPC deverão permitir todo o tráfego de entrada na porta 443 dos grupos de segurança da fonte de eventos.

Trabalhar com endpoints da VPC

Quando você usa endpoints da VPC, as chamadas de API para invocar sua função são roteadas por esses endpoints usando as ENIs. A entidade principal do serviço Lambda precisa chamar `lambda:InvokeFunction` para quaisquer funções que usem essas ENIs.

Por padrão, os endpoints da VPC têm políticas do IAM que são abertas. A prática recomendada é restringir essas políticas a fim de permitir somente que entidades principais específicas executem as ações necessárias usando esse endpoint. Para garantir que seu mapeamento da origem do evento seja capaz de invocar sua função do Lambda, a política de endpoint da VPC deve permitir que a entidade principal do serviço Lambda chame `lambda:InvokeFunction`. A restrição de suas políticas de endpoint da VPC para apenas permitir chamadas de API originadas em sua organização impedirá o funcionamento adequado do mapeamento da origem do evento.

O seguinte exemplo de políticas de endpoint da VPC mostra como conceder o acesso necessário aos endpoints do Lambda.

Example Política de endpoint da VPC: endpoint do Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
```

```
        }
    ]
}
```

Se o seu cluster do Amazon DocumentDB usar autenticação, você também poderá restringir a política de endpoint da VPC para o endpoint do Secrets Manager. Para chamar a API do Secrets Manager, o Lambda usa seu perfil de função, e não a entidade principal de serviço do Lambda. O exemplo a seguir mostra uma política de endpoint do Secrets Manager.

Example Política de endpoint da VPC: endpoint do Secrets Manager.

```
{
    "Statement": [
        {
            "Action": "secretsmanager:GetSecretValue",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "customer_function_execution_role_arn"
                ]
            },
            "Resource": "customer_secret_arn"
        }
    ]
}
```

Criar um mapeamento da origem do evento do Amazon DocumentDB (console)

Para que uma função do Lambda faça leituras de um fluxo de alterações de um cluster do Amazon DocumentDB, crie um mapeamento da origem do evento. Esta seção descreve como fazer isso pelo console do Lambda. Para obter o AWS SDK e instruções da AWS CLI, consulte the section called “Criar um mapeamento da origem do evento do Amazon DocumentDB (SDK ou CLI)”.

Para criar um mapeamento da origem do evento do Amazon DocumentDB (console)

1. Abra a página Funções do console do Lambda.
2. Escolha o nome de uma função.
3. Em Visão geral da função, escolha Adicionar gatilho.
4. Em Configuração do acionador, na lista suspensa, escolha DocumentDB.
5. Configure as opções necessárias e escolha Add (Adicionar).

O Lambda oferece suporte às seguintes opções de origem do evento do Amazon DocumentDB:

- Cluster do DocumentDB: selecione um cluster do Amazon DocumentDB.
- Ativar acionador: escolha se você deseja ativar o acionador imediatamente. Se você marcar esta caixa de seleção, a função começará imediatamente a receber tráfego do fluxo de alterações do Amazon DocumentDB especificado após a criação do mapeamento da origem do evento. Recomendamos desmarcar a caixa de seleção para criar o mapeamento da origem do evento em um estado desativado para testes. Após a criação, é possível ativar o mapeamento da origem do evento a qualquer momento.
- Nome do banco de dados: insira o nome de um banco de dados do cluster a ser consumido.
- (Opcional) Nome da coleção: insira o nome de uma coleção no banco de dados a ser consumida. Se você não especificar uma coleção, o Lambda receberá todos os eventos de cada coleção no banco de dados.
- Tamanho do lote: defina o número máximo de mensagens a serem recuperadas em um único lote, até 10 mil. O tamanho padrão do lote é 100.
- Posição inicial: escolha a posição no fluxo para começar a ler registros.
  - Mais recente: processe somente novos registros adicionados ao fluxo. Sua função começa a processar registros somente depois que o Lambda termina de criar a origem do evento. Isso significa que alguns registros podem ser descartados até que a origem do evento seja criada com sucesso.
  - Trim horizon (Redução horizontal): processe todos os registros na transmissão. O Lambda usa a duração da retenção de logs do cluster para determinar por onde começar a ler os eventos. Especificamente, o Lambda começa a ler a partir de `current_time - log_retention_duration`. O fluxo de alterações já deve estar ativo antes desse carimbo de data e hora para que o Lambda leia corretamente todos os eventos.
  - At timestamp (Na data e hora): processe registros a partir de uma hora específica. O fluxo de alterações já deve estar ativo antes do carimbo de data e hora especificado para que o Lambda leia corretamente todos os eventos.
- Autenticação: escolha o método de autenticação para acessar os agentes do Kafka no cluster.
  - BASIC_AUTH: com a autenticação básica, é necessário fornecer a chave do Secrets Manager que contém as credenciais para acessar o cluster.
- Chave do Secrets Manager: escolha a chave do Secrets Manager que contém os detalhes de autenticação (nome de usuário e senha) necessários para acessar o cluster do Amazon DocumentDB.

- (Opcional) Janela de lote: defina o tempo máximo em segundos para reunir registros antes de invocar a função, até 300.
- (Opcional) Configuração completa do documento: para operações de atualização de documentos, escolha o que você deseja enviar ao fluxo. O valor padrão é `Default`, o que significa que, para cada evento de fluxo de alterações, o Amazon DocumentDB enviará somente um delta descrevendo as alterações feitas. Para obter mais informações sobre esse campo, consulte FullDocument na documentação da API do Javadoc do MongoDB.
  - Padrão: o Lambda envia somente um documento parcial descrevendo as alterações feitas.
  - UpdateLookup: o Lambda envia um delta descrevendo as alterações, junto com uma cópia de todo o documento.

Criar um mapeamento da origem do evento do Amazon DocumentDB (SDK ou CLI)

Para criar ou gerenciar um mapeamento da origem do evento do Amazon DocumentDB com um AWS SDK, você pode usar as seguintes operações de API:

- CreateEventSourceMapping
- ListEventSourceMappings
- GetEventSourceMapping
- UpdateEventSourceMapping
- DeleteEventSourceMapping

Para criar o mapeamento de origem de eventos com a AWS CLI, use o comando `create-event-source-mapping`. O exemplo a seguir usa esse comando para mapear uma função denominada `my-function` para um fluxo de alterações do Amazon DocumentDB. A origem do evento é especificada por um nome do recurso da Amazon (ARN), com um tamanho de lote de 500, a partir do timestamp no horário do Unix. O comando também especifica a chave do Secrets Manager que o Lambda usa para se conectar ao Amazon DocumentDB. Além disso, ele inclui parâmetros `document-db-event-source-config` que especificam o banco de dados e a coleção de onde serão feitas as leituras.

```
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"}' \
```

Você deve ver uma saída semelhante a:

```
{
    "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"
}
```

Após a criação, você pode usar o comando `update-event-source-mapping` para atualizar as configurações da origem do evento do Amazon DocumentDB. O exemplo a seguir atualiza o tamanho do lote para 1 mil e a janela do lote para dez segundos. Para esse comando, é necessário o UUID do mapeamento da origem do evento, que pode ser recuperado usando o comando `list-event-source-mapping` ou o console do Lambda.

```
aws lambda update-event-source-mapping --function-name my-function \
    --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
    --batch-size 1000 \
    --batch-window 10
```

Você deve ver uma saída semelhante a:

```
{
```

```
    "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"
}
```

O Lambda atualiza as configurações de maneira assíncrona. Talvez você não veja essas alterações na saída até que o processo seja concluído. Para visualizar as configurações atuais do mapeamento da origem do evento, use o comando `get-event-source-mapping`.

```
aws lambda get-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b
```

Você deve ver uma saída semelhante a:

```
{
    "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"
}
```

Para excluir o mapeamento da origem do evento do Amazon DocumentDB, use o comando `delete-event-source-mapping`.

```
aws lambda delete-event-source-mapping \
    --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284
```

Posições iniciais de sondagem e fluxo

Esteja ciente de que a sondagem do fluxo durante a criação e as atualizações do mapeamento da origem do evento é, finalmente, consistente.

- Durante a criação do mapeamento da origem do evento, pode levar alguns minutos para a sondagem de eventos do fluxo iniciar.
- Durante as atualizações do mapeamento da origem do evento, pode levar alguns minutos para interromper e reiniciar a sondagem de eventos do fluxo.

Esse comportamento significa que, se você especificar `LATEST` como posição inicial do fluxo, o mapeamento da origem do evento poderá perder eventos durante a criação ou as atualizações. Para garantir que nenhum evento seja perdido, especifique a posição inicial do fluxo como `TRIM_HORIZON` ou `AT_TIMESTAMP`.

Monitorar a origem do evento do Amazon DocumentDB

Para ajudar você a monitorar a origem do evento do Amazon DocumentDB, o Lambda gera a métrica `IteratorAge` quando a função termina de processar um lote de registros. Idade do iterador é a diferença entre o carimbo de data e hora do evento mais recente e o carimbo de data e hora atual. Essencialmente a métrica `IteratorAge` indica a idade do último registro processado no lote. Se a função atualmente estiver processando novos eventos, você poderá usar a idade do iterador para estimar a latência entre quando um registro é adicionado e quando a função o processa. Uma tendência crescente em `IteratorAge` pode indicar problemas com a função. Para ter mais informações, consulte Trabalhar com métricas de funções Lambda.

Os fluxos de alterações do Amazon DocumentDB não são otimizados para processar grandes intervalos de tempo entre os eventos. Se sua origem de eventos do Amazon DocumentDB não receber nenhum evento por um longo período de tempo, o Lambda poderá desabilitar o mapeamento da origem do evento. A duração desse período pode variar de algumas semanas a alguns meses, dependendo do tamanho do cluster e de outras workloads.

O Lambda é compatível com cargas úteis de até 6 MB. Entretanto, os eventos de fluxo de alterações do Amazon DocumentDB podem ter até 16 MB. Se seu fluxo de alterações tentar enviar ao Lambda um evento de fluxo de alterações maior que 6 MB, o Lambda descartará a mensagem e emitirá a métrica `OversizedRecordCount`. O Lambda emite todas as métricas com base no melhor esforço.

Tutorial: uso do AWS Lambda com o Amazon DocumentDB Streams

Neste tutorial, você criará uma função do Lambda que consome eventos de um stream de alterações do Amazon DocumentDB (compativel com MongoDB). Para concluir este tutorial, você passará pelos seguintes estágios:

- Configure seu cluster Amazon DocumentDB, conecte-se a ele e ative streams de alterações nele.
- Crie a função do Lambda e configure seu cluster do Amazon DocumentDB como uma origem de eventos para a sua função.
- Teste a configuração de ponta a ponta inserindo itens no seu banco de dados do Amazon DocumentDB.

Tópicos

- Teste sua função excluindo um registro
- Limpe os recursos

Pré-requisitos

Cadastre-se em uma Conta da AWS

Se você ainda não tem Conta da AWS, siga as etapas a seguir para criar uma.

Para cadastrar-se em uma Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

   Quando você se cadastra em uma Conta da AWS, um Usuário raiz da conta da AWS é criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como prática recomendada de segurança, atribua o acesso administrativo a um usuário e use somente o usuário-raiz para executar tarefas que exigem acesso de usuário-raiz.

AWS envia um e-mail de confirmação depois que o processo de cadastramento é concluído. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se cadastrar em uma Conta da AWS, proteja seu Usuário raiz da conta da AWS, habilite o AWS IAM Identity Center e crie um usuário administrativo para não usar o usuário raiz em tarefas cotidianas.

Proteja seu Usuário raiz da conta da AWS

1. Faça login no AWS Management Console como o proprietário da conta ao selecionar a opção Root user (Usuário raiz) e inserir o endereço de e-mail da Conta da AWS. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário raiz, consulte Signing in as the root user (Fazer login como usuário raiz) no Guia do usuárioInício de Sessão da AWS.

2. Ative a autenticação multifator (MFA) para seu usuário raiz.

   Para obter instruções, consulte Habilitar um dispositivo MFA virtual para o usuário raiz (console)Conta da AWS no Guia do Usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center.

2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para obter um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso dos usuários com o Diretório do Centro de Identidade do IAM padrão no Guia do usuário do AWS IAM Identity Center.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário IAM Identity Center, use a URL de login enviada ao seu endereço de e-mail quando você criou o usuário IAM Identity Center user.

  Para obter ajuda com o login utilizando um usuário do IAM Identity Center, consulte Fazendo login no portal de acesso da AWS, no Guia do UsuárioInício de Sessão da AWS.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center.

2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center.

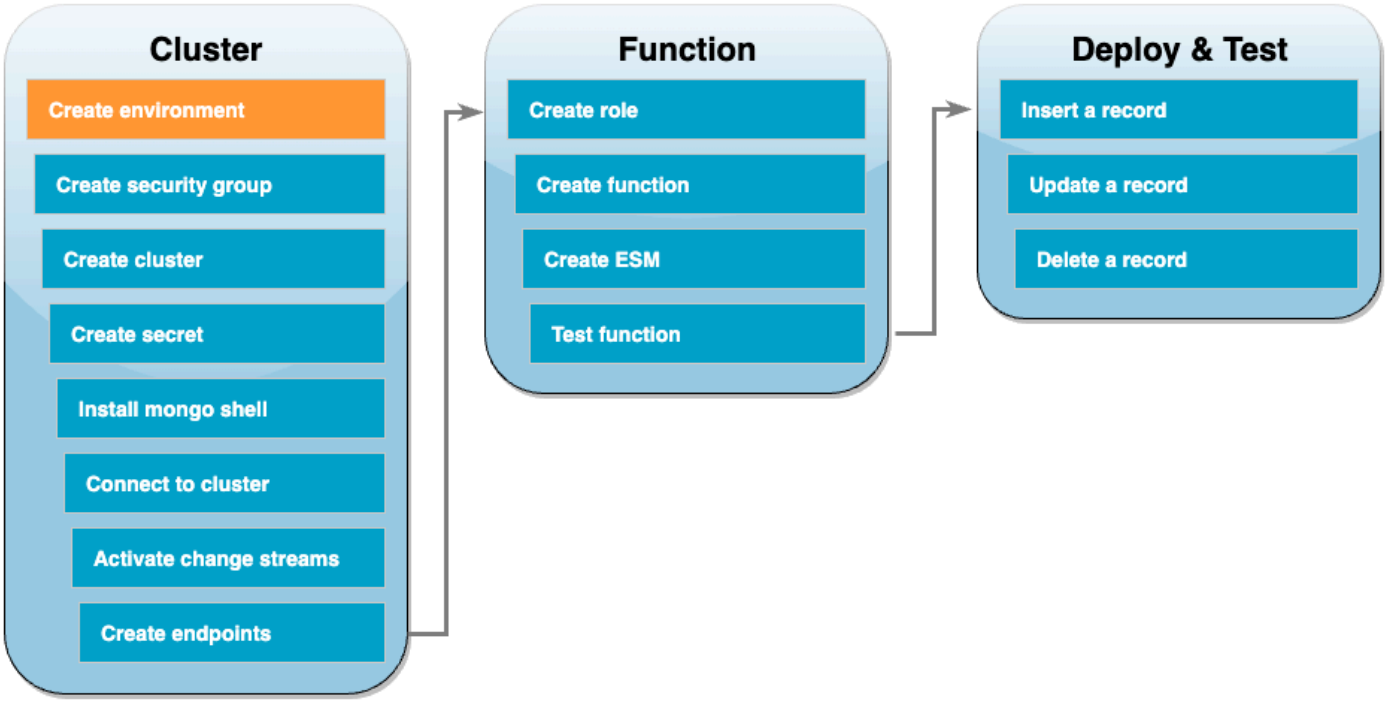
Instalar a AWS Command Line Interface

Se você ainda não instalou a AWS Command Line Interface, siga as etapas em Instalar ou atualizar a versão mais recente da AWS CLI para instalá-la.

O tutorial requer um terminal de linha de comando ou um shell para executar os comandos. No Linux e no macOS, use o gerenciador de pacotes e de shell de sua preferência.

> **Note**
>
> No Windows, alguns comandos da CLI do Bash que você costuma usar com o Lambda (como `zip`) não são compatíveis com os terminais integrados do sistema operacional. Para obter uma versão do Ubuntu com o Bash integrada no Windows, instale o Subsistema do Windows para Linux.

Criar o ambiente do AWS Cloud9



Antes de criar a função do Lambda, é necessário criar e configurar seu cluster do Amazon DocumentDB. As etapas para configurar seu cluster neste tutorial são baseadas no procedimento em Comece a usar o Amazon DocumentDB.

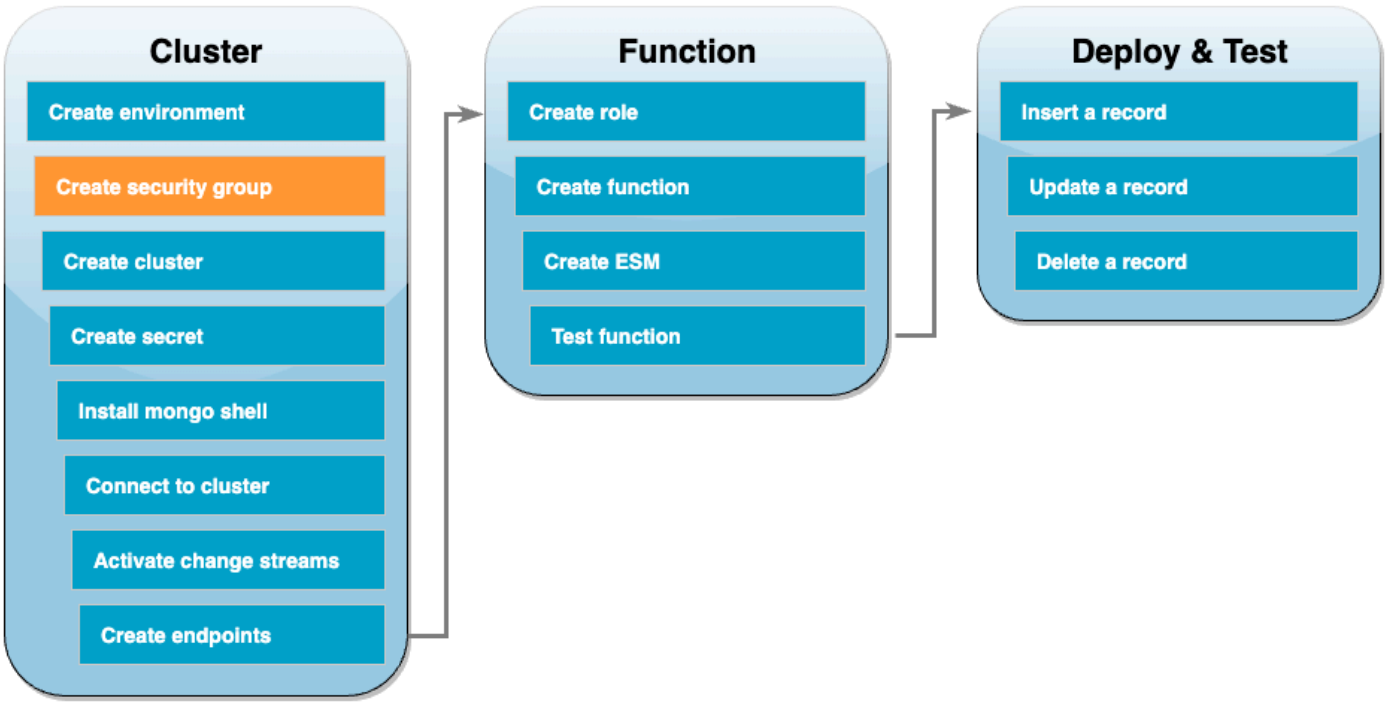
> **Note**
>
> Se você já tem um cluster do Amazon DocumentDB configurado, ative os streams de alterações e crie os endpoints da VPC de interface necessários. Em seguida, é possível pular diretamente para as etapas de criação da função.

Primeiro, crie um ambiente do AWS Cloud9. Você usará esse ambiente ao longo deste tutorial para se conectar e consultar seu cluster do DocumentDB.

Para criar um ambiente do AWS Cloud9

1. Abra o console do Cloud9 e escolha Criar ambiente.
2. Crie um ambiente com a seguinte configuração:
   - Em Detalhes:
     - Nome: `DocumentDBCloud9Environment`
     - Tipo de ambiente: nova instância do EC2
   - Em Nova instância do EC2:
     - Tipo de instância: `t2.micro` (1 GiB RAM + 1 vCPU)
     - Plataforma: Amazon Linux 2
     - Tempo limite: 30 minutos
   - Em Configurações de rede:
     - Conexão: AWS Systems Manager (SSM)
     - Expanda o menu suspenso Configurações da VPC.
     - Nuvem privada virtual (VPC) da Amazon: escolha sua VPC padrão.
     - Sub-rede: sem preferências
   - Mantenha todas as outras configurações como o padrão.
3. Escolha Criar. O provisionamento do seu novo ambiente do AWS Cloud9 pode levar alguns minutos.

Criar o grupo de segurança do EC2



Em seguida, crie um grupo de segurança do EC2 com regras que permitam o tráfego entre seu cluster do DocumentDB e seu ambiente Cloud9.

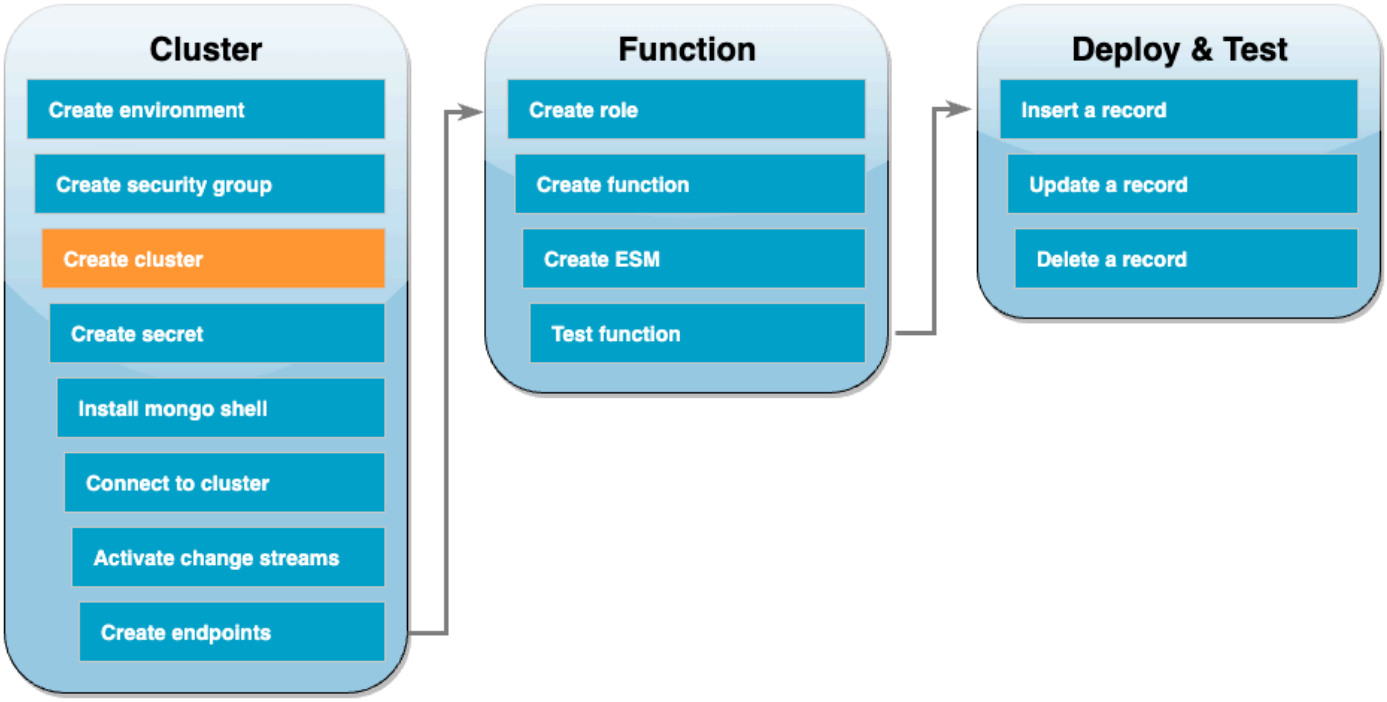
Para criar um grupo de segurança do EC2

1. Abra o console do EC2. Em Rede e segurança, escolha Grupos de segurança.
2. Escolha Create security group (Criar grupo de segurança).
3. Crie um grupo de segurança com a seguinte configuração:
   - Em Detalhes básicos:
     - Nome do grupo de segurança: `DocDBTutorial`
     - Descrição: Grupo de segurança para tráfego entre o Cloud9 e o DocumentDB.
     - VPC: escolha sua VPC padrão.
   - Em Inbound rules (Regras de entrada), escolha Add rule (Adicionar regra). Crie uma regra com a seguinte configuração:
     - Tipo: TCP personalizado
     - Intervalo de portas: 27017
     - Origem: personalizada

- Na caixa de pesquisa ao lado de Origem, escolha o grupo de segurança para o ambiente do AWS Cloud9 que você criou na etapa anterior. Para ver uma lista dos grupos de segurança disponíveis, insira `cloud9` na caixa de pesquisa. Escolha o grupo de segurança com o nome `aws-cloud9-<environment_name>`.
- Mantenha todas as outras configurações como o padrão.

4. Escolha Create security group (Criar grupo de segurança).

Crie o cluster do DocumentDB



Nesta etapa, você criará um cluster do DocumentDB usando o grupo de segurança da etapa anterior.

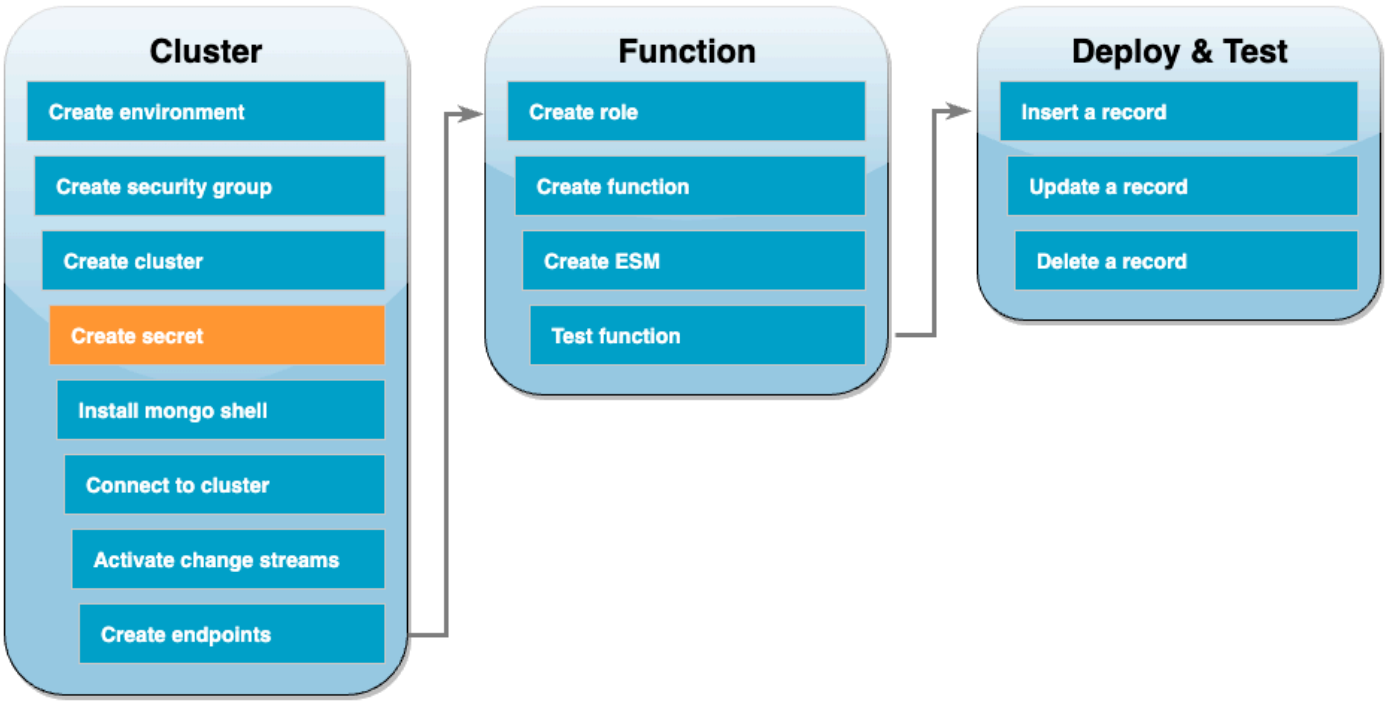
Para criar um cluster do DocumentDB

1. Abra o console do DocumentDB. Em Clusters, escolha Criar.
2. Crie um cluster com a seguinte configuração:

   - Em Tipo de cluster, escolha Cluster baseado em instância.
   - Em Configuração:
     - Versão do mecanismo: 5.0.0
     - Classe de instância: db.t3.medium (teste gratuito qualificado)

    - Número de instâncias: 1.
  - Em Autenticação:
    - Insira o Nome de usuário e a Senha necessários para se conectar ao seu cluster (as mesmas credenciais que você usou para criar o segredo na etapa anterior). Em Confirmar senha, confirme sua senha.
  - Ative Exibir configurações avançadas.
  - Em Configurações de rede:
    - Nuvem privada virtual (VPC): escolha sua VPC padrão.
    - Grupo da sub-rede: padrão
    - Grupos de segurança da VPC: além de `default (VPC)`, escolha o grupo de segurança `DocDBTutorial (VPC)` que você criou na etapa anterior.
  - Mantenha todas as outras configurações como o padrão.
3. Selecione Criar cluster. O provisionamento do seu cluster do DocumentDB pode levar vários minutos.

Criar o segredo no Secrets Manager



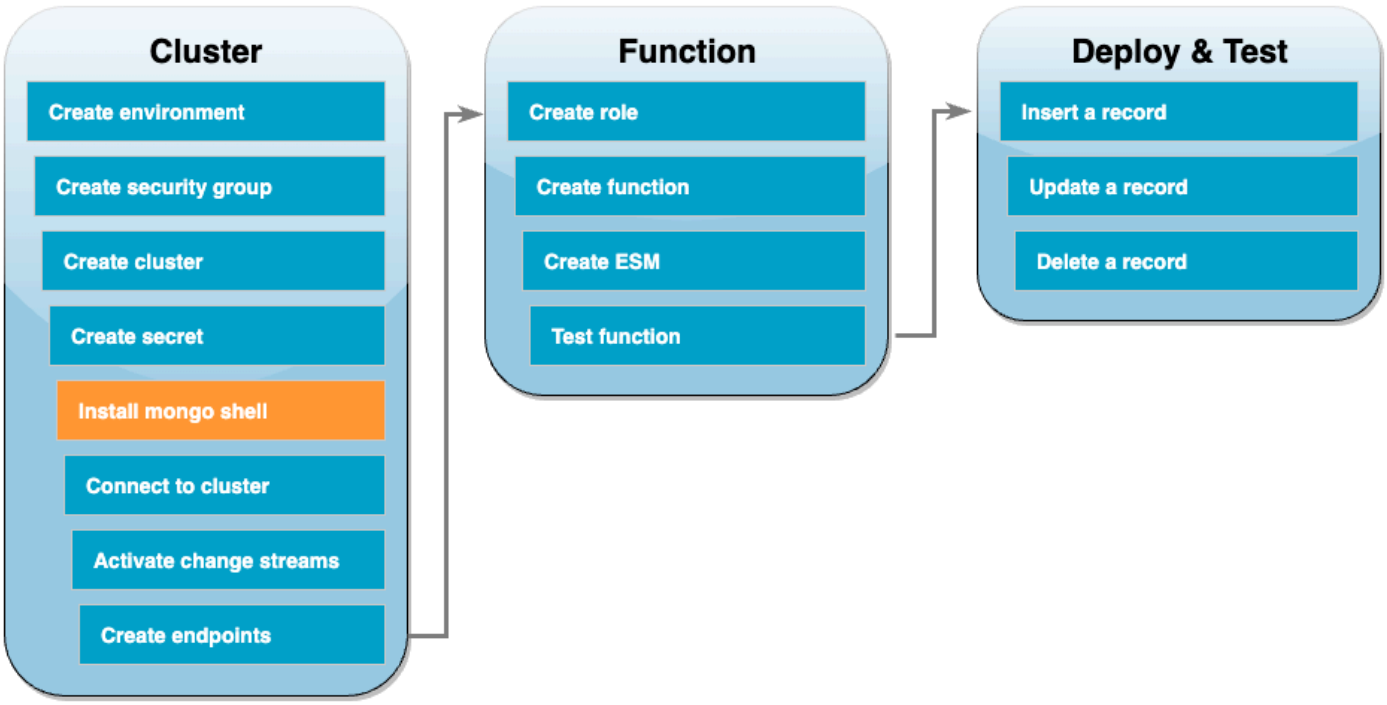
Para acessar seu cluster do DocumentDB manualmente, é necessário fornecer credenciais de nome de usuário e senha. Para que o Lambda acesse seu cluster, é necessário informar um secredo do Secrets Manager que contenha essas mesmas credenciais de acesso ao configurar seu mapeamento da origem do evento. Nesta etapa, você criará esse segredo.

Para criar o segredo no Secrets Manager

1. Abra o console do Secrets Manager e escolha Armazenar um novo segredo.
2. Em Escolha o tipo de segredo, selecione as seguintes opções:
   - Em Detalhes básicos:
     - Tipo de segredo: credenciais para o banco de dados do Amazon DocumentDB
     - Em Credenciais, insira o nome de usuário e a senha que você usará para acessar seu cluster DocumentDB.
     - Banco de dados: escolha seu cluster do DocumentDB.
     - Escolha Próximo.
3. Em Configurar segredo, escolha as seguintes opções:
   - Nome de segredo: `DocumentDBSecret`
   - Escolha Próximo.
4. Escolha Próximo.
5. Escolha Armazenar.
6. Atualize o console para verificar se você armazenou o segredo `DocumentDBSecret` com êxito.

Anote o ARN do segredo do seu segredo. Você precisará dele em uma etapa posterior.

Instale o shell do mongo



Nesta etapa, você instalará o shell do Mongo em seu ambiente do Cloud9. O shell do Mongo é um utilitário de linha de comando que você usa para se conectar e consultar seu cluster do DocumentDB.

Para instalar o shell mongo em seu ambiente do Cloud9

1. Abra o console do Cloud9. Ao lado do ambiente `DocumentDBCloud9Environment` que você criou anteriormente, clique no link Abrir abaixo da coluna IDE do Cloud9.
2. Na janela do terminal, crie o arquivo do repositório do MongoDB com o seguinte comando:

```
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
```

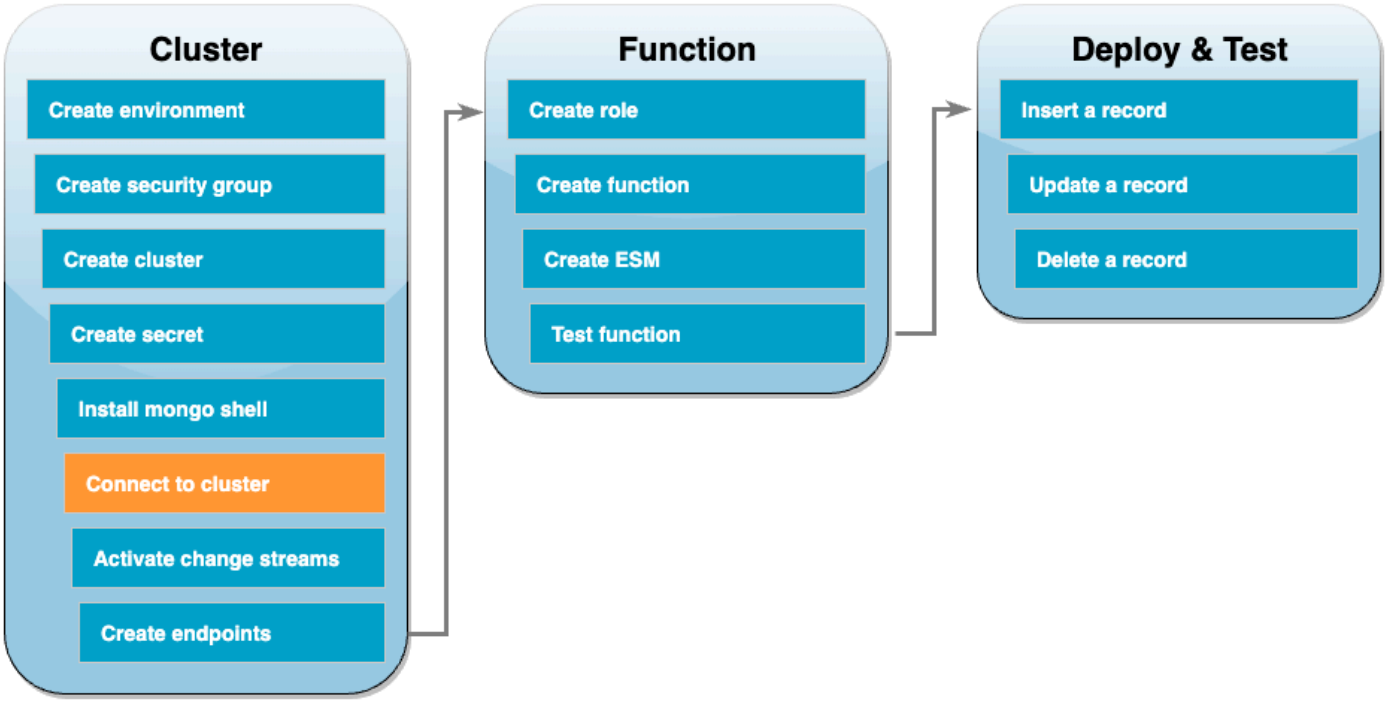
3. Em seguida, instale o shell do Mongo com o seguinte comando:

```
sudo yum install -y mongodb-org-shell
```

4. Para criptografar dados em trânsito, faça download da chave pública do Amazon DocumentDB. O comando a seguir faz download de um arquivo de nome `global-bundle.pem`:

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

Conectar ao cluster do DocumentDB



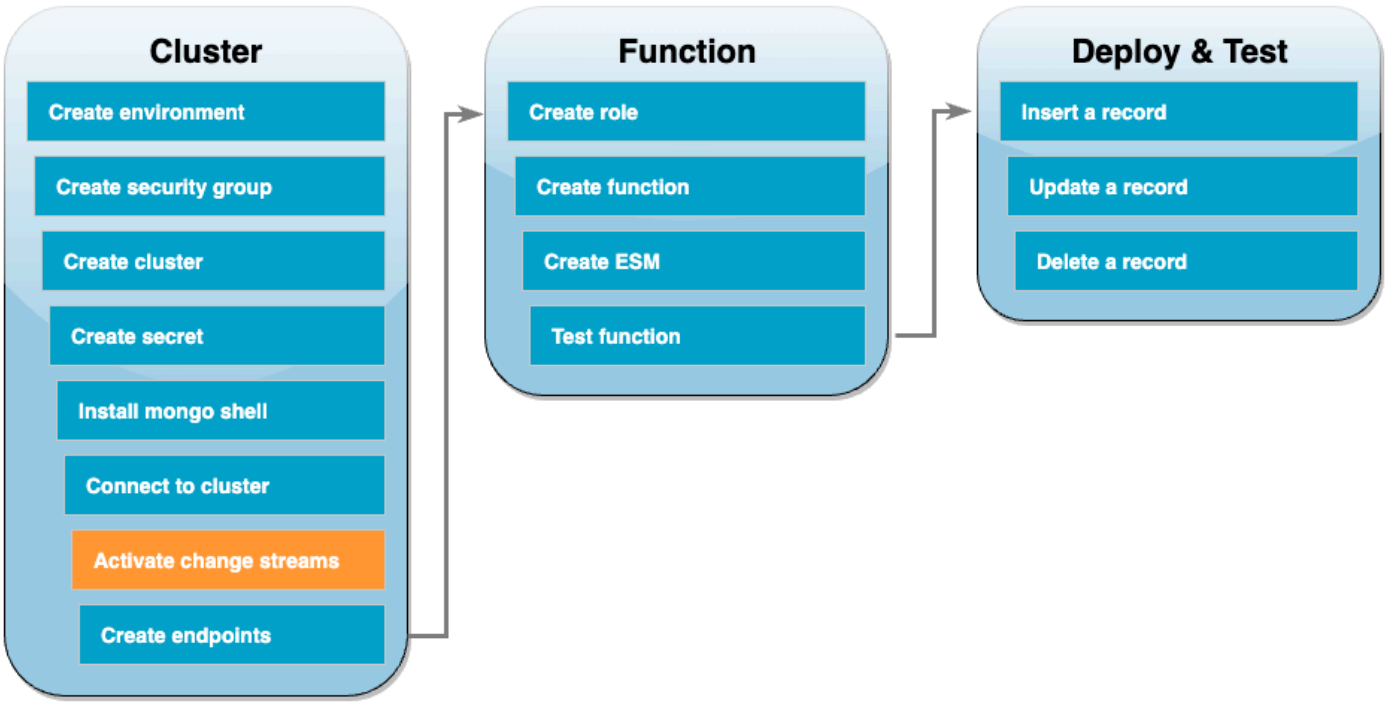
Agora você está pronto para se conectar ao seu cluster do DocumentDB usando o shell do Mongo.

Para se conectar ao seu cluster do DocumentDB

1. Abra o console do DocumentDB. Em Clusters, escolha seu cluster escolhendo seu identificador de cluster.
2. Na guia Conectividade e segurança, em Se conectar a este cluster com o shell do Mongo, escolha Copiar.
3. Em seu ambiente do Cloud9, cole esse comando no terminal. Substitua `<insertYourPassword>` pela senha correta.

Depois de inserir esse comando, se o prompt de comando se tornar `rs0:PRIMARY>`, você estará conectado ao seu cluster do Amazon DocumentDB.

Ativar os streams de alteração



Neste tutorial, você acompanhará as alterações na coleção de `products` do banco de dados `docdbdemo` em seu cluster do DocumentDB. Você faz isso com a ativação de streams de alterações. Primeiro, crie o banco de dados `docdbdemo` e teste-o, inserindo um registro.

Para criar um novo banco de dados em seu cluster

1. Em seu ambiente do Cloud9, certifique-se de que você ainda esteja conectado ao seu cluster do DocumentDB.
2. Na janela do terminal, use o comando a seguir para criar um novo banco de dados chamado `docdbdemo`:

   ```
   use docdbdemo
   ```

3. Em seguida, use o comando a seguir para inserir um registro no `docdbdemo`:

   ```
   db.products.insert({"hello":"world"})
   ```

   Você deve ver uma saída semelhante a:

```
WriteResult({ "nInserted" : 1 })
```

4. Use o comando a seguir para listar todos os bancos de dados:

```
show dbs
```

Certifique-se de que sua saída contenha o banco de dados `docdbdemo`:

```
docdbdemo  0.000GB
```

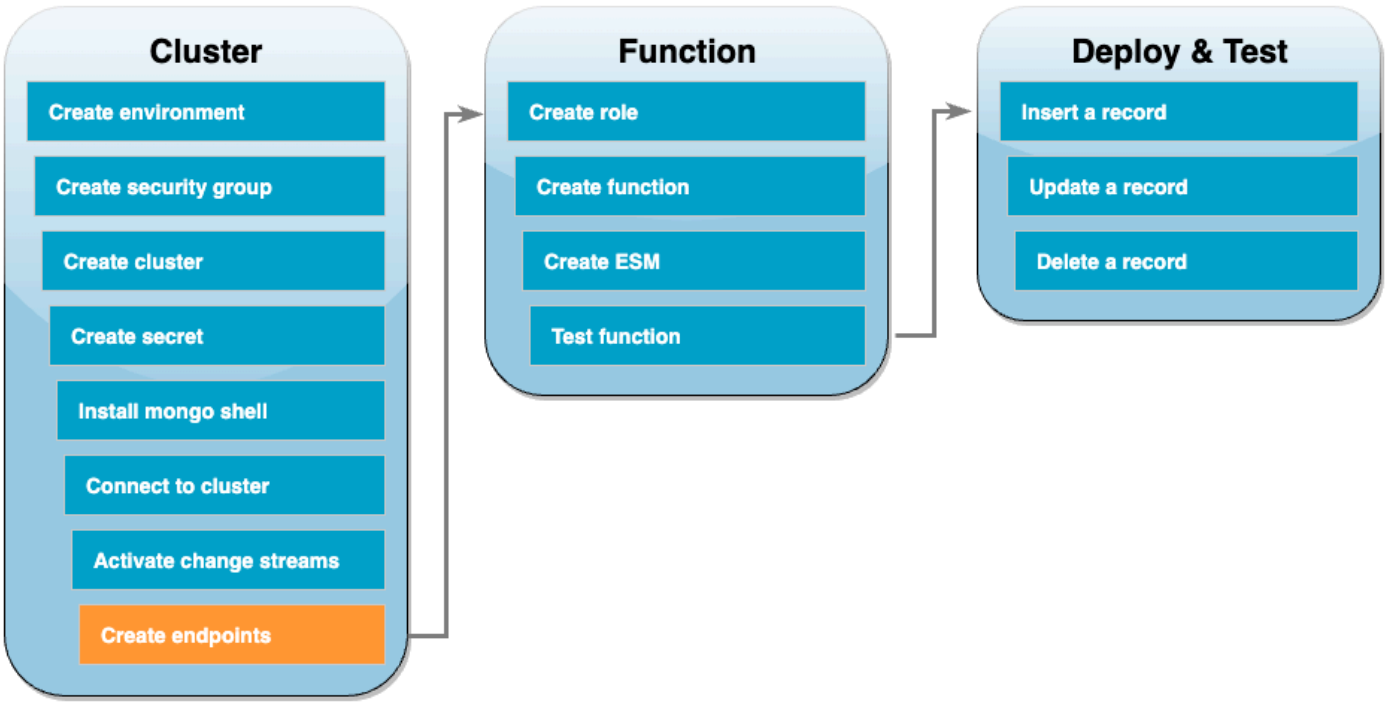
Em seguida, ative os streams de alterações na coleção de `products` do banco de dados `docdbdemo` usando o seguinte comando:

```
db.adminCommand({modifyChangeStreams: 1,
    database: "docdbdemo",
    collection: "products",
    enable: true});
```

Você deve ver uma saída semelhante a:

```
{ "ok" : 1, "operationTime" : Timestamp(1680126165, 1) }
```

Criar endpoints da VPC de interface



Em seguida, crie endpoints da VPC de interface para garantir que o Lambda e o Secrets Manager (usados posteriormente para armazenar nossas credenciais de acesso ao cluster) possam se conectar à sua VPC padrão.

Para criar endpoints da VPC de interface

1. Abra o console da VPC. No menu à esquerda, em Nuvem privada virtual, escolha Endpoints.
2. Escolha Criar endpoint. Crie um endpoint com a seguinte configuração:
   - Em Nomear tag, insira `lambda-default-vpc`.
   - Em Categoria do serviço, escolha serviços da AWS.
   - Em Serviços, insira `lambda` na caixa de pesquisa. Escolha o serviço com formato `com.amazonaws.<region>.lambda`.
   - Em VPC, escolha sua VPC padrão.
   - Em sub-redes, marque as caixas ao lado de cada zona de disponibilidade. Escolha O ID correto da sub-rede de cada zona de disponibilidade.
   - Em Tipo de endereço IP, selecione IPv4.

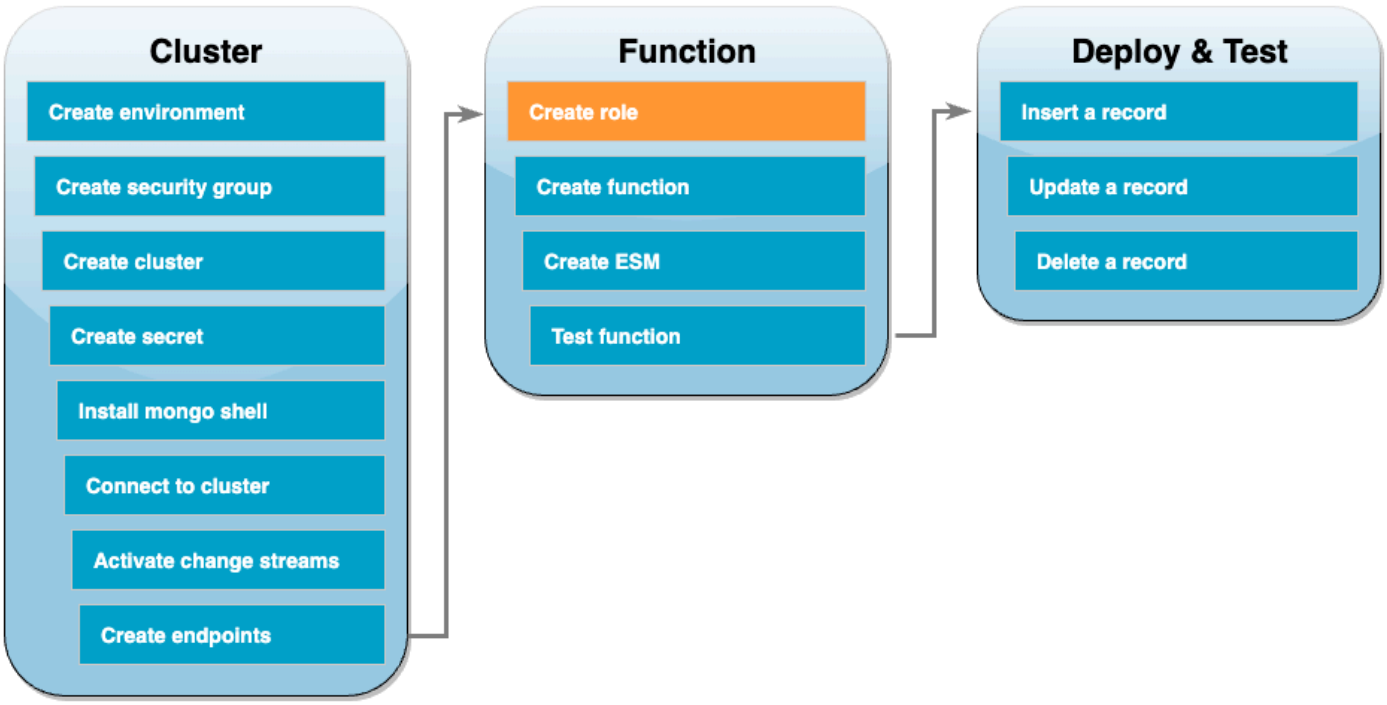
- Em Grupos de segurança, escolha o grupo de segurança da VPC padrão (nome do grupo de `default`) e o grupo de segurança que você criou anteriormente (nome do grupo de `DocDBTutorial`).
- Mantenha todas as outras configurações como o padrão.
- Escolha Criar endpoint.

3. Novamente, escolha Criar endpoint. Crie um endpoint com a seguinte configuração:

   - Em Nomear tag, insira `secretsmanager-default-vpc`.
   - Em Categoria do serviço, escolha serviços da AWS.
   - Em Serviços, insira `secretsmanager` na caixa de pesquisa. Escolha o serviço com formato `com.amazonaws.<region>.secretsmanager`.
   - Em VPC, escolha sua VPC padrão.
   - Em sub-redes, marque as caixas ao lado de cada zona de disponibilidade. Escolha O ID correto da sub-rede de cada zona de disponibilidade.
   - Em Tipo de endereço IP, selecione IPv4.
   - Em Grupos de segurança, escolha o grupo de segurança da VPC padrão (nome do grupo de `default`) e o grupo de segurança que você criou anteriormente (nome do grupo de `DocDBTutorial`).
   - Mantenha todas as outras configurações como o padrão.
   - Escolha Criar endpoint.

Isso conclui a parte de configuração de cluster deste tutorial.

Criar a função de execução



No próximo conjunto de etapas, você criará sua função do Lambda. Primeiro, é necessário criar o perfil de execução que concede à sua função permissão para acessar o seu cluster. Você criará uma política do IAM primeiro e, em seguida, anexará essa política a um perfil do IAM.

Para criar uma política do IAM

1. Abra a página Políticas no console do IAM e escolha Criar política.
2. Selecione a guia JSON. Na política a seguir, substitua o ARN do recurso Secrets Manager na linha final da declaração pelo ARN do seu segredo de antes, e copie a política no 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"
        }
    ]
}
```

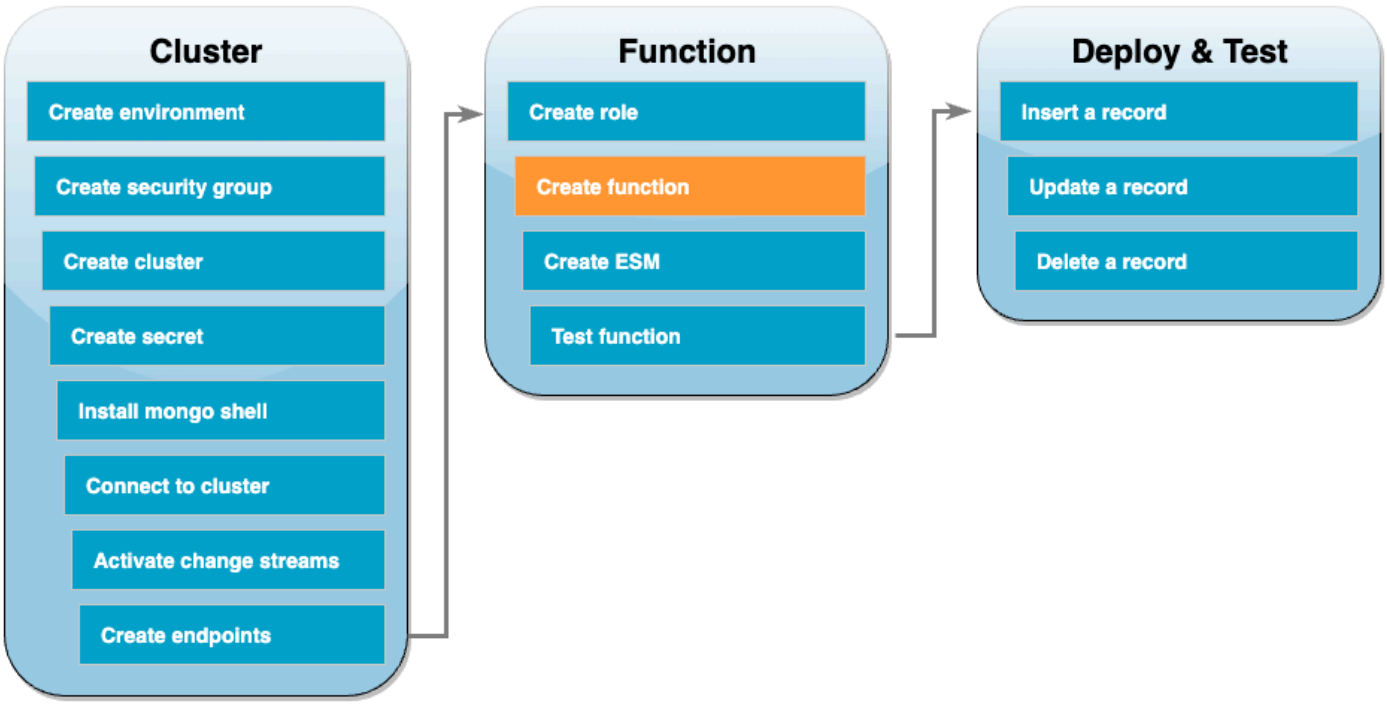
3. Escolha Próximo: Tags, e, em seguida, escolha Próximo: Revisar.
4. Em Nome, insira `AWSDocumentDBLambdaPolicy`.
5. Escolha Criar política.

Como criar o perfil do IAM

1. Abra a página Perfis no console do IAM e escolha Criar perfil.
2. Em Selecionar entidade confiável, escolha as seguintes opções:
   - Tipo de entidade confiável: serviço da AWS
   - Caso de uso: Lambda
   - Escolha Próximo.

3. Em Adicionar permissões, escolha a política `AWSDocumentDBLambdaPolicy` que você acabou de criar, bem como `AWSLambdaBasicExecutionRole` para conceder à sua função permissões para gravar no Amazon CloudWatch Logs.
4. Escolha Próximo.
5. Em Nome do perfil, insira `AWSDocumentDBLambdaExecutionRole`.
6. Escolha Create role (Criar função).

Criar a função do Lambda



O código de exemplo a seguir recebe uma entrada de evento do DocumentDB e processa a mensagem contida.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Amazon DocumentDB com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Amazon DocumentDB com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Amazon DocumentDB com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Amazon DocumentDB com o Lambda usando 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
```

Para criar a função do Lambda

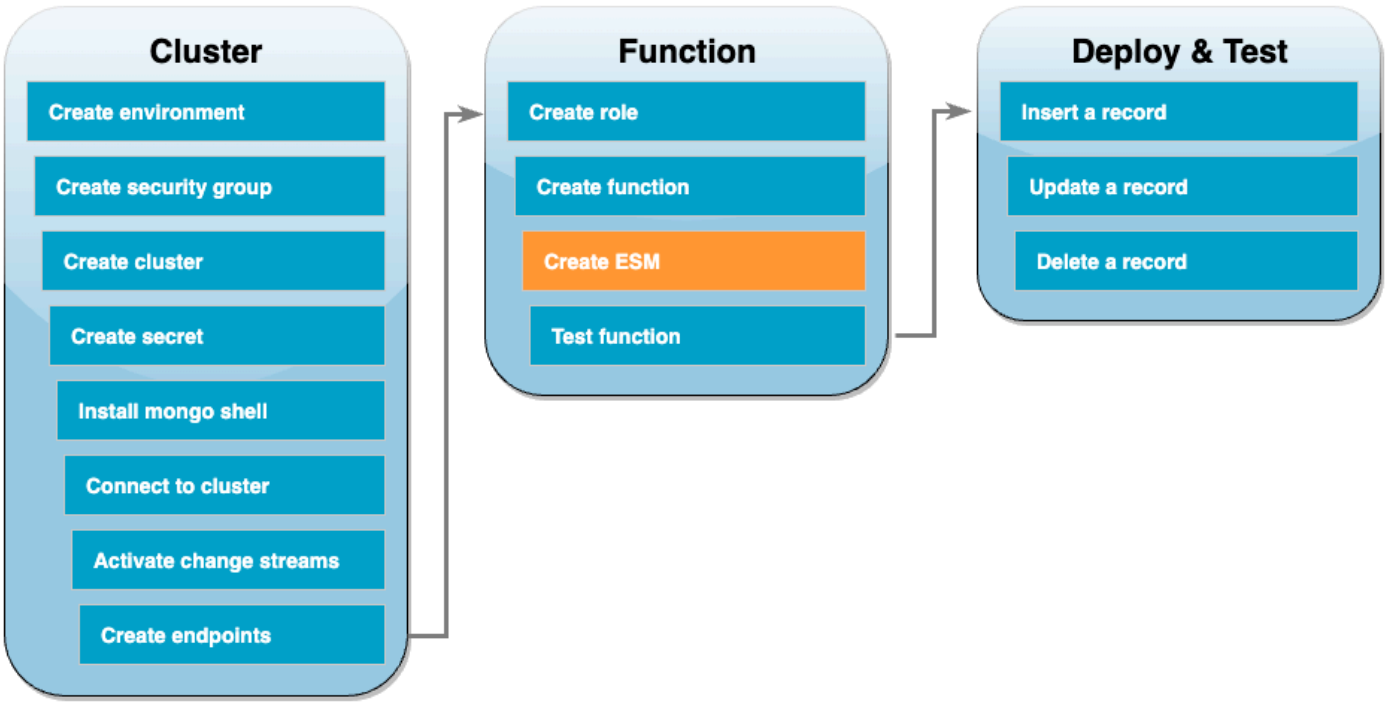
1. Copie o código de amostra em um arquivo chamado `index.js`.
2. Crie um pacote de implantação com o comando a seguir.

```
zip function.zip index.js
```

3. Use o comando a seguir da CLI para criar a função. Substitua `us-east-1` pela região, e `123456789012` pelo seu ID de conta.

```
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
```

Crie o mapeamento da origem do evento do Lambda



Crie o mapeamento da origem do evento que associa o seu stream de alterações do DocumentDB à sua função do Lambda. Depois que você criar esse mapeamento da origem do evento, o AWS Lambda começará imediatamente a sondar o stream.

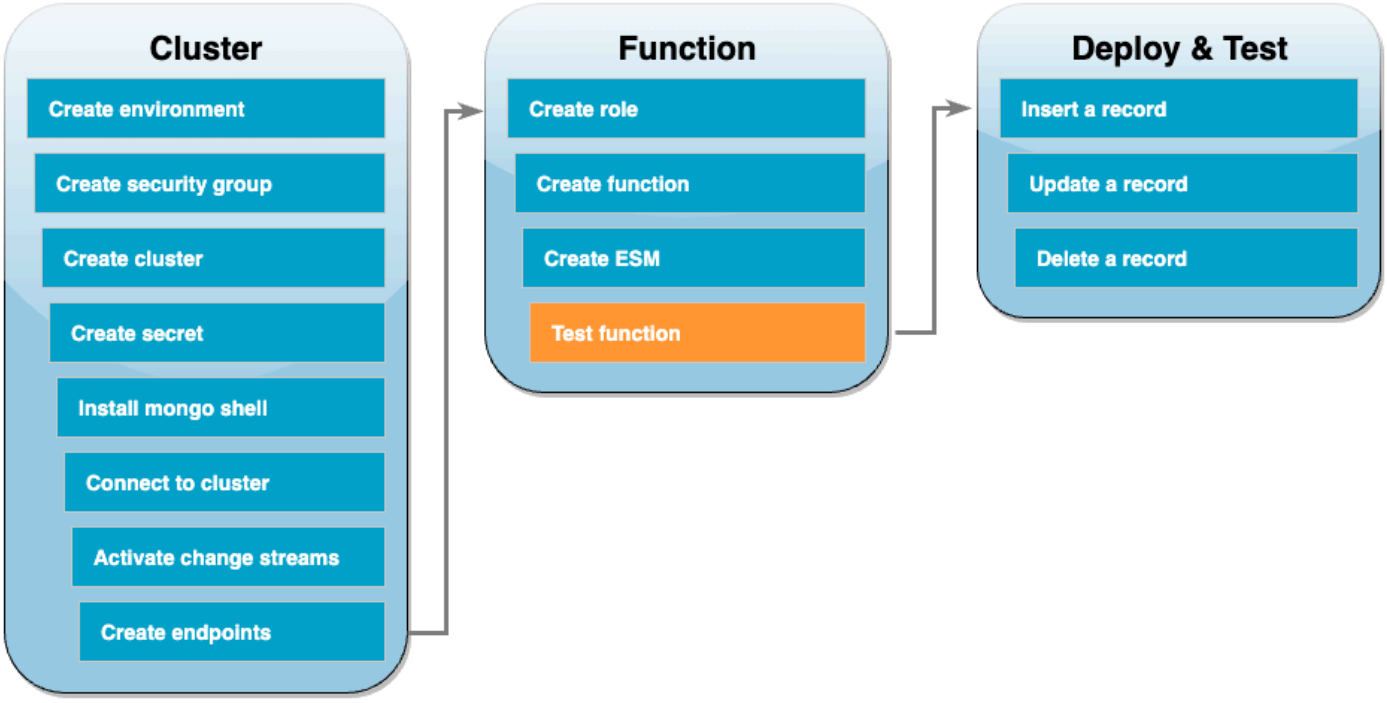
Para criar o mapeamento da origem do evento

1. Abra a página Funções no console do Lambda.
2. Escolha a função `ProcessDocumentDBRecords` criada anteriormente.
3. Escolha a guia Configuração e, em seguida, escolha Acionadores no menu à esquerda.
4. Escolha Add trigger.
5. Em Configuração do acionador, para a origem, selecione DocumentDB.
6. Crie o mapeamento da origem do evento com a seguinte configuração:
   - Cluster do DocumentDB: escolha o cluster que você criou anteriormente.
   - Nome do banco de dados: `docdbdemo`
   - Nome da coleção: Produtos
   - Batch size – 1
   - Posição inicial: última

- Autenticação: BASIC_AUTH
- Chave do Secrets Manager: escolha o `DocumentDBSecret` que você acabou de criar.
- Janela do lote: 1
- Configuração do documento completo: UpdateLookup

7. Escolha Adicionar. A criação do mapeamento da origem do evento pode levar alguns minutos.

Teste sua função com invocação manual



Para testar se você criou corretamente sua função e o mapeamento da origem do evento, invoque sua função usando o comando `invoke`. Para fazer isso, primeiro copie o evento JSON a seguir em um arquivo chamado `input.txt`:

```
{
  "eventSourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:canaryclusterb2a659a2-
qo5tcmqkcl03",
  "events": [
    {
      "event": {
        "_id": {
          "_data": "0163eeb6e700000009010000009000041e1"
        },
```

```
        "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"
}
```

Em seguida, use o comando a seguir para invocar sua função com este evento:

```
aws lambda invoke --function-name ProcessDocumentDBRecords \
    --cli-binary-format raw-in-base64-out \
    --region us-east-1 \
    --payload file://input.txt out.txt
```

Você verá uma resposta parecida com a seguinte:

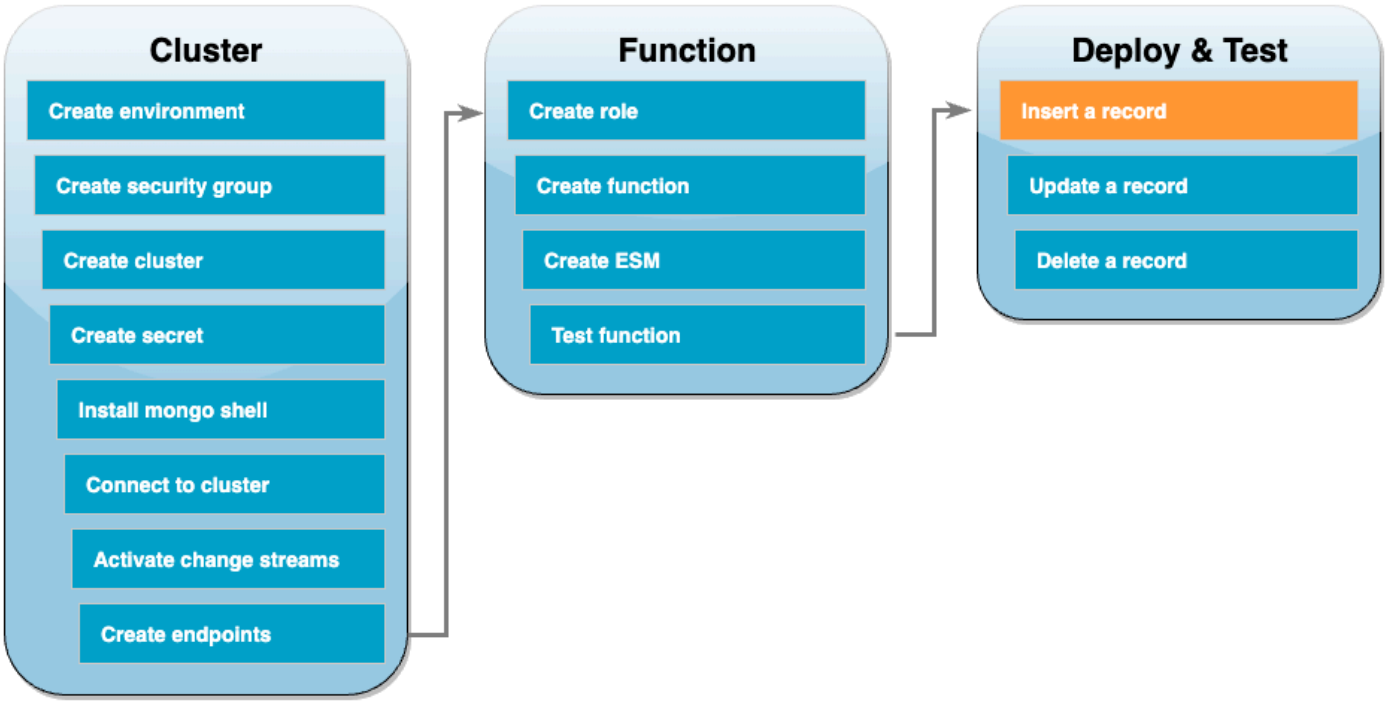
```
{
   "StatusCode": 200,
   "ExecutedVersion": "$LATEST"
}
```

É possível verificar se sua função processou o evento com êxito examinando o CloudWatch Logs.

Para verificar a invocação manual por meio do CloudWatch Logs

1. Abra a página Funções no console do Lambda.
2. Escolha a guia Monitor e selecione Visualizar logs do CloudWatch. Isso leva você ao grupo de logs específico associado à sua função no console do CloudWatch.
3. Escolha o stream de logs mais recente. Nas mensagens de log, você deverá ver o evento JSON.

Teste sua função inserindo um registro



Teste sua configuração de ponta a ponta interagindo diretamente com seu banco de dados do DocumentDB. No próximo conjunto de etapas, você inserirá um registro, o atualizará e o excluirá.

Para inserir um registro

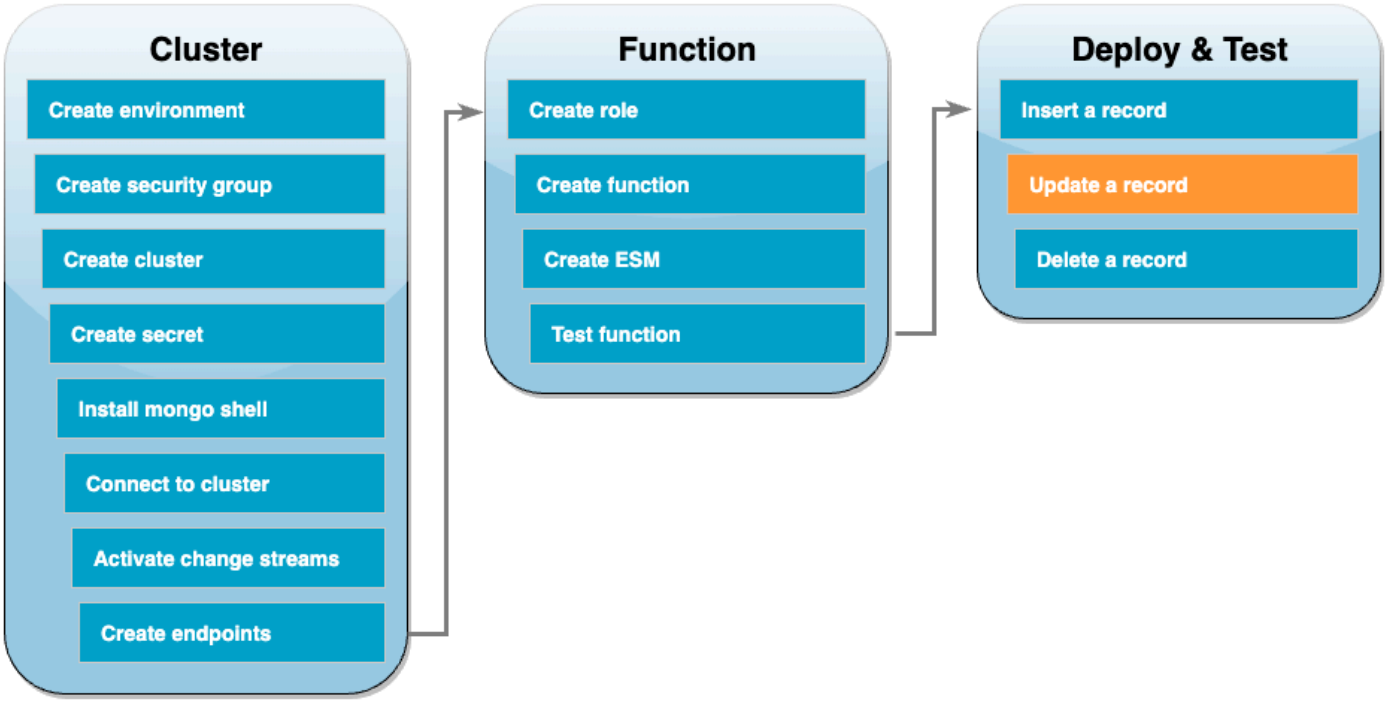
1. Reconecte-se ao seu cluster do DocumentDB em seu ambiente do Cloud9.
2. Use este comando para garantir que você esteja usando o banco de dados `docdbdemo` no momento:

```
use docdbdemo
```

3. Insira um registro na coleção `products` do banco de dados `docdbdemo`:

```
db.products.insert({"name":"Pencil", "price": 1.00})
```

Teste sua função atualizando um registro

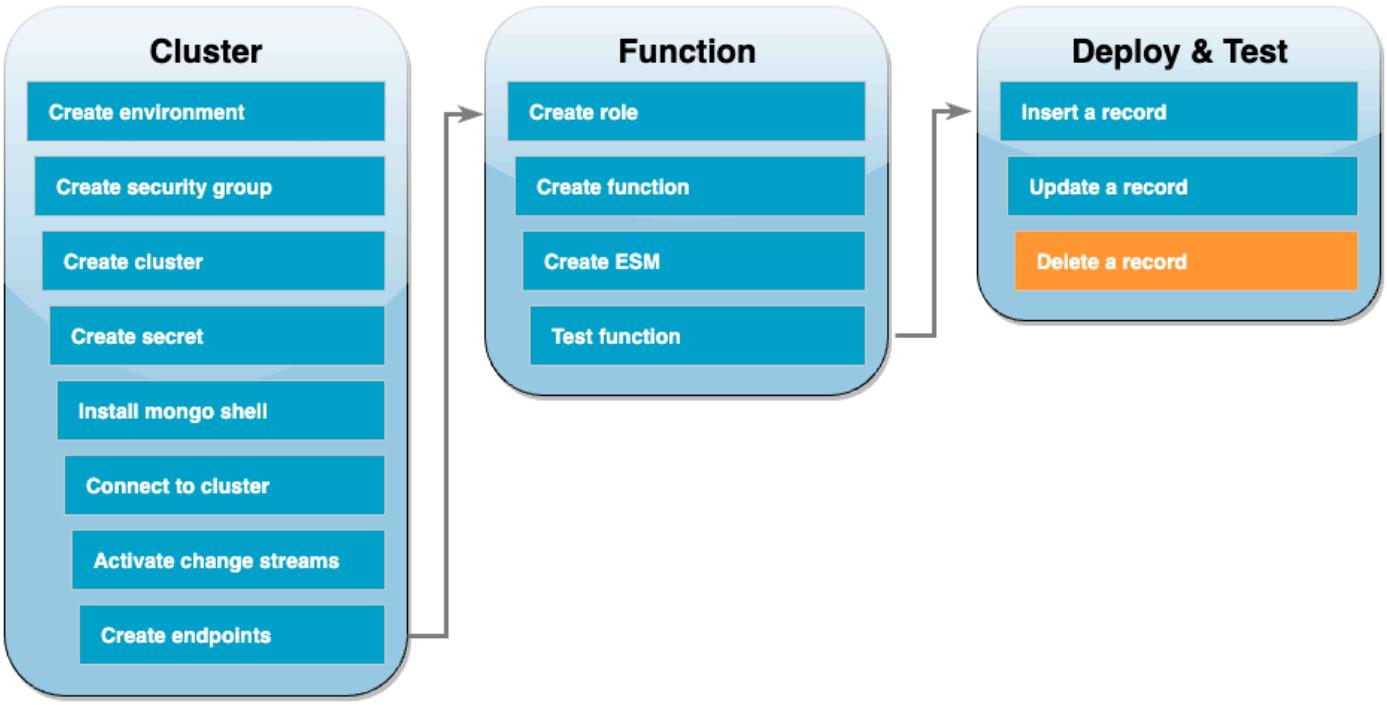


Em seguida, atualize o registro que você acabou de inserir com o seguinte comando:

```
db.products.update(
    { "name": "Pencil" },
    { $set: { "price": 0.50 }}
)
```

Verifique se sua função processou o evento com êxito examinando o CloudWatch Logs.

Teste sua função excluindo um registro



Por fim, exclua o registro que você acabou de inserir com o seguinte comando:

```
db.products.remove( { "name": "Pencil" } )
```

Verifique se sua função processou o evento com êxito examinando o CloudWatch Logs.

Limpe os recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Para excluir a função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Escolha Excluir.
4. Insira o nome do perfil no campo de entrada de texto e escolha Delete (Excluir).

Para excluir os endpoints da VPC

1. Abra o console da VPC. No menu à esquerda, em Nuvem privada virtual, escolha Endpoints.
2. Selecione os endpoints que você criou.
3. Escolha Actions (Ações), Delete VPC endpoints (Excluir endpoints da VPC).
4. Digite **`delete`** no campo de entrada de texto.
5. Escolha Excluir.

Para excluir o cluster do Amazon DocumentDB

1. Abra o console do DocumentDB.
2. Escolha o cluster do DocumentDB que você criou para este tutorial e desative a proteção contra exclusão.
3. Na página principal de Clusters, escolha seu cluster do DocumentDB novamente.
4. Escolha Ações, Excluir.
5. Em Criar snapshot final do cluster, selecione Não.
6. Digite **`delete`** no campo de entrada de texto.
7. Escolha Excluir.

Para excluir o segredo no Secrets Manager

1. Abra o console do Secrets Manager.
2. Escolha o segredo que você criou para este tutorial.
3. Escolha Ações, Excluir segredo.
4. Escolha Schedule deletion.

Para excluir o grupo de segurança do Amazon EC2

1. Abra o console do EC2. Em Rede e segurança, escolha Grupos de segurança.
2. Selecione o grupo de segurança que você criou para este tutorial.
3. Escolha Ações, Excluir grupos de segurança.
4. Escolha Excluir.

Para excluir o ambiente do Cloud9

1. Abra o console do Cloud9.
2. Selecione o ambiente que você criou para este tutorial.
3. Escolha Excluir.
4. Digite **`delete`** no campo de entrada de texto.
5. Escolha Excluir.

Filtragem de eventos do Lambda

É possível usar filtragem de eventos para controlar quais registros de um stream ou fila que o Lambda enviará para a função. Por exemplo, é possível adicionar um filtro para que sua função processe somente mensagens do Amazon SQS contendo determinados parâmetros de dados. A filtragem de eventos funciona com mapeamentos da origem do evento. É possível adicionar filtros aos mapeamentos da origem do evento para os seguintes serviços da AWS:

- Amazon DynamoDB
- Amazon Kinesis Data Streams
- Amazon MQ
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Apache Kafka autogerenciado
- Amazon Simple Queue Service (Amazon SQS)

O Lambda não oferece suporte à filtragem de eventos para o Amazon DocumentDB.

Por padrão, é possível definir até cinco filtros diferentes para um único mapeamento da origem do evento. Seus filtros estão logicamente relacionados por OR. Se um registro da sua origem de

eventos satisfizer um ou mais dos seus filtros, o Lambda incluirá o registro no próximo evento enviado à sua função. Se nenhum dos seus filtros for satisfeito, o Lambda descartará o registro.

> **Note**
>
> Se você precisar definir mais de cinco filtros para uma origem de evento, poderá solicitar um aumento de cota de até 10 filtros para cada origem de evento. Se tentar adicionar mais filtros do que a cota atual permite, o Lambda retornará um erro quando você tentar criar a origem de eventos.

Tópicos

Fundamentos de filtragem de eventos

Um objeto de critérios de filtro (`FilterCriteria`) é uma estrutura que consiste em uma lista de filtros (`Filters`). Cada filtro é uma estrutura que define um padrão de filtragem de eventos (`Pattern`). Um padrão é uma representação em string de uma regra de filtro JSON. A estrutura de um objeto `FilterCriteria` é descrita a seguir.

```
{
   "Filters": [
```

```
        {
            "Pattern": "{ \"Metadata1\": [ rule1 ], \"data\": { \"Data1\":
 [ rule2 ] }}"
        }
    ]
}
```

Para maior clareza, aqui está o valor de `Pattern` do filtro expandido em JSON simples.

```
{
    "Metadata1": [ rule1 ],
    "data": {
        "Data1": [ rule2 ]
    }
}
```

Seu padrão de filtro pode incluir propriedades de metadados, propriedades de dados ou ambas. Os parâmetros de metadados disponíveis e o formato dos parâmetros de dados variam de acordo com o AWS service (Serviço da AWS) que está atuando como origem do evento. Por exemplo, suponha que seu mapeamento da origem do evento receba o registro a seguir de uma fila do 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"
}
```

- As propriedades dos metadados são os campos que contêm informações sobre o evento que criou o registro. No exemplo de registro do Amazon SQS, as propriedades dos metadados incluem campos como `messageID`, `eventSourceArn`, e `awsRegion`.

- As propriedades dos dados são os campos do registro que contêm os dados do seu stream ou fila. No exemplo de evento do Amazon SQS, a chave para o campo de dados é `body`, e as propriedades dos dados são os campos `City`, `State` e `Temperature`.

Diferentes tipos de origens de eventos usam diferentes valores-chave para seus campos de dados. Para filtrar as propriedades de dados, certifique-se de usar a chave correta no padrão do seu filtro. Para obter uma lista de chaves de filtragem de dados e ver exemplos de padrões de filtro para cada um dos AWS service (Serviço da AWS) com suporte, consulteUso de filtros com diferentes Serviços da AWS.

A filtragem de eventos pode lidar com a filtragem JSON de vários níveis. Por exemplo, considere o fragmento a seguir de um registro de um stream do DynamoDB:

```
"dynamodb": {
    "Keys": {
        "ID": {
            "S": "ABCD"
        }
        "Number": {
            "N": "1234"
    },
    ...
}
```

Suponha que você queira processar somente os registros em que o valor da chave de classificação `Number` seja 4567. Nesse caso, seu objeto `FilterCriteria` deve ser semelhante a:

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\": { \"Keys\": { \"Number\": { \"N\":
 [ "4567" ] } } } }"
        }
    ]
}
```

Para maior clareza, aqui está o valor de `Pattern` do filtro expandido em JSON simples.

```
{
    "dynamodb": {
```

```
        "Keys": {
            "Number": {
                "N": [ "4567" ]
                }
            }
        }
}
```

Tratamento de registros que não atendem aos critérios de filtragem

A forma como os registros que não atendem ao seu filtro são tratados depende da origem do evento.

- Para o Amazon SQS, se a mensagem não atender aos critérios de filtro, o Lambda removerá automaticamente a mensagem da fila. Não é necessário excluir manualmente essas mensagens no Amazon SQS.
- Para o Kinesis e no DynamoDB, quando seus critérios de filtro processam um registro, o iterador de streams avança além desse registro. Se o registro não atender aos critérios de filtro, não será necessário excluir manualmente o registro da fonte de eventos. Após o período de retenção, o Kinesis e o DynamoDB excluem esses registros antigos automaticamente. Se quiser que os registros sejam excluídos antes, consulte Alterar o período de retenção de dados.
- Para mensagens do Amazon MSK, do Apache Kafka autogerenciadas e do Amazon MQ, o Lambda descarta as mensagens que não correspondem a todos os campos incluídos no filtro. Para o Apache Kafka autogerenciado, o Lambda confirma desvios para mensagens correspondentes ou não correspondentes depois de invocar a função com êxito. Para o Amazon MQ, o Lambda reconhece as mensagens coincidentes após invocar a função com êxito e reconhece as mensagens não coincidentes ao filtrá-las.

Sintaxe das regras de filtros

Para as regras de filtros, o Lambda é compatível com as regras do Amazon EventBridge e usa a mesma sintaxe que o EventBridge. Para obter mais informações, consulte Padrões de eventos do Amazon EventBridge no Guia do usuário do Amazon EventBridge.

A seguir, há um resumo de todas as operações de comparação disponíveis para a filtragem de eventos do Lambda.

Operador de comparação	Exemplo	Sintaxe da regra
Nulo	UserID é null	"UserID": [null]
Vazio	LastName está vazio	"LastName": [""]
Igual	Name é “Alice”	"Name": ["Alice"]
É igual a (ignorar maiúsculas e minúsculas)	Name é “Alice”	"Name": [{ "equals-ignore-case": "alice" }]
E	Location é “Nova York” e o Day é “Monday” (Segunda-feira)	"Location": ["New York"], "Day": ["Monday"]
Ou	PaymentType é “Credit” (Crédito) ou “Debit” (Débito)	"PaymentType": ["Credit", "Debit"]
Ou (vários campos)	Location é “New York” ou Day é “Monday”.	"$or": [{ "Location": ["New York"] }, { "Day": ["Monday"] }]
Não	Weather é qualquer valor, exceto “Raining” (Chovendo)	"Weather": [{ "anything-but": ["Raining"] }]
Numeric (equals) (Numérico, igual)	Price é 100	"Price": [{ "numeric": ["=", 100] }]
Numeric (range) (Numérico, intervalo)	Price é superior a 10 e menor que ou igual a 20	"Price": [{ "numeric": [">", 10, "<=", 20] }]
Existe	ProductName existe	"ProductName": [{ "exists": true }]
Não existe	ProductName não existe	"ProductName": [{ "exists": false }]
Começa com	Region é nos EUA	"Region": [{"prefix": "us-" }]

Operador de comparação	Exemplo	Sintaxe da regra
Termina com	FileName termina com uma extensão .png.	"FileName": [{ "suffix": ".png" }]

> **Note**
>
> Como o EventBridge, para strings, o Lambda usa correspondência exata caractere por caractere sem case-folding nem qualquer normalização de strings. Para números, o Lambda também usa representação de string. Por exemplo, 300, 300.0 e 3.0e2 não são considerados iguais.

Observe que o operador Exists só funciona em nós terminais em seu JSON de origem de eventos. Não corresponde aos nós intermediários. Por exemplo, com o seguinte JSON, o padrão de filtro `{ "person": { "address": [ { "exists": true } ] } }` não encontraria uma correspondência porque `"address"` é um nó intermediário.

```
{
  "person": {
    "name": "John Doe",
    "age": 30,
    "address": {
      "street": "123 Main St",
      "city": "Anytown",
      "country": "USA"
    }
  }
}
```

Anexar critérios de filtro a um mapeamento de fonte de eventos (console)

Siga estas etapas para criar um novo mapeamento de fonte de eventos com critérios de filtro usando o console do Lambda.

Para criar um novo mapeamento de fonte de eventos com critérios de filtro (console)

1. Abra a página Funções do console do Lambda.
2. Escolha o nome de uma função para a qual criar um mapeamento de fonte de evento.

3. Em Visão geral da função, escolha Adicionar gatilho.
4. Em Trigger configuration (Configuração do acionador), escolha um tipo de acionador com suporte a filtragem de eventos. Para obter uma lista dos serviços com suporte, consulte a lista no início desta página.
5. Expanda Additional settings (Configurações adicionais).
6. Em Filter criteria, (Critérios de filtros), escolha Add, (Adicionar) e insira os filtros. Por exemplo, é possível inserir o seguinte.

   ```
   { "Metadata" : [ 1, 2 ] }
   ```

   Isso instrui o Lambda a processar somente os registros em que o campo `Metadata` é igual a 1 ou 2. É possível continuar selecionando Adicionar para adicionar mais filtros até o número máximo permitido.
7. Ao concluir a inclusão de filtros, escolha Salvar.

Ao inserir critérios de filtros usando o console, insira somente o padrão do filtro, pois não é necessário fornecer a chave `Pattern` ou as aspas escape. Na etapa 6 das instruções anteriores, `{ "Metadata" : [ 1, 2 ] }` corresponde aos `FilterCriteria` a seguir.

```
{
   "Filters": [
      {
          "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
      }
   ]
}
```

Após criar o mapeamento da fonte de eventos no console, você pode ver os `FilterCriteria` formatados nos detalhes do acionador. Para obter mais exemplos de criação de filtros de eventos usando o console, consulte Uso de filtros com diferentes Serviços da AWS.

Anexar critérios de filtros a um mapeamento de fonte de eventos (AWS CLI)

Suponha que você queira que um mapeamento de fonte de eventos tenha os seguintes `FilterCriteria`:

```
{
```

```
   "Filters": [
      {
          "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
      }
   ]
}
```

Para criar um novo mapeamento da origem do evento com esses critérios de filtro usando a AWS Command Line Interface (AWS CLI), execute o comando a seguir.

```
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 ]}"}]}'
```

Esse comando create-event-source-mapping cria um novo mapeamento da origem do evento do Amazon SQS para a função `my-function` com os `FilterCriteria` especificados.

Para adicionar esses critérios de filtro a um mapeamento da origem do evento existente, execute o comando a seguir.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"Metadata\" : [ 1, 2 ]}"}]}'
```

Para atualizar um mapeamento de fonte de eventos, você precisa do UUID. É possível obter o UUID com uma chamada a list-event-source-mappings. O Lambda também retorna o UUID na resposta da CLI de create-event-source-mapping.

Para remover critérios de filtro de uma origem de eventos, execute o comando UpdateEventSourceMapping a seguir com um objeto `FilterCriteria` vazio.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria "{}"
```

Para obter mais exemplos de criação de filtros de eventos usando a AWS CLI, consulte Uso de filtros com diferentes Serviços da AWS.

Anexar critérios de filtros a um mapeamento de fonte de eventos (AWS SAM)

Suponha que você deseje configurar uma origem de eventos no AWS SAM para usar os seguintes critérios de filtro:

```
{
   "Filters": [
      {
          "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
      }
   ]
}
```

Para adicionar esses critérios de filtro ao mapeamento da origem do evento, insira o trecho a seguir no modelo YAML da origem do evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{"Metadata": [1, 2]}'
```

Para obter mais informações sobre como criar e configurar um modelo do AWS SAM para um mapeamento da origem do evento, consulte a seção EventSource do Guia do desenvolvedor do AWS SAM. Para obter mais exemplos de criação de filtros de eventos usando modelos do AWS SAM, consulte Uso de filtros com diferentes Serviços da AWS.

Uso de filtros com diferentes Serviços da AWS

Diferentes tipos de origens de eventos usam diferentes valores-chave para seus campos de dados. Para filtrar as propriedades de dados, certifique-se de usar a chave correta no padrão do seu filtro. A tabela a seguir fornece as chaves de filtragem para cada um dos AWS service (Serviço da AWS) com suporte.

AWS service (Serviço da AWS)	Chave de filtragem
DynamoDB	`dynamodb`
Kinesis	`data`
Amazon MQ	`data`

AWS service (Serviço da AWS)	Chave de filtragem
Amazon MSK	`value`
Apache Kafka autogerenciado	`value`
Amazon SQS	`body`

As seções a seguir fornecem exemplos de padrões de filtro para diferentes tipos de origens de eventos. Eles também fornecem definições de formatos de dados recebidos e formatos de corpo de padrões de filtro para cada serviço com suporte.

Filtragem com DynamoDB

Suponha que você tenha uma tabela do DynamoDB com chave primária `CustomerName` e atributos `AccountManager` e `PaymentTerms`. Veja a seguir um exemplo de registro do stream da sua tabela do 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"
```

```
          }
      }
  }
```

Para filtrar com base nos valores de chave e atributo em sua tabela do DynamoDB, use a chave `dynamodb` no registro. As seções a seguir fornecem exemplos de diferentes tipos de filtro.

Como filtrar com chaves de tabela

Suponha que você queira que sua função processe somente os registros em que a chave primária `CustomerName` seja “AnyCompany Industries”. O objeto `FilterCriteria` seria como a seguir.

```
{
     "Filters": [
          {
              "Pattern": "{ \"dynamodb\" : { \"Keys\" : { \"CustomerName\" : { \"S\" :
 [ \"AnyCompany Industries\" ] } } } }"
          }
      ]
 }
```

Para maior clareza, aqui está o valor de `Pattern` do filtro expandido em JSON simples.

```
{
     "dynamodb": {
          "Keys": {
              "CustomerName": {
                  "S": [ "AnyCompany Industries" ]
                  }
              }
          }
 }
```

É possível adicionar seu filtro usando o console, a AWS CLI ou um modelo do AWS SAM.

Console

Para adicionar esse filtro usando o console, siga as instruções em Anexar critérios de filtro a um mapeamento de fonte de eventos (console) e insira a string a seguir em Critérios do filtro.

```
{ "dynamodb" : { "Keys" : { "CustomerName" : { "S" : [ "AnyCompany
 Industries" ] } } } }
```

AWS CLI

Para criar um novo mapeamento da origem do evento com esses critérios de filtro usando a AWS Command Line Interface (AWS CLI), execute o comando a seguir.

```
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\" ] } } } }"}]}'
```

Para adicionar esses critérios de filtro a um mapeamento da origem do evento existente, execute o comando a seguir.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"Keys\" :
 { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"}]}'
```

AWS SAM

Para adicionar esse filtro usando o AWS SAM, adicione o trecho a seguir ao modelo YAML da origem do evento.

```
FilterCriteria:
   Filters:
     - Pattern: '{ "dynamodb" : { "Keys" : { "CustomerName" : { "S" : [ "AnyCompany
 Industries" ] } } } }'
```

Como filtrar com atributos de tabela

Com o DynamoDB, você também pode usar as teclas `NewImage` e `OldImage` para filtrar os valores dos atributos. Suponha que você queira filtrar registros em que o atributo `AccountManager` na imagem mais recente da tabela seja “Pat Candella” ou “Shirley Rodriguez”. O objeto `FilterCriteria` seria como a seguir.

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S
\" : [ \"Pat Candella\", \"Shirley Rodriguez\" ] } } } }"
```

```
        }
    ]
}
```

Para maior clareza, aqui está o valor de `Pattern` do filtro expandido em JSON simples.

```
{
    "dynamodb": {
        "NewImage": {
            "AccountManager": {
                "S": [ "Pat Candella", "Shirley Rodriguez" ]
            }
        }
    }
}
```

É possível adicionar seu filtro usando o console, a AWS CLI ou um modelo do AWS SAM.

Console

Para adicionar esse filtro usando o console, siga as instruções em Anexar critérios de filtro a um mapeamento de fonte de eventos (console) e insira a string a seguir em Critérios do filtro.

```
{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat Candella",
 "Shirley Rodriguez" ] } } } }
```

AWS CLI

Para criar um novo mapeamento da origem do evento com esses critérios de filtro usando a AWS Command Line Interface (AWS CLI), execute o comando a seguir.

```
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
\" ] } } } }"}]}'
```

Para adicionar esses critérios de filtro a um mapeamento da origem do evento existente, execute o comando a seguir.

```
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

Para adicionar esse filtro usando o AWS SAM, adicione o trecho a seguir ao modelo YAML da origem do evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella", "Shirley Rodriguez" ] } } } }'
```

Como filtrar com expressões boolianas

Você também pode criar filtros usando expressões boolianas AND. Essas expressões podem incluir os parâmetros de chave e de atributo da tabela. Suponha que você queira filtrar registros em que o valor de `NewImage` de `AccountManager` seja “Pat Candella” e o valor de `OldImage` seja “Terry Whitlock”. O objeto `FilterCriteria` seria como a seguir.

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S
\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" : { \"OldImage\" : { \"AccountManager
\" : { \"S\" : [ \"Terry Whitlock\" ] } } } }"
        }
    ]
}
```

Para maior clareza, aqui está o valor de `Pattern` do filtro expandido em JSON simples.

```
{
    "dynamodb" : {
        "NewImage" : {
            "AccountManager" : {
                "S" : [
                    "Pat Candella"
                ]
```

```
            }
        }
    },
    "dynamodb": {
        "OldImage": {
            "AccountManager": {
                "S": [
                    "Terry Whitlock"
                ]
            }
        }
    }
}
```

É possível adicionar seu filtro usando o console, a AWS CLI ou um modelo do AWS SAM.

Console

Para adicionar esse filtro usando o console, siga as instruções em Anexar critérios de filtro a um mapeamento de fonte de eventos (console) e insira a string a seguir em Critérios do filtro.

```
{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" :
 [ "Terry Whitlock" ] } } } }
```

AWS CLI

Para criar um novo mapeamento da origem do evento com esses critérios de filtro usando a AWS Command Line Interface (AWS CLI), execute o comando a seguir.

```
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\" ] } } } }
 "}]}'
```

Para adicionar esses critérios de filtro a um mapeamento da origem do evento existente, execute o comando a seguir.

```
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

Para adicionar esse filtro usando o AWS SAM, adicione o trecho a seguir ao modelo YAML da origem do evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" :
 [ "Terry Whitlock" ] } } } }'
```

> **Note**
>
> A filtragem de eventos do DynamoDB não oferece suporte ao uso de operadores numéricos (igualdade numérica e intervalo numérico). Mesmo que os itens em sua tabela sejam armazenados como números, esses parâmetros são convertidos em strings no objeto do registro JSON.

Como usar o operador Exists com o DynamoDB

Devido à forma de estruturação dos objetos de evento JSON do DynamoDB, o uso do operador Exists requer cuidados especiais. O operador Exists só funciona em nós terminais no evento JSON. Portanto, se seu padrão de filtro usar Exists para testar um nó intermediário, ele não funcionará. Considere o seguinte item de tabela do DynamoDB:

```
{
  "UserID": {"S": "12345"},
  "Name": {"S": "John Doe"},
  "Organizations": {"L": [
      {"S":"Sales"},
      {"S":"Marketing"},
      {"S":"Support"}
    ]
```

```
  }
}
```

Talvez você queira criar um padrão de filtro como o seguinte, que teste os eventos que contenham `"Organizations"`:

```
{ "dynamodb" : { "NewImage" : { "Organizations" : [ { "exists": true } ] } } }
```

No entanto, esse padrão de filtro nunca retornaria uma correspondência porque `"Organizations"` não é um nó terminal. O exemplo a seguir mostra como usar corretamente o operador Exists para estruturar o padrão de filtro desejado:

```
{ "dynamodb" : { "NewImage" : {"Organizations": {"L": {"S": [ {"exists":
 true } ] } } } } }
```

Formato JSON para filtragem do DynamoDB

Para filtrar corretamente eventos de origens do DynamoDB, tanto o campo de dados como os critérios de filtro para o campo de dados (`dynamodb`) devem estar em formato JSON válido. Se algum desses campos não estiver em um formato JSON válido, o Lambda descartará a mensagem ou emitirá uma exceção. A tabela a seguir resume o comportamento específico:

Formato dos dados recebidos	Formato de filtro padrão para propriedades de dados	Ação resultante
JSON válido	JSON válido	Filtros do Lambda com base em seus critérios de filtro.
JSON válido	Nenhum padrão de filtro para propriedades de dados	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.
JSON válido	Não JSON	O Lambda emite uma exceção no momento da criação ou atualização do mapeamento da fonte de eventos. O padrão de filtro para propriedades

Formato dos dados recebidos	Formato de filtro padrão para propriedades de dados	Ação resultante
		de dados deve estar em um formato JSON válido.
Não JSON	JSON válido	O Lambda descarta o registro.
Não JSON	Nenhum padrão de filtro para propriedades de dados	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.
Não JSON	Não JSON	O Lambda emite uma exceção no momento da criação ou atualização do mapeamento da fonte de eventos. O padrão de filtro para propriedades de dados deve estar em um formato JSON válido.

Filtragem com o Kinesis

Suponha que um produtor esteja inserindo dados formatados em JSON em seu fluxo de dados do Kinesis. Um exemplo de registro seria semelhante ao a seguir, com os dados JSON convertidos em uma string codificada em Base64 no 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"
}
```

Desde que os dados que o produtor coloque no stream sejam JSON válido, é possível usar a filtragem de eventos para filtrar registros usando a chave `data`. Suponha que um produtor esteja inserindo dados em seu stream do Kinesis no formato JASON a seguir.

```
{
    "record": 12345,
    "order": {
        "type": "buy",
        "stock": "ANYCO",
        "quantity": 1000
        }
}
```

Para filtrar somente os registros em que o tipo de pedido é “comprar”, o objeto `FilterCriteria` seria como a seguir.

```
{
    "Filters": [
        {
            "Pattern": "{ \"data\" : { \"order\" : { \"type\" : [ \"buy\" ] } } }"
        }
    ]
}
```

Para maior clareza, aqui está o valor de `Pattern` do filtro expandido em JSON simples.

```
{
    "data": {
        "order": {
            "type": [ "buy" ]
            }
      }
}
```

É possível adicionar seu filtro usando o console, a AWS CLI ou um modelo do AWS SAM.

Console

Para adicionar esse filtro usando o console, siga as instruções em Anexar critérios de filtro a um mapeamento de fonte de eventos (console) e insira a string a seguir em Critérios do filtro.

```
{ "data" : { "order" : { "type" : [ "buy" ] } } }
```

AWS CLI

Para criar um novo mapeamento da origem do evento com esses critérios de filtro usando a AWS Command Line Interface (AWS CLI), execute o comando a seguir.

```
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\" ] } } }"}]}'
```

Para adicionar esses critérios de filtro a um mapeamento da origem do evento existente, execute o comando a seguir.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"order\" : { \"type
\" : [ \"buy\" ] } } }"}]}'
```

AWS SAM

Para adicionar esse filtro usando o AWS SAM, adicione o trecho a seguir ao modelo YAML da origem do evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : { "order" : { "type" : [ "buy" ] } } }'
```

Para filtrar corretamente eventos de origens do Kinesis, tanto o campo de dados como os critérios de filtro para o campo de dados devem estar em formato JSON válido. Se algum desses campos não estiver em um formato JSON válido, o Lambda descartará a mensagem ou emitirá uma exceção. A tabela a seguir resume o comportamento específico:

Formato dos dados recebidos	Formato de filtro padrão para propriedades de dados	Ação resultante
JSON válido	JSON válido	Filtros do Lambda com base em seus critérios de filtro.
JSON válido	Nenhum padrão de filtro para propriedades de dados	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.
JSON válido	Não JSON	O Lambda emite uma exceção no momento da criação ou atualização do mapeamento da fonte de eventos. O padrão de filtro para propriedades de dados deve estar em um formato JSON válido.
Não JSON	JSON válido	O Lambda descarta o registro.
Não JSON	Nenhum padrão de filtro para propriedades de dados	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.
Não JSON	Não JSON	O Lambda emite uma exceção no momento da criação ou atualização do mapeamento da fonte de eventos. O padrão de filtro para propriedades de dados deve estar em um formato JSON válido.

Filtragem de registros agregados do Kinesis

Com o Kinesis, é possível agregar vários registros em um único registro do Kinesis Data Streams para aumentar seu throughput. O Lambda pode aplicar critérios de filtro a registros agregados somente quando você usar a distribuição avançada do Kinesis. Não há suporte para a filtragem de registros agregados com o Kinesis padrão. Ao usar a distribuição avançada, você configura um consumidor de throughput dedicado do Kinesis para atuar como acionador para sua função do Lambda. Em seguida, o Lambda filtra os registros agregados e passa somente os registros que atendam aos seus critérios de filtragem.

Para saber mais sobre a agregação de registros do Kinesis, consulte a seção Agregação na página Conceitos principais da Kinesis Producer Library (KPL). Para saber mais sobre como usar o Lambda com a distribuição avançada do Kinesis, consulte Aumento do desempenho do processamento de streams em tempo real com a distribuição avançada do Amazon Kinesis Data Streams e o AWS Lambda no blog de computação da AWS.

Filtragem com o Amazon MQ

Suponha que sua fila de mensagens do Amazon MQ contenha mensagens em formato JSON válido ou como strings simples. Um exemplo de registro seria semelhante ao a seguir, com os dados convertidos em uma string codificada em Base64 no 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=="
}
```

Para os agentes Active MQ e Rabbit MQ, é possível usar a filtragem de eventos para filtrar registros usando a chave `data`. Suponha que a fila do Amazon MQ contenha mensagens no formato JSON a seguir.

```
{
    "timeout": 0,
    "IPAddress": "203.0.113.254"
}
```

Para filtrar somente os registros onde o campo `timeout` é maior que 0, o objeto `FilterCriteria` seria como a seguir.

```
{
    "Filters": [
        {
            "Pattern": "{ \"data\" : { \"timeout\" : [ { \"numeric\": [ \">\",
 0] } } ] } }"
        }
    ]
}
```

Para maior clareza, aqui está o valor de `Pattern` do filtro expandido em JSON simples.

```
{
    "data": {
        "timeout": [ { "numeric": [ ">", 0 ] } ]
        }
```

```
}
```

É possível adicionar seu filtro usando o console, a AWS CLI ou um modelo do AWS SAM.

Console

Para adicionar esse filtro usando o console, siga as instruções em Anexar critérios de filtro a um mapeamento de fonte de eventos (console) e insira a string a seguir em Critérios do filtro.

```
{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }
```

AWS CLI

Para criar um novo mapeamento da origem do evento com esses critérios de filtro usando a AWS Command Line Interface (AWS CLI), execute o comando a seguir.

```
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 ] } ] } }"}]}'
```

Para adicionar esses critérios de filtro a um mapeamento da origem do evento existente, execute o comando a seguir.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :
 [ { \"numeric\": [ \">\", 0 ] } ] } }"}]}'
```

AWS SAM

Para adicionar esse filtro usando o AWS SAM, adicione o trecho a seguir ao modelo YAML da origem do evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }'
```

Com o Amazon MQ, você também pode filtrar registros onde a mensagem é uma string simples. Suponha que você queira processar somente registros onde a mensagem comece com “Resultado: ”. O objeto `FilterCriteria` seria como a seguir.

```
{
    "Filters": [
        {
            "Pattern": "{ \"data\" : [ { \"prefix\": \"Result: \" } ] }"
        }
    ]
}
```

Para maior clareza, aqui está o valor de `Pattern` do filtro expandido em JSON simples.

```
{
    "data": [
        {
        "prefix": "Result: "
        }
    ]
}
```

É possível adicionar seu filtro usando o console, a AWS CLI ou um modelo do AWS SAM.

Console

Para adicionar esse filtro usando o console, siga as instruções em Anexar critérios de filtro a um mapeamento de fonte de eventos (console) e insira a string a seguir em Critérios do filtro.

```
{ "data" : [ { "prefix": "Result: " } ] }
```

AWS CLI

Para criar um novo mapeamento da origem do evento com esses critérios de filtro usando a AWS Command Line Interface (AWS CLI), execute o comando a seguir.

```
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: \" } ] }"}]}'
```

Para adicionar esses critérios de filtro a um mapeamento da origem do evento existente, execute o comando a seguir.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : [ { \"prefix\":
 \"Result: \" } ] }"}]}'
```

AWS SAM

Para adicionar esse filtro usando o AWS SAM, adicione o trecho a seguir ao modelo YAML da origem do evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : [ { "prefix": "Result " } ] }'
```

As mensagens do Amazon MQ devem ser strings codificadas em UTF-8, sejam em strings simples ou no formato JSON. Isso porque o Lambda decodifica as matrizes de bytes do Amazon MQ em UTF-8 antes de aplicar os critérios de filtragem. Se as mensagens usarem outra codificação, como UTF-16 ou ASCII, ou se o formato da mensagem não corresponder ao formato dos `FilterCriteria`, o Lambda processará somente os filtros de metadados. A tabela a seguir resume o comportamento específico:

Formato do da mensagem recebida	Formato padrão de filtro para propriedades de mensagem	Ação resultante
String simples	String simples	Filtros do Lambda com base em seus critérios de filtro.
String simples	Nenhum padrão de filtro para propriedades de dados	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.

Formato do da mensagem recebida	Formato padrão de filtro para propriedades de mensagem	Ação resultante
String simples	JSON válido	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.
JSON válido	String simples	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.
JSON válido	Nenhum padrão de filtro para propriedades de dados	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.
JSON válido	JSON válido	Filtros do Lambda com base em seus critérios de filtro.
String não codificada em UTF-8	JSON, string de texto simples ou nenhum padrão	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.

Filtragem com o Amazon MSK e o Apache Kafka autogerenciado

Suponha que um produtor esteja escrevendo mensagens para um tópico em seu cluster do Amazon MSK ou do Apache Kafka autogerenciado, em formato JSON válido ou como strings simples. Um exemplo de registro seria semelhante ao a seguir, com a mensagem convertida em uma string codificada em Base64 no campo `value`.

```
{
    "mytopic-0":[
        {
            "topic":"mytopic",
            "partition":0,
```

```
            "offset":15,
            "timestamp":1545084650987,
            "timestampType":"CREATE_TIME",
            "value":"SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
            "headers":[]
        }
    ]
}
```

Suponha que o produtor do Apache Kafka esteja escrevendo mensagens para o tópico no formato JSON a seguir.

```
{
    "device_ID": "AB1234",
    "session":{
        "start_time": "yyyy-mm-ddThh:mm:ss",
        "duration": 162
    }
}
```

É possível usar a chave `value` para filtrar registros. Suponha que você queira filtrar somente os registros onde `device_ID` comece com as letras AB. O objeto `FilterCriteria` seria como a seguir.

```
{
    "Filters": [
        {
            "Pattern": "{ \"value\" : { \"device_ID\" : [ { \"prefix\": \"AB\" } ] } }"
        }
    ]
}
```

Para maior clareza, aqui está o valor de `Pattern` do filtro expandido em JSON simples.

```
{
    "value": {
        "device_ID": [ { "prefix": "AB" } ]
      }
}
```

É possível adicionar seu filtro usando o console, a AWS CLI ou um modelo do AWS SAM.

Console

Para adicionar esse filtro usando o console, siga as instruções em Anexar critérios de filtro a um mapeamento de fonte de eventos (console) e insira a string a seguir em Critérios do filtro.

```
{ "value" : { "device_ID" : [ { "prefix":  "AB" } ] } }
```

AWS CLI

Para criar um novo mapeamento da origem do evento com esses critérios de filtro usando a AWS Command Line Interface (AWS CLI), execute o comando a seguir.

```
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\" } ] } }"}]}'
```

Para adicionar esses critérios de filtro a um mapeamento da origem do evento existente, execute o comando a seguir.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :
 [ { \"prefix\":  \"AB\" } ] } }"}]}'
```

AWS SAM

Para adicionar esse filtro usando o AWS SAM, adicione o trecho a seguir ao modelo YAML da origem do evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : { "device_ID" : [ { "prefix":  "AB" } ] } }'
```

Com o Amazon MSK e o Apache Kafka autogerenciado, também é possível filtrar registros onde a mensagem seja uma sequência de caracteres simples. Suponha que você queira ignorar as mensagens em que a string seja “erro”. O objeto `FilterCriteria` seria como a seguir.

```
{
    "Filters": [
        {
            "Pattern": "{ \"value\" : [ { \"anything-but\": [ \"error\" ] } ] }"
        }
    ]
}
```

Para maior clareza, aqui está o valor de `Pattern` do filtro expandido em JSON simples.

```
{
    "value": [
        {
        "anything-but": [ "error" ]
        }
    ]
}
```

É possível adicionar seu filtro usando o console, a AWS CLI ou um modelo do AWS SAM.

Console

Para adicionar esse filtro usando o console, siga as instruções em Anexar critérios de filtro a um mapeamento de fonte de eventos (console) e insira a string a seguir em Critérios do filtro.

```
{ "value" : [ { "anything-but": [ "error" ] } ] }
```

AWS CLI

Para criar um novo mapeamento da origem do evento com esses critérios de filtro usando a AWS Command Line Interface (AWS CLI), execute o comando a seguir.

```
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\" ] } ] }"}]}'
```

Para adicionar esses critérios de filtro a um mapeamento da origem do evento existente, execute o comando a seguir.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":
 [ \"error\" ] } ] }"}]}'
```

AWS SAM

Para adicionar esse filtro usando o AWS SAM, adicione o trecho a seguir ao modelo YAML da origem do evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : [ { "anything-but": [ "error" ] } ] }'
```

As mensagens do Amazon MSK e do Apache Kafka autogerenciado devem ser strings codificadas em UTF-8, sejam em texto simples ou no formato JSON. Isso porque o Lambda decodifica as matrizes de bytes do Amazon MSK em UTF-8 antes de aplicar os critérios de filtragem. Se as mensagens usarem outra codificação, como UTF-16 ou ASCII, ou se o formato da mensagem não corresponder ao formato dos `FilterCriteria`, o Lambda processará somente os filtros de metadados. A tabela a seguir resume o comportamento específico:

Formato do da mensagem recebida	Formato padrão de filtro para propriedades de mensagem	Ação resultante
String simples	String simples	Filtros do Lambda com base em seus critérios de filtro.
String simples	Nenhum padrão de filtro para propriedades de dados	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.
String simples	JSON válido	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.

Formato do da mensagem recebida	Formato padrão de filtro para propriedades de mensagem	Ação resultante
JSON válido	String simples	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.
JSON válido	Nenhum padrão de filtro para propriedades de dados	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.
JSON válido	JSON válido	Filtros do Lambda com base em seus critérios de filtro.
String não codificada em UTF-8	JSON, string de texto simples ou nenhum padrão	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.

Filtragem com o Amazon SQS

Suponha que sua fila do Amazon SQS contenha mensagens no formato JSON a seguir.

```
{
    "RecordNumber": 0000,
    "TimeStamp": "yyyy-mm-ddThh:mm:ss",
    "RequestCode": "AAAA"
}
```

Um exemplo de registro para essa fila seria como a seguir.

```
{
    "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"
}
```

Para filtrar com base no conteúdo de suas mensagens do Amazon SQS, use a chave `body` no registro de mensagens do Amazon SQS. Suponha que você queira processar somente os registros onde o `RequestCode` na sua mensagem do Amazon SQS seja “BBBB”. O objeto `FilterCriteria` seria como a seguir.

```
{
    "Filters": [
        {
            "Pattern": "{ \"body\" : { \"RequestCode\" : [ \"BBBB\" ] } }"
        }
    ]
}
```

Para maior clareza, aqui está o valor de `Pattern` do filtro expandido em JSON simples.

```
{
    "body": {
        "RequestCode": [ "BBBB" ]
        }
}
```

É possível adicionar seu filtro usando o console, a AWS CLI ou um modelo do AWS SAM.

Console

Para adicionar esse filtro usando o console, siga as instruções em Anexar critérios de filtro a um mapeamento de fonte de eventos (console) e insira a string a seguir em Critérios do filtro.

```
{ "body" : { "RequestCode" : [ "BBBB" ] } }
```

AWS CLI

Para criar um novo mapeamento da origem do evento com esses critérios de filtro usando a AWS Command Line Interface (AWS CLI), execute o comando a seguir.

```
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\" ] } }"}]}'
```

Para adicionar esses critérios de filtro a um mapeamento da origem do evento existente, execute o comando a seguir.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RequestCode\" :
 [ \"BBBB\" ] } }"}]}'
```

AWS SAM

Para adicionar esse filtro usando o AWS SAM, adicione o trecho a seguir ao modelo YAML da origem do evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "body" : { "RequestCode" : [ "BBBB" ] } }'
```

Suponha que você queira que sua função processe somente os registros onde `RecordNumber` seja maior que 9999. O objeto `FilterCriteria` seria como a seguir.

```
{
    "Filters": [
        {
            "Pattern": "{ \"body\" : { \"RecordNumber\" : [ { \"numeric\": [ \">\",
 9999 ] } ] } }"
        }
    ]
}
```

Para maior clareza, aqui está o valor de `Pattern` do filtro expandido em JSON simples.

```
{
    "body": {
        "RecordNumber": [
            {
                "numeric": [ ">", 9999 ]
            }
        ]
    }
}
```

É possível adicionar seu filtro usando o console, a AWS CLI ou um modelo do AWS SAM.

Console

Para adicionar esse filtro usando o console, siga as instruções em Anexar critérios de filtro a um mapeamento de fonte de eventos (console) e insira a string a seguir em Critérios do filtro.

```
{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }
```

AWS CLI

Para criar um novo mapeamento da origem do evento com esses critérios de filtro usando a AWS Command Line Interface (AWS CLI), execute o comando a seguir.

```
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 ] } ] } }"}]}'
```

Para adicionar esses critérios de filtro a um mapeamento da origem do evento existente, execute o comando a seguir.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RecordNumber\" :
 [ { \"numeric\": [ \">\", 9999 ] } ] } }"}]}'
```

AWS SAM

Para adicionar esse filtro usando o AWS SAM, adicione o trecho a seguir ao modelo YAML da origem do evento.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }'
```

No Amazon SQS, o corpo da mensagem pode ser qualquer string. Porém, isso pode ser problemático se os `FilterCriteria` esperarem que o `body` esteja em um formato JSON válido. O cenário oposto também é verdadeiro: se o corpo da mensagem recebida estiver em formato JSON, mas seus critérios de filtragem esperarem que o `body` seja uma string simples, isso poderá levar um comportamento não pretendido.

Para evitar esse problema, certifique-se de que o formato do corpo nos seus `FilterCriteria` corresponda ao formato esperado do `body` nas mensagens que você receber da fila. Antes de filtrar suas mensagens, o Lambda avalia automaticamente o formato do corpo da mensagem recebida e do seu padrão de filtro para o `body`. Se houver incompatibilidade, o Lambda descartará a mensagem. A tabela a seguir resume essa avaliação:

Formato do **body** da mensagem recebida	Formato do **body** do padrão de filtro	Ação resultante
String simples	String simples	Filtros do Lambda com base em seus critérios de filtro.
String simples	Nenhum padrão de filtro para propriedades de dados	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.
String simples	JSON válido	O Lambda descarta a mensagem.
JSON válido	String simples	O Lambda descarta a mensagem.

Formato do **body** da mensagem recebida	Formato do **body** do padrão de filtro	Ação resultante
JSON válido	Nenhum padrão de filtro para propriedades de dados	Filtros do Lambda (somente nas outras propriedades de metadados) com base nos seus critérios de filtro.
JSON válido	JSON válido	Filtros do Lambda com base em seus critérios de filtro.

Como testar funções do Lambda no console

Você pode testar sua função do Lambda no console invocando sua função com um evento de teste. Um evento de teste é uma entrada JSON para sua função. Se a função não necessitar de entrada, o evento poderá ser um documento vazio (`{}`).

Quando você executa um teste no console, o Lambda invoca sua função de maneira síncrona com o evento de teste. O runtime da função converte o JSON do evento em um objeto e o transmite ao método do manipulador do código para processamento.

> **Criar um evento de teste**
>
> Antes de testar no console, é necessário criar um evento de teste privado ou compartilhável.

Como invocar funções com eventos de teste

Para testar uma função

1. Abra a página Funções do console do Lambda.
2. Escolha o nome da função que deseja testar.
3. Selecione a guia Test (Testar).
4. Em Evento de teste, escolha Criar evento ou Editar evento salvo e escolha o evento salvo que deseja usar.
5. Opcionalmente, escolha um Modelo para o JSON do evento.
6. Escolha Testar.
7. Para analisar os resultados do teste, em Execution result (Resultado da execução), expanda Details (Detalhes).

Para invocar sua função sem salvar seu evento de teste, escolha Test (Testar) antes de salvar. Isso cria um evento de teste não salvo que o Lambda preservará somente durante a sessão.

Na guia Code (Código), também é possível acessar seus eventos de teste salvos e não salvos. Em seguida, escolha Test (Testar) e escolha seu evento de teste.

Criar eventos de teste privados

Os eventos privados de teste estão disponíveis apenas para o criador do evento e não exigem permissões adicionais para uso. É possível criar e salvar até 10 eventos privados de teste por função.

Para criar um evento privado de teste

1. Abra a página Funções do console do Lambda.
2. Escolha o nome da função que deseja testar.
3. Selecione a guia Test (Testar).
4. Em Test event (Evento de teste), faça o seguinte:
    a. Escolha um Template (Modelo).
    b. Insira um Name (Nome) para o teste.
    c. Na caixa de entrada de texto, insira o evento de teste JSON.
    d. Em Event sharing settings (Configurações de compartilhamento de eventos), escolha Private (Privado).
5. Escolha Salvar alterações.

Você também pode criar novos eventos de teste na guia Code (Código). Em seguida, escolha Test (Testar), Configure test event (Configurar evento de teste).

Criar eventos de teste compartilháveis

Eventos de teste compartilháveis são eventos de teste que você pode compartilhar com outros usuários na mesma conta da AWS. É possível editar eventos compartilháveis de teste de outros usuários e invocar sua função com esses eventos.

O Lambda salva eventos de teste compartilháveis como esquemas em um registro de esquema da Amazon EventBridge (CloudWatch Events) chamado. `lambda-testevent-schemas` Como o Lambda utiliza esse registro para armazenar e chamar eventos compartilháveis de teste que você cria, recomendamos que não edite esse registro nem crie um registro usando o nome `lambda-testevent-schemas`.

Para ver, compartilhar e editar eventos de teste compartilháveis, você deve ter permissões para todas as seguintes operações da API de registro de esquema EventBridge (CloudWatch Eventos):

- schemas.CreateRegistry
- schemas.CreateSchema
- schemas.DeleteSchema
- schemas.DeleteSchemaVersion
- schemas.DescribeRegistry
- schemas.DescribeSchema
- schemas.GetDiscoveredSchema
- schemas.ListSchemaVersions
- schemas.UpdateSchema

Observe que o salvamento de edições feitas em um evento compartilhável de teste resulta na substituição do respectivo evento.

Se não puder criar, editar ou ver eventos compartilháveis de teste, verifique se sua conta tem as permissões necessárias para essas operações. Se você tiver as permissões necessárias, mas ainda não conseguir acessar eventos de teste compartilháveis, verifique se há políticas baseadas em recursos que possam limitar o acesso ao registro EventBridge (CloudWatch Eventos).

Para criar um evento compartilhável de teste

1. Abra a página Funções do console do Lambda.
2. Escolha o nome da função que deseja testar.
3. Selecione a guia Test (Testar).
4. Em Test event (Evento de teste), faça o seguinte:
    a. Escolha um Template (Modelo).
    b. Insira um Name (Nome) para o teste.
    c. Na caixa de entrada de texto, insira o evento de teste JSON.
    d. Em Event sharing settings (Configurações de compartilhamento de eventos), escolha Shareable (Compartilhável).
5. Escolha Salvar alterações.

Use eventos de teste compartilháveis com AWS Serverless Application Model.

É possível usar o AWS SAM para invocar eventos de teste compartilháveis. Consulte `sam remote test-event` no Guia do desenvolvedor do AWS Serverless Application Model

Excluir esquemas de eventos compartilháveis de teste

Quando você exclui eventos compartilháveis de teste, o Lambda os remove do registro `lambda-testevent-schemas`. Se você remover o último evento compartilhável de teste do registro, o Lambda excluirá o registro.

Se excluir a função, o Lambda não excluirá nenhum esquema associado de evento compartilhável de teste. Você deve limpar esses recursos manualmente no console EventBridge (CloudWatch Eventos).

Estados da função do Lambda

O Lambda inclui um campo de estado na configuração da função para todas as funções para indicar quando a função está pronta para ser invocada. `State` fornece informações sobre o status atual da função, incluindo se você pode invocá-la com êxito. Os estados de função não alteram o comportamento de invocações de função ou como sua função executa o código. Os estados da função incluem:

- `Pending` – Depois que o Lambda cria a função, ele define o estado como pendente. Enquanto estiver em estado pendente, o Lambda tenta criar ou configurar recursos para a função, como recursos de VPC ou EFS. O Lambda não invoca uma função durante o estado pendente. As invocações ou outras ações de API que operam na função falharão.
- `Active` – Sua função transita para o estado ativo depois que o Lambda concluir a configuração e o provisionamento de recursos. As funções só podem ser invocadas com êxito enquanto estiverem ativas.
- `Failed` – Indica que a configuração de recursos ou o provisionamento encontrou um erro.
- `Inactive` – Uma função torna-se inativa quando está ociosa tempo suficiente para que o Lambda recupere os recursos externos que foram configurados para ela. Quando tenta invocar uma função que está inativa, a invocação falha e o Lambda define a função para o estado pendente até que os recursos da função são recriados. Se o Lambda não conseguir recriar os recursos, a função voltará ao estado inativo. Se sua função estiver presa no estado inativo, consulte os atributos `StatusCode` e `StatusCodeReason` da função para obter mais informações de solução de problemas. Talvez seja necessário resolver os erros e reimplantar a função para que ela volte ao estado ativo.

Se você estiver utilizando fluxos de trabalho de automação baseados em SDK ou chamando as APIs de serviço do Lambda diretamente, verifique o estado de uma função antes da invocação para conferir se ela está ativa. Você pode fazer isso com a ação da API do Lambda GetFunction ou configurando um waiter usando o AWS SDK for Java 2.0.

```
aws lambda get-function --function-name my-function --query 'Configuration.[State,
 LastUpdateStatus]'
```

A seguinte saída deverá ser mostrada:

```
[
```

```
 "Active",
 "Successful"
]
```

Ocorre falha nas seguintes operações enquanto a criação da função está pendente:

- Invoke
- UpdateFunctionCode
- UpdateFunctionConfiguration
- PublishVersion

Estados de função durante a atualização

O Lambda fornece contexto adicional para funções submetidas a atualizações com o atributo `LastUpdateStatus`, que pode ter os seguintes status:

- `InProgress` – Uma atualização está acontecendo em uma função existente. Enquanto uma atualização de função está em andamento, as invocações vão para o código e configuração anteriores da função.
- `Successful` – A atualização foi concluída. Assim que o Lambda terminar a atualização, permanecerá definido até uma nova atualização.
- `Failed` – Falha na atualização da função. O Lambda aborta a atualização e o código e a configuração anteriores da função permanecem disponíveis.

Example

O seguinte é o resultado de `get-function-configuration` em uma função submetida a uma atualização.

```
{
    "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 tem dois outros atributos, `LastUpdateStatusReason` e `LastUpdateStatusReasonCode` para ajudar a solucionar os problemas com os de atualização.

Ocorre falha nas seguintes operações enquanto uma atualização assíncrona está em andamento:

- UpdateFunctionCode
- UpdateFunctionConfiguration
- PublishVersion
- TagResource

Compreender o comportamento de novas tentativas no Lambda

Ao invocar um função diretamente, você determinará a estratégia para manipular erros relacionados ao código da função. O Lambda não repete automaticamente esses tipos de erros por você. Para tentar novamente, você pode invocar a função manualmente mais uma vez, enviar o evento com falha para uma fila de depuração ou ignorá-lo. O código de sua função pode ter sido executado completa ou parcialmente, ou nem ter sido executado. Se você tentar novamente, certifique-se de que o código de sua função pode lidar com o mesmo evento várias vezes sem duplicar transações nem ter efeitos colaterais indesejados.

Ao invocar um função indiretamente, você deve estar ciente do comportamento de novas tentativas do invocador e dos serviços que podem interagir com a solicitação no processo. Isso inclui as seguintes situações.

- Asynchronous Invocation (Chamada assíncrona): o Lambda faz duas novas tentativas de erros de função. Se a função não tiver capacidade suficiente para lidar com todas as solicitações em andamento, os eventos poderão ter de aguardar na fila por horas ou dias até serem enviados para a função. Você pode configurar uma fila de mensagens mortas na função para registrar eventos que não foram processadas com êxito. Para ter mais informações, consulte Invocação assíncrona.
- Event source mappings (Mapeamento da fonte do evento): mapeamentos da fonte do evento que leem das transmissões repetem todo o lote de itens. Os erros repetidos bloqueiam o processamento do estilhaço afetado até o erro ser resolvido ou os itens expirarem. Para detectar estilhaços interrompidos, é possível monitorar a métrica Iterator Age (Idade do iterador).

  Para mapeamentos da fonte do evento que leem de uma fila, você mesmo determina o intervalo de tempo entre repetições e o destino de eventos falhos. Para fazer isso, configure o tempo limite de visibilidade e a política de redirecionamento na fila de origem. Para obter mais informações, consulte Como o Lambda processa registros de origens de eventos baseadas em fluxos e filas e os tópicos específicos do serviço em Invocando o Lambda com eventos de outros serviços da AWS.
- Serviços da AWS: os serviços da AWS podem chamar uma função de maneira síncrona ou assíncrona. Para chamada síncrona, o serviço decide se deve tentar novamente. Por exemplo, as operações em lote do Amazon S3 tentam novamente a operação se a função Lambda retornar um código de resposta `TemporaryFailure`. Os serviços que usam proxy em solicitações de um usuário ou cliente de upstream podem ter uma estratégia de repetição ou retransmitir a resposta de erro de volta para o solicitante. Por exemplo, o API Gateway sempre retransmite a resposta de erro de volta para o solicitante.

Para invocação assíncrona, o comportamento é o mesmo de quando você invoca a função de maneira síncrona. Para obter mais informações, consulte os tópicos específicos do serviço em Invocando o Lambda com eventos de outros serviços da AWS e a documentação do serviço de invocação.

- Other Accounts and Clients (Outras contas e clientes): ao conceder acesso a outras contas, você pode usar as políticas baseadas em recursos para restringir os serviços ou recursos que eles podem configurar para chamar sua função. Para evitar que a função fique sobrecarregada, considere colocar uma camada de API na frente da função com o Amazon API Gateway.

Para ajudar você a lidar com erros em aplicações do Lambda, o Lambda se integra com serviços como o Amazon CloudWatch e o AWS X-Ray. Você pode usar um conjunto de logs, métricas, alarmes e rastreamento do para detectar e identificar rapidamente os problemas no código da função, na API ou em outros recursos compatíveis com sua aplicação. Para ter mais informações, consulte Monitoramento e solução de problemas de funções do Lambda.

Usar a detecção de loop recursivo do Lambda para evitar loops infinitos

Ao configurar uma função do Lambda para gerar a saída para o mesmo serviço ou recurso que invoca a função, é possível criar um loop recursivo infinito. Por exemplo, uma função do Lambda pode gravar uma mensagem em uma fila do Amazon Simple Queue Service (Amazon SQS) que, em seguida, invoca a mesma função. Essa invocação faz com que a função grave outra mensagem na fila, que, por sua vez, invoca a função novamente.

Loops recursivos não intencionais podem resultar em cobranças inesperadas na sua Conta da AWS. Os loops também podem fazer com que o Lambda seja escalado e use toda a simultaneidade disponível de sua conta. Para ajudar a reduzir o impacto de loops não intencionais, o Lambda pode detectar determinados tipos de loops recursivos logo após a ocorrência deles. Ao detectar um loop recursivo, o Lambda interrompe a invocação da função e notifica você.

Se o projeto usar padrões recursivos intencionalmente, você poderá solicitar a desativação da detecção de loop recursivo do Lambda. Para solicitar essa alteração, entre em contato com o AWS Support.

> **Important**
>
> Se o projeto usar uma função do Lambda intencionalmente para gravar dados no mesmo recurso da AWS que invoca a função, tenha cuidado e implemente proteções adequadas para evitar incorrer em cobranças inesperadas na Conta da AWS. Para saber mais sobre as práticas recomendadas para usar os padrões de invocação recursiva, consulte Recursive patterns that cause run-away Lambda functions no Serverless Land.

Seções

Compreensão da detecção de loop recursivo

A detecção de loop recursivo no Lambda funciona com o rastreamento de eventos. O Lambda é um serviço de computação orientado a eventos que executa seu código de função quando ocorrem determinados eventos. Por exemplo, quando um item é adicionado a uma fila do Amazon SQS ou a um tópico do Amazon Simple Notification Service (Amazon SNS). O Lambda transmite eventos para a função como objetos JSON, que contêm informações sobre a alteração no estado do sistema. Quando um evento faz com que a função seja executada, isso é chamado de invocação.

Para detectar loops recursivos, o Lambda usa cabeçalhos de rastreamento do AWS X-Ray. Quando os Serviços da AWS compatíveis com a detecção de loop recursivo enviam eventos para o Lambda, esses eventos são anotados automaticamente com metadados. Quando a função do Lambda grava um desses eventos em outro AWS service (Serviço da AWS) compatível usando uma versão compatível de um AWS SDK, ela atualiza esses metadados. Os metadados atualizados incluem uma contagem do número de vezes que o evento invocou a função.

> **Note**
>
> Você não precisa habilitar o rastreamento ativo do X-Ray para que esse recurso funcione. Por padrão, a detecção de loop recursivo é ativada para todos os clientes da AWS. Não há cobranças pelo uso do recurso.

Uma cadeia de solicitações é uma sequência de invocações do Lambda causada pelo mesmo evento de acionamento. Por exemplo, imagine que uma fila do Amazon SQS invoque sua função do Lambda. Em seguida, a função do Lambda envia o evento processado para a mesma fila do Amazon SQS, que invoca a função novamente. Neste exemplo, cada invocação da função é direcionada para a mesma cadeia de solicitações.

Se a função for invocada mais de 16 vezes na mesma cadeia de solicitações, o Lambda interromperá automaticamente a próxima invocação da função nessa cadeia de solicitações e notificará você. Se a função estiver configurada com diversos acionadores, as invocações de outros acionadores não serão afetadas.

> **Note**
>
> Quando a configuração `maxReceiveCount` na política de redrive da fila de origem é maior que 16, a proteção de recursão do Lambda não impede que o Amazon SQS repita

a mensagem depois que um loop recursivo é detectado e encerrado. Quando o Lambda detecta um loop recursivo e descarta as invocações subsequentes, ele retorna uma `RecursiveInvocationException` para o mapeamento da origem do evento. Isso incrementa o valor `receiveCount` da mensagem. O Lambda continua refazendo a tentativa da mensagem e continua bloqueando as invocações de funções, até que o Amazon SQS determine que o `maxReceiveCount` foi excedido e envie a mensagem para a fila de mensagens não entregues configurada.

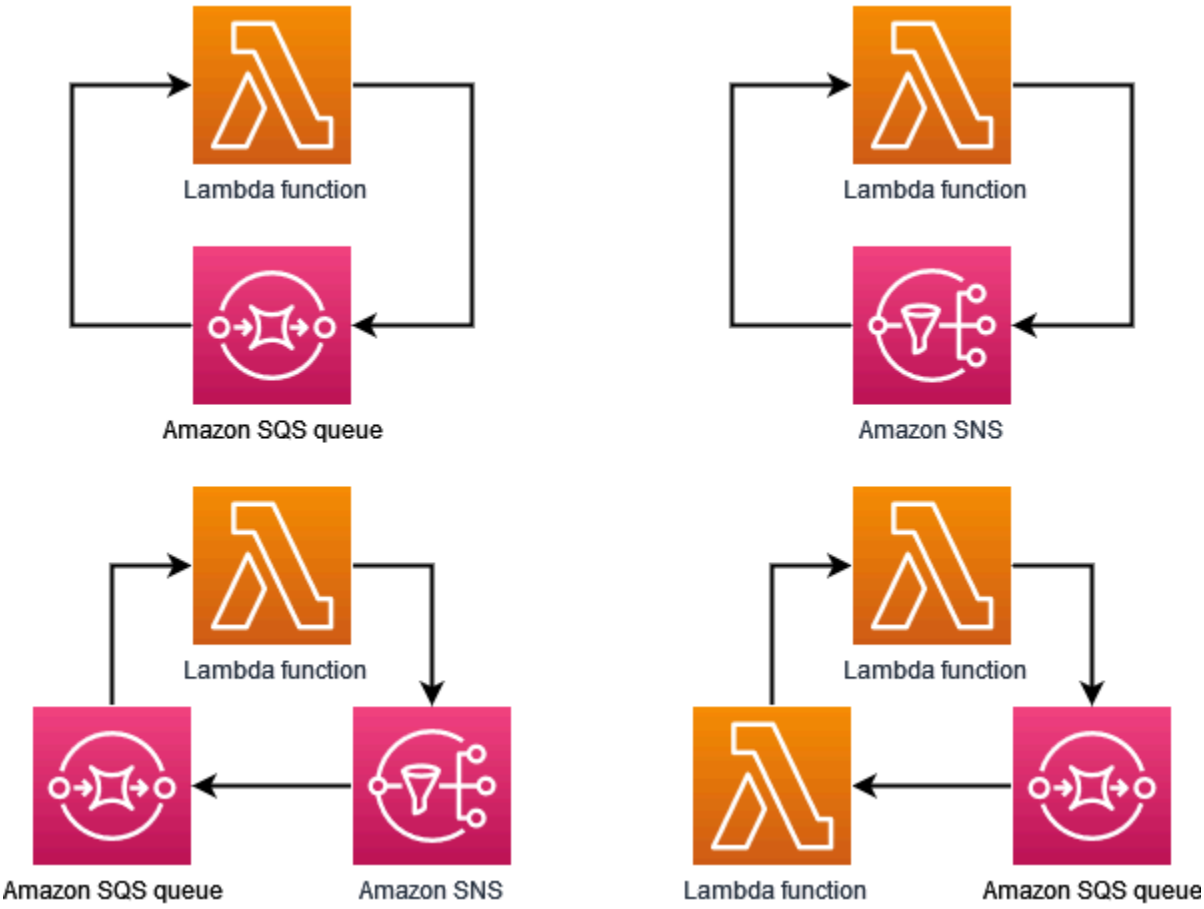
Se você tiver um destino em caso de falha ou uma fila de mensagens não entregues configurados para a função, o Lambda também enviará o evento da invocação interrompida para o destino ou a fila de mensagens não entregues. Se você configurar um destino ou uma fila de mensagens não entregues para a função, certifique-se de não usar um tópico do Amazon SNS ou uma fila do Amazon SQS que a função também use como um acionador de evento ou mapeamento da origem do evento. Se você enviar eventos para o mesmo recurso que invoca a função, poderá criar outro loop recursivo.

Serviços da AWS e SDKs compatíveis

O Lambda pode detectar somente loops recursivos que incluem determinados Serviços da AWS compatíveis. Para que os loops recursivos sejam detectados, a função também deve usar um dos AWS SDKs compatíveis.

Com suporte Serviços da AWS

No momento, o Lambda detecta loops recursivos entre suas funções, o Amazon SQS e o Amazon SNS. O Lambda também detecta loops compostos somente por funções do Lambda, que podem invocar umas às outras de forma síncrona ou assíncrona. Os seguintes diagramas apresentam alguns exemplos de loops que o Lambda pode detectar:



Quando outro AWS service (Serviço da AWS), como o Amazon DynamoDB ou o Amazon Simple Storage Service (Amazon S3), faz parte do loop, o Lambda não pode detectá-lo e interrompê-lo.

No momento, como o Lambda detecta somente loops recursivos envolvendo o Amazon SQS e o Amazon SNS, ainda é possível que loops envolvendo outros Serviços da AWS possam resultar no uso não intencional de suas funções do Lambda.

Para evitar cobranças inesperadas em sua Conta da AWS, recomendamos que você configure os alarmes do Amazon CloudWatch para alertar sobre padrões de uso incomuns. Por exemplo, é possível configurar o CloudWatch para notificar você sobre picos na simultaneidade ou nas invocações da função do Lambda. Também é possível configurar um alarme de faturamento para notificar você quando os gastos em sua conta excederem um limite especificado. Como alternativa, você pode usar o AWS Cost Anomaly Detection para alertas sobre padrões de faturamento incomuns.

AWS SDKs compatíveis

Para que o Lambda detecte loops recursivos, a função deve usar uma das seguintes versões do SDK ou superiores:

Runtime	Versão mínima exigida do AWS SDK
Node.js	2.1147.0 (SDK versão 2) 3.105.0 (SDK versão 3)
Python	1.24.46 (boto3) 1.27.46 (botocore)
Java 8 e Java 11	1.12.200 (SDK versão 1) 2.17.135 (SDK versão 2)
Java 17	2.20.81
Java 21	2.21.24
.NET	3.7.293.0
Ruby	3.134.0
PHP	3.232.0

Alguns runtimes do Lambda, como Python e Node.js, incluem uma versão do AWS SDK. Se a versão do SDK inclusa no runtime de sua função for inferior ao mínimo exigido, você poderá adicionar uma versão compatível do SDK ao pacote de implantação da função. Você também pode adicionar uma versão do SDK compatível à função usando uma camada do Lambda. Para obter uma lista dos SDKs inclusos em cada runtime do Lambda, consulte Runtimes do Lambda.

A detecção de recursão do Lambda não é compatível com o runtime Go do Lambda.

Notificações de loop recursivo

Quando o Lambda interrompe um loop recursivo, você recebe notificações por meio do AWS Health Dashboard e por e-mail. Você também pode usar as métricas do CloudWatch para monitorar o número de invocações recursivas que o Lambda interrompeu.

Notificações do AWS Health Dashboard

Quando o Lambda interrompe uma invocação recursiva, o AWS Health Dashboard exibe uma notificação na página de integridade da sua conta, em Problemas abertos e recentes. Observe que pode demorar até três horas após o Lambda interromper uma invocação recursiva até que essa notificação seja exibida. Para obter mais informações sobre como visualizar eventos da conta no AWS Health Dashboard, consulte Getting started with your AWS Health Dashboard – Your account health no Guia do usuário do AWS Health.

Alertas de e-mail

Quando o Lambda interrompe uma invocação recursiva da função pela primeira vez, ele envia um alerta por e-mail. O Lambda envia, no máximo, um e-mail a cada 24 horas para cada função em sua Conta da AWS. Após o Lambda enviar uma notificação por e-mail, você não receberá mais e-mails sobre essa função por mais 24 horas, mesmo que o Lambda interrompa outras invocações recursivas da função. Observe que pode demorar até três horas após o Lambda interromper uma invocação recursiva até que você receba este alerta por e-mail.

O Lambda envia alertas de e-mail de loop recursivo para o contato principal da conta e para o contato alternativo de operações da sua Conta da AWS. Para obter informações sobre como visualizar ou atualizar os endereços de e-mail em sua conta, consulte Updating contact information na Referência geral da AWS.

Métricas do Amazon CloudWatch

A métrica do CloudWatch `RecursiveInvocationsDropped` registra o número de invocações de função que o Lambda interrompeu porque a função foi invocada mais de 16 vezes em uma única cadeia de solicitações. O Lambda emite essa métrica logo após interromper uma invocação recursiva. Para visualizar essa métrica, siga as instruções em Exibir métricas no console do CloudWatch e escolha a métrica `RecursiveInvocationsDropped`.

Como responder a notificações de detecção de loop recursivo

Quando a função é invocada mais de 16 vezes pelo mesmo evento de acionamento, o Lambda interrompe a próxima invocação de função desse evento para interromper o loop recursivo. Para evitar a recorrência de um loop recursivo que o Lambda interrompeu, faça o seguinte:

- Reduza a simultaneidade disponível da função para zero, pois isso limita todas as invocações futuras.
- Remova ou desabilite o acionador ou o mapeamento da origem do evento que está invocando a função.
- Identifique e corrija defeitos de código que gravam eventos no recurso da AWS que está invocando a função. Uma origem comum de defeitos ocorre quando você usa variáveis para definir a origem e o destino do evento de uma função. Verifique se você não está usando o mesmo valor para ambas as variáveis.

Além disso, se a origem do evento para a função do Lambda for uma fila do Amazon SQS, considere configurar uma fila de mensagens não entregues na fila de origem.

> **Note**
>
> Certifique-se de configurar a fila de mensagens não entregues na fila de origem, e não na função do Lambda. A fila de mensagens não entregues que você configura em uma função é usada para a fila de invocação assíncrona da função, e não para filas de origem de evento.

Se a origem do evento for um tópico do Amazon SNS, considere adicionar um destino em caso de falha para a função.

Reduzir a simultaneidade disponível da função para zero (console)

1. Abra a página Funções do console do Lambda.
2. Escolha o nome da sua função.
3. Escolha Controlar.
4. Na caixa de diálogo Controlar a função, escolha Confirmar.

Remover um acionador ou mapeamento da origem do evento para a função (console)

1. Abra a página Funções do console do Lambda.
2. Escolha o nome da sua função.
3. Escolha a guia Configuração e, em seguida, escolha Acionadores.
4. Em Acionadores, selecione o acionador ou o mapeamento da origem do evento que deseja excluir e, em seguida, escolha Excluir.
5. Na caixa de diálogo Excluir acionadores, escolha Excluir.

Desabilitar um mapeamento da origem do evento para a função (AWS CLI)

1. Para descobrir o UUID para o mapeamento da origem do evento que você deseja desabilitar, execute o comando list-event-source-mappings da AWS Command Line Interface (AWS CLI).

```
aws lambda list-event-source-mappings
```

2. Para desabilitar o mapeamento da origem do evento, execute o comando update-event-source-mapping da AWS CLI apresentado a seguir.

```
aws lambda update-event-source-mapping --function-name MyFunction \
--uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 --no-enabled
```

URLs de função do Lambda

Um URL de função do Lambda é um endpoint HTTP(S) dedicado para a função do Lambda. Você pode criar e configurar um URL de função no console do Lambda ou na API do Lambda. Ao criar um URL de função, o Lambda gera automaticamente um endpoint de URL exclusivo para você. Após a criação de um URL de função, o endpoint de URL nunca muda. Os endpoints de URLs de função têm o seguinte formato:

```
https://<url-id>.lambda-url.<region>.on.aws
```

> **Note**
>
> Não há suporte para URLs de funções nas seguintes regiões: Ásia-Pacífico (Hyderabad) (`ap-south-2`), Ásia-Pacífico (Melbourne) (`ap-southeast-4`), Oeste do Canadá (Calgary) (`ca-west-1`), Europa (Espanha) (`eu-south-2`), Europa (Zurique) (), Israel (Tel Aviveu-`central-2`) () e Oriente Médio (`il-central-1`EAU) (). `me-central-1`

Os URLs de função são habilitados para pilha dupla, sendo compatíveis com IPv4 e IPv6. Depois de configurar um URL de função para a sua função, você pode invocar a função por meio de seu endpoint HTTP(S) via um navegador da Web, curl, Postman ou qualquer cliente HTTP.

> **Note**
>
> Você só pode acessar a URL da função pela Internet pública. Embora as funções do Lambda sejam compatíveis com o AWS PrivateLink, as URLs de função não o são.

Os URLs de função do Lambda usam políticas baseadas em recursos para segurança e controle de acesso. Os URLs de função também são compatíveis com opções de configuração de compartilhamento de recursos de origem cruzada (CORS).

Você pode aplicar URLs de função a qualquer alias de função ou à versão da função não publicada `$LATEST`. Você não pode adicionar um URL de função a nenhuma outra versão da função.

Tópicos

- Criar e gerenciar URLs de função do Lambda

- Controlar o acesso aos URLs de função do Lambda
- Invocar URLs de função do Lambda
- Monitorar URLs de função do Lambda
- Tutorial: criar uma função do Lambda com um URL de função

Criar e gerenciar URLs de função do Lambda

Um URL de função é um endpoint HTTP(S) dedicado para a função do Lambda. Você pode criar e configurar um URL de função no console do Lambda ou na API do Lambda. Ao criar um URL de função, o Lambda gera automaticamente um endpoint de URL exclusivo para você. Após a criação de um URL de função, o endpoint de URL nunca muda. Os endpoints de URLs de função têm o seguinte formato:

```
https://<url-id>.lambda-url.<region>.on.aws
```

> **Note**
>
> As URLs de função não são compatíveis com as seguintes regiões: Ásia-Pacífico (Hyderabad) (`ap-south-2`), Ásia-Pacífico (Melbourne) (`ap-southeast-4`), Canadá Ocidental (Calgary) (`ca-west-1`), Europa (Espanha) (`eu-south-2`), Europa (Zurique) (`eu-central-2`), Israel (Tel Aviv) (`il-central-1`) e Oriente Médio (EAU) (`me-central-1`).

A seção a seguir mostra como criar e gerenciar um URL da função usando o console do Lambda, a AWS CLI e o modelo do AWS CloudFormation.

Tópicos

Criar um URL de função (console)

Siga estas etapas para criar um URL de função usando o console.

Para criar um URL de função para uma função existente (console)

1. Abra a página Funções do console do Lambda.

2. Escolha o nome da função para a qual você deseja criar o URL de função.
3. Escolha a guia Configuration (Configuração) e depois Function URL (URL de função).
4. Escolha Create function URL (Criar URL de função).
5. Para Auth type (Tipo de autenticação), escolha AWS_IAM ou NONE (NENHUM). Para obter mais informações sobre autenticação de URLs de função, consulte Controle de acesso.
6. (Opcional) Selecione Configure cross-origin resource sharing (CORS) [Configurar compartilhamento de recursos de origem cruzada (CORS)] e defina as configurações de CORS para o URL de função. Para obter mais informações sobre CORS, consulte Compartilhamento de recursos de origem cruzada (CORS).
7. Escolha Salvar.

Isso cria um URL de função para a versão da função não publicada `$LATEST`. O URL de função aparece na seção Function overview (Visão geral de funções) do console.

Para criar um URL de função para um alias existente (console)

1. Abra a página Funções do console do Lambda.
2. Escolha o nome da função com o alias para o qual você deseja criar o URL de função.
3. Escolha a guia Aliases e depois escolha o nome do alias para o qual você deseja criar o URL de função.
4. Escolha a guia Configuration (Configuração) e depois Function URL (URL de função).
5. Escolha Create function URL (Criar URL de função).
6. Para Auth type (Tipo de autenticação), escolha AWS_IAM ou NONE (NENHUM). Para obter mais informações sobre autenticação de URLs de função, consulte Controle de acesso.
7. (Opcional) Selecione Configure cross-origin resource sharing (CORS) [Configurar compartilhamento de recursos de origem cruzada (CORS)] e defina as configurações de CORS para o URL de função. Para obter mais informações sobre CORS, consulte Compartilhamento de recursos de origem cruzada (CORS).
8. Escolha Salvar.

Isso cria um URL de função para o alias da função. O URL de função aparece na seção Function overview (Visão geral de funções) do console para o alias.

Para criar uma nova função com um URL de função (console)

Para criar uma nova função com um URL de função (console)

1. Abra a página Funções do console do Lambda.
2. Escolha a opção Criar função.
3. Em Basic information (Informações básicas), faça o seguinte:
   a. Em Function Name (Nome da função), insira o nome da função, como **`my-function`**.
   b. Em Runtime, escolha o runtime da linguagem de sua preferência, como Node.js 18.x.
   c. Para Architecture (Arquitetura), escolha x86_64 ou arm64.
   d. Expanda Permissions (Permissões) e depois escolha se vai criar uma nova função de execução ou usar uma já existente.
4. Expanda Advanced settings (Configurações avançadas) e selecione Function URL (URL de função).
5. Para Auth type (Tipo de autenticação), escolha AWS_IAM ou NONE (NENHUM). Para obter mais informações sobre autenticação de URLs de função, consulte Controle de acesso.
6. (Opcional) Selecione Configure cross-origin resource sharing (CORS) [(Configurar compartilhamento de recursos de origem cruzada (CORS)]. Se essa opção for selecionada durante a criação da função, o URL de função permitirá solicitações de todas as origens por padrão. Você pode editar as configurações de CORS para o URL de função após criar a função. Para obter mais informações sobre CORS, consulte Compartilhamento de recursos de origem cruzada (CORS).
7. Escolha a opção Criar função.

Isso cria um novo URL de função para a versão da função não publicada `$LATEST`. O URL de função aparece na seção Function overview (Visão geral de funções) do console.

Criar um URL de função (AWS CLI)

Para criar uma URL de função para uma função existente do Lambda usando a AWS Command Line Interface (AWS CLI), execute o seguinte comando:

```
aws lambda create-function-url-config \
    --function-name my-function \
    --qualifier prod \ // optional
```

```
    --auth-type AWS_IAM
    --cors-config {AllowOrigins="https://example.com"} // optional
```

Isso adiciona um URL de função ao qualificador **`prod`** da função **`my-function`**. Para obter mais informações sobre esses parâmetros de configuração, consulte CreateFunctionURLConfig na referência da API.

> **Note**
>
> Para criar um URL de função por meio da AWS CLI, a função já deve existir.

Adicionar um URL de função a um modelo do CloudFormation

Para adicionar um recurso de `AWS::Lambda::Url` ao modelo do AWS CloudFormation, use a seguinte sintaxe:

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
```

Parâmetros

- (Obrigatório) `AuthType`: define o tipo de autenticação para o URL de função. Os valores possíveis são `AWS_IAM` ou `NONE`. Para restringir o acesso somente para usuários autenticados, defina como `AWS_IAM`. Para ignorar a autenticação do IAM e permitir que qualquer usuário faça solicitações à a função, defina como `NONE`.
- (Optional) `Cors`: defina CORS settings (Configurações de CORS) para o URL de função. Para adicionar `Cors` ao recurso `AWS::Lambda::Url` no CloudFormation, use a sintaxe a seguir.

  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
  ```

- (Optional) `Qualifier`: o nome do alias.
- (Obrigatório) `TargetFunctionArn`: o nome ou o nome do recurso da Amazon (ARN) da função do Lambda. Os formatos de nome válidos incluem o seguinte:
  - Nome da função: `my-function`
  - ARN da função: `arn:aws:lambda:us-west-2:123456789012:function:my-function`
  - ARN parcial: `123456789012:function:my-function`

Compartilhamento de recursos de origem cruzada (CORS)

Para definir como diferentes origens podem acessar o URL de função, use Compartilhamento de recursos de origem cruzada (CORS). Recomendamos configurar o CORS se você pretender chamar o URL de função em um domínio diferente. O Lambda é compatível com os cabeçalhos de CORS a seguir para URLs de função.

Cabeçalhos de CORS	Propriedade da configuração de CORS	Exemplos de valores

Access-Control-Allow-Origin	`AllowOrigins`	`*` (permitir todas as origens)
		`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` (padrão), `300`

Quando você configura o CORS para um URL de função usando o console do Lambda ou a AWS CLI, o Lambda adiciona automaticamente os cabeçalhos de CORS a todas as respostas por meio do URL de função. Como alternativa, você pode adicionar manualmente cabeçalhos de CORS à resposta da função. Se houver cabeçalhos conflitantes, os cabeçalhos de CORS configurados no URL de função terão precedência.

Controlar a utilização de URLs de função

O controle de utilização limita a taxa na qual a função processa as solicitações. Isso é útil em muitas situações, como para impedir que a função sobrecarregue recursos downstream ou para lidar com um aumento repentino de solicitações.

Você pode limitar a taxa de solicitações que a função do Lambda processa por meio de um URL de função configurando a simultaneidade reservada. A simultaneidade reservada limita o número máximo de invocações simultâneas para a função. A taxa máxima de solicitação por segundo (RPS) da função é equivalente a 10 vezes a simultaneidade reservada configurada. Por exemplo, se você configurar a função com uma simultaneidade reservada de 100, o RPS máximo será 1.000.

Sempre que a simultaneidade da função exceder a simultaneidade reservada, o URL de função retornará um código de status HTTP 429. Se a função receber uma solicitação que exceda o máximo de 10x RPS com base na simultaneidade reservada configurada, você também receberá um erro HTTP 429. Para obter mais informações sobre simultaneidade reservada, consulte Configurar a simultaneidade reservada para uma função.

Desativar URLs de função

Em uma emergência, talvez você queira rejeitar todo o tráfego para o URL de função. Para reativar o URL de função, defina a simultaneidade reservada como zero. Isso controla a utilização de todas as solicitações ao URL de função, gerando respostas de status HTTP 429. Para reativar o URL de função, exclua a configuração de simultaneidade reservada ou defina a configuração como um valor maior que zero.

Excluir URLs de funções

Quando você exclui um URL de função, não pode recuperá-lo. A criação de um novo URL de função resultará em um endereço URL diferente.

> **Note**
>
> Se você excluir um URL da função com o tipo de autenticação `NONE`, o Lambda não excluirá a política baseada em recursos associada a ela. Se quiser excluir essa política, deverá excluí-la manualmente.

1. Abra a página Funções do console do Lambda.

2. Escolha o nome da função.
3. Escolha a guia Configuration (Configuração) e depois Function URL (URL de função).
4. Escolha Excluir.
5. Insira a palavra delete (excluir) no campo para confirmar a exclusão.
6. Escolha Excluir.

> **Note**
>
> Quando você exclui uma função que tem uma URL da função, o Lambda a exclui a URL da função de modo assíncrono. Se você criar imediatamente uma nova função com o mesmo nome e na mesma conta, é possível que a URL da função original seja mapeada para a nova função em vez de ser excluída.

Controlar o acesso aos URLs de função do Lambda

É possível controlar o acesso a URLs de função do Lambda usando o parâmetro `AuthType` combinado com as políticas baseadas em recursos anexadas à função específica. A configuração desses dois componentes determina quem pode invocar ou executar outras ações administrativas no URL de função.

O parâmetro `AuthType` determina como o Lambda autentica ou autoriza solicitações ao URL de função. Ao configurar o URL de função, você deve especificar uma das seguintes opções de `AuthType`:

- `AWS_IAM`: o Lambda usa o AWS Identity and Access Management (IAM) para autenticar e autorizar solicitações com base na política de identidade da entidade principal do IAM e na política baseada em recursos da função. Escolha essa opção se quiser que apenas usuários e perfis autenticados do invoquem a função por meio do URL de função.

- `NONE`: o Lambda não realiza nenhuma autenticação antes de invocar a função. Porém, a política baseada em recursos da função está sempre em vigor e deve conceder acesso público antes que o URL de função possa receber solicitações. Escolha essa opção para permitir acesso público e não autenticado ao URL de função.

Além de `AuthType`, você também pode usar políticas baseadas em recursos para conceder permissões a outras Contas da AWS para invocar a função. Para ter mais informações, consulte Trabalhar com políticas baseadas em recursos no Lambda.

Para obter informações adicionais sobre segurança, você pode usar AWS Identity and Access Management Access Analyzer para obter uma análise abrangente do acesso externo ao URL de função. O IAM Access Analyzer também monitora permissões novas ou atualizadas nas funções do Lambda para ajudá-lo a identificar as permissões que concedem acesso público e acesso entre contas. O IAM Access Analyzer é gratuito para qualquer cliente da AWS. Para começar a usar o IAM Access Analyzer, consulte Usando o AWS IAM Access Analyzer.

Esta página contém exemplos de políticas baseadas em recursos para ambos os tipos de autenticação e também mostra como criar essas políticas usando a operação da API AddPermission ou o console do Lambda. Para obter informações sobre como invocar o URL de função depois de configurar as permissões, consulte Invocar URLs de função do Lambda.

Tópicos

- Usar o tipo de autenticação AWS_IAM
- Usar o tipo de autenticação NONE
- Governança e controle de acesso

Usar o tipo de autenticação **AWS_IAM**

Se você escolher o tipo de autenticação `AWS_IAM`, os usuários que precisarem invocar a URL de função do Lambda deverão ter a permissão `lambda:InvokeFunctionUrl`. Dependendo de quem faz a solicitação de invocação, pode ser necessário conceder essa permissão usando uma política baseada em recursos.

Se a entidade principal fazendo a solicitação estiver na mesma Conta da AWS que o URL de função, a entidade principal deve ou ter permissões `lambda:InvokeFunctionUrl` na política baseada em identidade ou ter permissões concedidas na política baseada em recursos da função. Em outras palavras, uma política baseada em recursos será opcional se o usuário já tiver permissões `lambda:InvokeFunctionUrl` na política baseada em identidade. A avaliação da política segue as regras descritas em Determinar se uma solicitação é permitida ou negada em uma conta.

Se a entidade principal fazendo a solicitação estiver em uma conta diferente, a entidade principal deverá ter ambas, uma política baseada em identidade que forneça a ela as permissões `lambda:InvokeFunctionUrl` e permissões concedidas em uma política baseada em recursos para a função que ela está tentando invocar. Nesses casos entre contas, a avaliação da política segue as regras descritas em Determinar se uma solicitação entre contas é permitida.

Para obter um exemplo de interação entre contas, a política baseada em recursos a seguir permite que a função `example` na Conta da AWS `444455556666` invoque o URL de função associado à função `my-function`:

Example política de invocação entre contas de URL de função

```
{
    "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"
                }
            }
        }
    ]
}
```

Você pode criar essa instrução da política por meio do console seguindo estas etapas:

Como conceder permissões de invocação de URL a outra conta (console)

1. Abra a página Funções do console do Lambda.
2. Escolha o nome da função para a qual deseja conceder permissões de invocação de URL.
3. Escolha a guia Configuration (Configuração) e, depois, Permissions (Permissões).
4. Em Resource-based policy (Política baseada em recursos), escolha Add permissions (Adicionar permissões).
5. Escolha Function URL (URL de função).
6. Para Auth type (Tipo de autenticação), escolha AWS_IAM.
7. (Opcional) Para Statement ID (ID da instrução), insira um ID de instrução para a instrução da política.
8. Em Entidade principal, insira o ID da conta ou o nome do recurso da Amazon (ARN) do usuário ou da função para a qual você deseja conceder permissões. Por exemplo: **444455556666**.
9. Escolha Salvar.

Como alternativa, você pode criar essa instrução de política usando o seguinte comando add-permission da 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
```

No exemplo anterior, a chave-valor da condição `lambda:FunctionUrlAuthType` é `AWS_IAM`. Essa política só permite acesso quando o tipo de autenticação do URL de função também for `AWS_IAM`.

Usar o tipo de autenticação **NONE**

> **Important**
>
> Quando o tipo de autenticação do URL de função for `NONE` e você tiver uma política baseada em recursos que concede acesso público, qualquer usuário não autenticado com o URL de função poderá invocar a função.

Em alguns casos, você pode querer que o URL de função seja público. Por exemplo, você pode querer atender a solicitações feitas diretamente de um navegador da Web. Para permitir acesso público ao URL de função, escolha o tipo de autenticação `NONE`.

Se você escolher o tipo de autenticação `NONE`, o Lambda não usará o IAM para autenticar solicitações ao URL de função. Porém, os usuários ainda precisam ter permissões `lambda:InvokeFunctionUrl` para invocar com êxito o URL de função. Você pode conceder as permissões `lambda:InvokeFunctionUrl` usando a seguinte política baseada em recursos:

Example política de invocação de URL de função para todas as entidades principais não autenticadas

```
{
    "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 você cria um URL de função com o tipo de autenticação `NONE` por meio do console ou da AWS Serverless Application Model(AWS SAM), o Lambda cria automaticamente a declaração de política baseada em recursos anterior para você. Se a política já existir ou se o usuário ou função que está criando a aplicação não tiver as permissões apropriadas, o Lambda não a criará para você. Se estiver usando a AWS CLI, o AWS CloudFormation ou a API do Lambda diretamente, você mesmo deverá adicionar as permissões `lambda:InvokeFunctionUrl`. Isso torna a função pública.
> Além disso, se você excluir o URL da função com o tipo de autenticação `NONE`, o Lambda não excluirá a política baseada em recursos associada a ela. Se quiser excluir essa política, deverá excluí-la manualmente.

Nesta instrução, a chave-valor da condição `lambda:FunctionUrlAuthType` é `NONE`. Essa instrução da política só permite acesso quando o tipo de autenticação do URL de função também é `NONE`.

Se a política baseada em recursos para uma função não conceder permissões `lambda:invokeFunctionUrl`, os usuários receberão um código de erro 403 Forbidden (Proibido) quando tentarem invocar o URL de função, mesmo que o URL de função use o tipo de autenticação `NONE`.

Governança e controle de acesso

Além das permissões de invocação de URL de função, você também pode controlar o acesso nas ações usadas para configurar URLs de função. O Lambda é compatível com as seguintes ações da política do IAM para URLs de função:

- `lambda:InvokeFunctionUrl`: chamar uma função do Lambda usando o URL de função.
- `lambda:CreateFunctionUrlConfig`: criar um URL de função e definir seu `AuthType`.
- `lambda:UpdateFunctionUrlConfig`: atualizar a configuração de um URL de função e seu `AuthType`.
- `lambda:GetFunctionUrlConfig`: visualizar os detalhes de um URL de função.
- `lambda:ListFunctionUrlConfigs`: listar as configurações de URL de função.

- `lambda:DeleteFunctionUrlConfig`: excluir um URL de função.

> **Note**
>
> O console do Lambda é compatível com a adição de permissões apenas para `lambda:InvokeFunctionUrl`. Para todas as outras ações, você deve adicionar permissões usando a API do Lambda ou a AWS CLI.

Para permitir ou negar acesso a URLs de função a outras entidades da AWS, inclua essas ações nas políticas do IAM. Por exemplo, a política a seguir concede à função `example` na Conta da AWS `444455556666` permissões para atualizar o URL de função para a função **`my-function`** na conta `123456789012`.

Example política de URL de função entre contas

```
{
    "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"
        }
    ]
}
```

Chaves de condição

Para ter um controle de acesso refinado sobre os URLs de função, use uma chave de condição. O Lambda é compatível com uma chave de condição adicional para URLs de função:`FunctionUrlAuthType`. A chave `FunctionUrlAuthType` define um valor de enumerador descrevendo o tipo de autenticação que o URL de função usa. O valor pode ser `AWS_IAM` ou `NONE`.

É possível usar essa chave de condição em políticas associadas à função. Por exemplo, você pode restringir quem pode fazer alterações nas configurações dos URLs de função. Para negar todas as

solicitações `UpdateFunctionUrlConfig` a qualquer função com o tipo de autenticação de URL `NONE`, você pode definir a seguinte política:

Example política de URL de função com negação explícita

```
{
    "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"
                }
            }
        }
    ]
}
```

Para conceder à função `example` na Conta da AWS `444455556666` permissões para fazer solicitações `CreateFunctionUrlConfig` e `UpdateFunctionUrlConfig` em funções com o tipo de autenticação de URL `AWS_IAM`, você pode definir a seguinte política:

Example política de URL de função com permissão explícita

```
{
    "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"
                }
            }
        }
    ]
}
```

Você também pode usar essa chave de condição em uma política de controle de serviço (SCP). Use SCPs para gerenciar permissões em toda a organização em AWS Organizations. Por exemplo, para não permitir que usuários criem ou atualizem URLs de função que usem qualquer tipo de autenticação, exceto `AWS_IAM`, use a seguinte política de controle de serviço:

Example SCP de URL de função com negação explícita

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action":[
                "lambda:CreateFunctionUrlConfig",
                "lambda:UpdateFunctionUrlConfig"
            ],
            "Resource": "arn:aws:lambda:*:123456789012:function:*",
            "Condition": {
                "StringNotEquals": {
                    "lambda:FunctionUrlAuthType": "AWS_IAM"
                }
            }
        }
    ]
}
```

Invocar URLs de função do Lambda

Um URL de função é um endpoint HTTP(S) dedicado para a função do Lambda. Você pode criar e configurar um URL de função no console do Lambda ou na API do Lambda. Ao criar um URL de função, o Lambda gera automaticamente um endpoint de URL exclusivo para você. Após a criação de um URL de função, o endpoint de URL nunca muda. Os endpoints de URLs de função têm o seguinte formato:

```
https://<url-id>.lambda-url.<region>.on.aws
```

> **Note**
>
> Não há suporte para URLs de funções nas seguintes regiões: Ásia-Pacífico (Hyderabad) (`ap-south-2`), Ásia-Pacífico (Melbourne) (`ap-southeast-4`), Oeste do Canadá (Calgary) (`ca-west-1`), Europa (Espanha) (`eu-south-2`), Europa (Zurique) (), Israel (Tel Aviv`eu-central-2`) () e Oriente Médio (`il-central-1`EAU) (). `me-central-1`

Os URLs de função são habilitados para pilha dupla, sendo compatíveis com IPv4 e IPv6. Após configurar o URL de função, você pode invocar a função por meio de seu endpoint HTTP(S) via um navegador da Web, curl, Postman ou qualquer cliente HTTP. Para invocar um URL de função, você deve ter permissões `lambda:InvokeFunctionUrl`. Para ter mais informações, consulte Controle de acesso.

Tópicos

- Noções básicas sobre invocação de URL de função
- Cargas úteis de solicitações e respostas

Noções básicas sobre invocação de URL de função

Se o URL de função usar o tipo de autenticação, `AWS_IAM`você deve assinar cada solicitação HTTP usando a {AWS Signature Version 4 (SigV4)}. Ferramentas como awscurl, Postman e AWSProxy SigV4 oferecem maneiras integradas de assinar solicitações com o Sigv4.

Se não usar uma ferramenta para assinar as solicitações HTTP ao URL de função, você deverá assinar manualmente cada solicitação usando o SigV4. Quando o URL de função recebe uma solicitação, o Lambda também analisa a assinatura do Sigv4. O Lambda só processa a solicitação

se as assinaturas corresponderem. Para obter instruções sobre como assinar manualmente as solicitações com o SigV4, consulte Assinar solicitações da AWS com o Signature versão 4, no Guia Referência geral da Amazon Web Services.

Se o URL de função usar o tipo de autenticação `NONE`, você não precisará assinar as solicitações usando o Sigv4. Você pode invocar a função usando um navegador da Web, curl, Postman ou qualquer cliente HTTP.

Para testar simples solicitações `GET` à função, use um navegador da Web. Por exemplo, se o URL de função for `https://abcdefg.lambda-url.us-east-1.on.aws` e aceitar um parâmetro de string `message`, o URL de solicitação pode ser semelhante a este:

```
https://abcdefg.lambda-url.us-east-1.on.aws/?message=HelloWorld
```

Para testar outras solicitações HTTP, como uma solicitação `POST`, você pode usar uma ferramenta como curl. Por exemplo, se quiser incluir alguns dados JSON em uma solicitação `POST` ao URL de função, você poderá usar o seguinte comando curl:

```
curl -v 'https://abcdefg.lambda-url.us-east-1.on.aws/?message=HelloWorld' \
-H 'content-type: application/json' \
-d '{ "example": "test" }'
```

Cargas úteis de solicitações e respostas

Quando um cliente chama o URL de função, o Lambda mapeia a solicitação para um objeto de evento antes de passá-la para a função. A resposta da função é então mapeada para uma resposta HTTP que o Lambda envia de volta ao cliente por meio do URL de função.

Os formatos de evento de solicitação e resposta seguem o mesmo esquema do formato de carga útil do Amazon API Gateway versão 2.0.

Formato de carga útil de solicitação

Uma carga útil de solicitação tem a seguinte estrutura:

```
{
  "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
}
```

Parâmetro	Descrição	Exemplo
`version`	A versão do formato de carga útil para esse evento. Atualmente, os URLs de função do Lambda são compatíveis com o formato de carga útil versão 2.0.	`2.0`
`routeKey`	URLs de função não usam esse parâmetro. O Lambda define isso como `$default` para marcar um espaço reservado.	`$default`
`rawPath`	O caminho da solicitação. Por exemplo, se o URL de solicitação for `https://{url-id}.lambda-url.{region}.on.aws/example/test/demo` , o valor do caminho bruto será `/example/test/demo` .	`/example/test/demo`
`rawQueryString`	A string bruta que contém os parâmetros de string de consulta da solicitação. Os caracteres compatíveis incluem `a-z`, `A-Z`, `0-9`, `.`, `_`, `-`, `%`, `&`, `=` e `+`.	`"?parameter1=value1&parameter2=value2"`
`cookies`	Uma matriz contendo todos os cookies enviados como parte da solicitação.	`["Cookie_1=Value_1", "Cookie_2=Value_2"]`

Parâmetro	Descrição	Exemplo
`headers`	A lista de cabeçalhos de solicitação, apresentada como pares chave-valor.	`{"header1": "value1", "header2": "value2"}`
`queryStringParameters`	Os parâmetros de consulta para a solicitação. Por exemplo, se o URL de solicitação for `https://{url-id}.lambda-url.{region}.on.aws/example?name=Jane` , o valor `queryStringParameters` será um objeto JSON com uma chave `name` e um valor `Jane`.	`{"name": "Jane"}`
`requestContext`	Um objeto que contém informações adicionais sobre a solicitação, como o `requestId` , a hora da solicitação e a identidade do chamador se autorizado pelo AWS Identity and Access Management (IAM).	
`requestContext.accountId`	O ID da Conta da AWS do proprietário da função.	`"123456789012"`
`requestContext.apiId`	O ID do URL de função.	`"33anwqw8fj"`
`requestContext.authentication`	URLs de função não usam esse parâmetro. O Lambda define isso como `null`.	`null`

Parâmetro	Descrição	Exemplo
`requestContext.authorizer`	Um objeto que contém informações sobre a identidade do chamador, se o URL de função usar o tipo de autenticação `AWS_IAM`. Do contrário, o Lambda define isso como `null`.	
`requestContext.authorizer.iam.accessKey`	A chave de acesso da identidade do chamador.	`"AKIAIOSFODNN7EXAMPLE"`
`requestContext.authorizer.iam.accountId`	O ID da Conta da AWS da identidade do chamador.	`"111122223333"`
`requestContext.authorizer.iam.callerId`	O ID (ID do usuário) do chamador.	`"AIDACKCEVSQ6C2EXAMPLE"`
`requestContext.authorizer.iam.cognitoIdentity`	URLs de função não usam esse parâmetro. O Lambda define isso como `null` ou exclui isso do JSON.	`null`
`requestContext.authorizer.iam.principalOrgId`	O ID da organização da entidade principal associado à identidade do chamador.	`"AIDACKCEVSQORGEXAMPLE"`
`requestContext.authorizer.iam.userArn`	O nome do recurso da Amazon (ARN) do usuário da identidade do chamador.	`"arn:aws:iam::111122223333:user/example-user"`
`requestContext.authorizer.iam.userId`	O ID de usuário da identidade do chamador.	`"AIDACOSFODNN7EXAMPLE2"`

Parâmetro	Descrição	Exemplo
`requestContext.domainName`	O nome do domínio do URL de função.	`"<url-id>.lambda-url.us-west-2.on.aws"`
`requestContext.domainPrefix`	O prefixo do domínio do URL de função.	`"<url-id>"`
`requestContext.http`	Um objeto que contém detalhes sobre a solicitação HTTP.	
`requestContext.http.method`	O método HTTP usado na solicitação. Os valores válidos incluem `GET`, `POST`, `PUT`, `HEAD`, `OPTIONS`, `PATCH` e `DELETE`.	`GET`
`requestContext.http.path`	O caminho da solicitação. Por exemplo, se o URL de solicitação for `https://{url-id}.lambda-url.{region}.on.aws/example/test/demo`, o valor do caminho será `/example/test/demo`.	`/example/test/demo`
`requestContext.http.protocol`	O protocolo da solicitação.	`HTTP/1.1`
`requestContext.http.sourceIp`	O endereço IP de origem da conexão TCP imediata que está fazendo a solicitação.	`123.123.123.123`

Parâmetro	Descrição	Exemplo
`requestContext.http.userAgent`	O valor do cabeçalho da solicitação User-Agent.	`Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) Gecko/20100101 Firefox/42.0`
`requestContext.requestId`	O ID da solicitação da invocação. Você pode usar esse ID para logs de invocações relacionadas à função.	`e1506fd5-9e7b-434f-bd42-4f8fa224b599`
`requestContext.routeKey`	URLs de função não usam esse parâmetro. O Lambda define isso como `$default` para marcar um espaço reservado.	`$default`
`requestContext.stage`	URLs de função não usam esse parâmetro. O Lambda define isso como `$default` para marcar um espaço reservado.	`$default`
`requestContext.time`	O timestamp da solicitação.	`"07/Sep/2021:22:50:22 +0000"`
`requestContext.timeEpoch`	O timestamp da solicitação, no horário da era UNIX.	`"1631055022677"`
`body`	O corpo da solicitação. Se o tipo de conteúdo da solicitação for binário, o corpo será codificado na base 64.	`{"key1": "value1", "key2": "value2"}`

Parâmetro	Descrição	Exemplo
`pathParameters`	URLs de função não usam esse parâmetro. O Lambda define isso como `null` ou exclui isso do JSON.	`null`
`isBase64Encoded`	`TRUE` se o corpo for uma carga útil binária e for codificado na base 64. `FALSE` nos demais casos.	`FALSE`
`stageVariables`	URLs de função não usam esse parâmetro. O Lambda define isso como `null` ou exclui isso do JSON.	`null`

Formato de carga útil de resposta

Quando a função retorna uma resposta, o Lambda analisa a resposta e converte-a em uma resposta HTTP. As cargas úteis de resposta de função têm o seguinte formato:

```
{
   "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
}
```

O Lambda infere o formato de resposta para você. Se a função do Lambda retornar JSON válido e não retornar um `statusCode`, o Lambda pressupõe o seguinte:

- `statusCode` é `200`.
- `content-type` é `application/json`.
- `body` é a resposta da função.
- `isBase64Encoded` é `false`.

Os exemplos a seguir mostram como a saída da função do Lambda é mapeada para a carga útil da resposta e como a carga útil da resposta é mapeada para a resposta HTTP final. Quando o cliente invoca o URL de função, ele vê a resposta HTTP.

Exemplo de saída para uma resposta de string

Saída da função do Lambda	Saída de resposta interpretada	Resposta HTTP (o que o cliente vê)

`"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!"`

Exemplo de saída para uma resposta JSON

Saída da função do Lambda	Saída de resposta interpretada	Resposta HTTP (o que o cliente vê)

`{ "message": "Hello, world!" }`	`{ "statusCode": 200, "body": { "message": "Hello, world!" }, "headers": {`	`HTTP/2 200`
		`date: Wed, 08 Sep 2021 18:02:24 GMT`
		`content-type: application/json`
		`content-length: 34`

Saída da função do Lambda	Saída de resposta interpret ada	Resposta HTTP (o que o cliente vê)
	<pre>"content-type": "application/json" }, "isBase64Encoded": false }</pre>	<pre>{ "message": "Hello, world!" }</pre>

Exemplo de saída para uma resposta personalizada

Saída da função do Lambda	Saída de resposta interpret ada	Resposta HTTP (o que o cliente vê)

<pre>{ "statusCode": 201, "headers": { "Content-Type": "application/json", "My-Custom-Header": "Custom Value" }, "body": JSON.stringify({ "message": "Hello, world!" }), "isBase64Encoded": false }</pre>	<pre>{ "statusCode": 201, "headers": { "Content-Type": "application/json", "My-Custom-Header": "Custom Value" }, "body": JSON.stringify({ "message": "Hello, world!" }), "isBase64Encoded": false }</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
		
		{ "message": "Hello, world!" }</pre>

Cookies

Para retornar cookies da função, não adicione manualmente cabeçalhos `set-cookie`. Em vez disso, inclua os cookies no objeto de carga útil da resposta. O Lambda interpreta isso automaticamente e adiciona-os como cabeçalhos `set-cookie` na resposta HTTP, como no exemplo a seguir.

Exemplo de saída para uma resposta que retorna cookies

Saída da função do Lambda	Resposta HTTP (o que o cliente vê)

```
{
   "statusCode": 201,
    "headers": {
        "Content-Type": "application/
json",
        "My-Custom-Header": "Custom
 Value"
    },
    "body": JSON.stringify({
        "message": "Hello, world!"
    }),
    "cookies": [
        "Cookie_1=Value1; Expires=21
 Oct 2021 07:48 GMT",
        "Cookie_2=Value2; Max-Age=7
8000"
    ],
    "isBase64Encoded": false
}
```

```
HTTP/2 201
date: Wed, 08 Sep 2021 18:02:24 GMT
content-type: application/json
content-length: 27
my-custom-header: Custom Value
set-cookie: Cookie_1=Value2;
 Expires=21 Oct 2021 07:48 GMT
set-cookie: Cookie_2=Value2; Max-
Age=78000

{
  "message": "Hello, world!"
}
```

Monitorar URLs de função do Lambda

Você pode usar o AWS CloudTrail e o Amazon CloudWatch para monitorar os URLs de função.

Tópicos

Monitorar URLs de função com o CloudTrail

Para URLs de função, o Lambda é automaticamente compatível com o registro das seguintes operações de API como eventos nos arquivos de log do CloudTrail:

- CreateFunctionUrlConfig
- UpdateFunctionUrlConfig
- DeleteFunctionUrlConfig
- GetFunctionUrlConfig
- ListFunctionUrlConfigs

Toda entrada de log contém informações sobre a identidade do chamador, quando a solicitação foi feita e outros detalhes. Você pode ver todos os eventos nos últimos 90 dias visualizando o Event history (Histórico de eventos) do CloudTrail. Para reter registros por mais de 90 dias, você pode criar uma trilha.

Por padrão, o CloudTrail não registra solicitações `InvokeFunctionUrl`, que são consideradas eventos de dados. Porém, você pode ativar o registro de eventos de dados no CloudTrail. Para obter mais informações, consulte Registro eventos de dados em logs para trilhas no Guia do usuário do AWS CloudTrail.

Métricas do CloudWatch para URLs de função

O Lambda envia métricas agregadas sobre solicitações de URL de função para o CloudWatch. Com essas métricas, você pode monitorar os URLs de função, criar painéis e configurar alarmes no console do CloudWatch.

Os URLs de função são compatíveis as seguintes métricas de invocação. Recomendamos visualizar essas métricas com a estatística `Sum`.

- `UrlRequestCount`: o número de solicitações feitas a esse URL de função.
- `Url4xxCount`: o número de solicitações que retornaram um código de status HTTP 4XX. Os códigos da série 4XX indicam erros do lado do cliente, como solicitações incorretas.
- `Url5xxCount`: o número de solicitações que retornaram um código de status HTTP 5XX. Os códigos da série 5XX indicam erros do lado do servidor, como erros de função e de tempo limite.

Os URLs de função também são compatíveis com a métrica de performance a seguir. Recomendamos visualizar essas métricas com as estatísticas `Average` ou `Max`.

- `UrlRequestLatency`: o tempo decorrido entre a hora que o URL de função recebe uma solicitação e a hora que o URL de função retorna uma resposta.

Cada uma dessas métricas de invocação e performance é compatível com as seguintes dimensões:

- `FunctionName`: visualizar as métricas agregadas para os URLs de função atribuídos a uma versão da função não publicada `$LATEST` ou para qualquer um dos aliases da função. Por exemplo, `hello-world-function`.
- `Resource`: visualizar as métricas para um URL de função específico. Isso é definido por um nome de função, junto com a versão da função não publicada `$LATEST` ou um dos aliases da função. Por exemplo, `hello-world-function:$LATEST`.
- `ExecutedVersion`: visualizar as métricas para um URL de função específico com base na versão executada. Você pode usar essa dimensão principalmente para rastrear o URL de função atribuído à versão não publicada `$LATEST`.

Tutorial: criar uma função do Lambda com um URL de função

Neste tutorial, você criará uma função do Lambda definida como um arquivo .zip com um endpoint de URL de função público que retorna o produto de dois números. Para obter mais informações sobre a configuração de URLs de função, consulte Criar e gerenciar URLs de função.

Pré-requisitos

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em Criar uma função do Lambda com o console para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, a AWS Command Line Interface (AWS CLI) versão 2 será necessária. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

A seguinte saída deverá ser mostrada:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell.

> **Note**
>
> No Windows, alguns comandos da CLI do Bash que você costuma usar com o Lambda (como `zip`) não são compatíveis com os terminais integrados do sistema operacional. Para obter uma versão do Ubuntu com o Bash integrada no Windows, instale o Subsistema do Windows para Linux. Os exemplos de comandos da CLI neste guia usam a formatação Linux. Os comandos que incluem documentos JSON em linha deverão ser reformatados se você estiver usando a CLI do Windows.

Criar uma função de execução

Crie a função de execução que dá à sua função do Lambda permissão para acessar recursos da AWS.

Para criar uma função de execução

1. Abra a página Roles (Funções) no console do AWS Identity and Access Management (IAM).
2. Selecione Criar função.
3. Em Tipo de entidade confiável, selecione serviço da AWS e, em Caso de uso, selecione Lambda.
4. Escolha Próximo.
5. No painel Políticas de permissões, insira **`AWSLambdaBasicExecutionRole`** na caixa de pesquisa.
6. Marque a caixa de seleção ao lado da política gerenciada pela AWS `AWSLambdaBasicExecutionRole` e escolha Avançar.
7. Insira **`lambda-url-role`** em Nome do perfil e depois escolha Criar perfil.

A política `AWSLambdaBasicExecutionRole` tem as permissões de que a função precisa para gravar logs no Amazon CloudWatch Logs. Mais adiante neste tutorial, você precisará do nome do recurso da Amazon (ARN) do perfil para criar a função do Lambda.

Para localizar o ARN do seu perfil de execução

1. Abra a página Roles (Funções) no console do AWS Identity and Access Management (IAM).
2. Escolha o perfil que você acabou de criar (`lambda-url-role`).
3. No painel Resumo, copie o ARN.

Criar uma função do Lambda com um URL de função (arquivo .zip)

Crie uma função do Lambda com um endpoint de URL de função usando um arquivo .zip.

Para criar a função

1. Copie o exemplo de código a seguir em um arquivo com o nome `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. Crie um pacote de implantação.

```
zip function.zip index.js
```

3. Crie uma função do Lambda com o comando `create-function`. Certifique-se de substituir o ARN do perfil pelo ARN do seu próprio perfil de execução que você copiou anteriormente no 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. Adicione uma política baseada em recurso à função concedendo permissões para acesso público ao URL da função.

```
aws lambda add-permission \
    --function-name my-url-function \
    --action lambda:InvokeFunctionUrl \
    --principal "*" \
    --function-url-auth-type "NONE" \
    --statement-id url
```

5. Crie um endpoint de URL para a função com o comando `create-function-url-config`.

```
aws lambda create-function-url-config \
    --function-name my-url-function \
    --auth-type NONE
```

Testar o endpoint de URL de função

Invoque a função do Lambda chamando o endpoint do URL de função usando um cliente HTTP, como curl ou Postman.

```
curl 'https://abcdefg.lambda-url.us-east-1.on.aws/' \
-H 'Content-Type: application/json' \
-d '{"num1": "10", "num2": "10"}'
```

A seguinte saída deverá ser mostrada:

```
The product of 10 and 10 is 100
```

Criar uma função do Lambda com um URL de função (CloudFormation)

Você também pode criar uma função do Lambda com um endpoint de URL de função usando o tipo de 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
```

Criar uma função do Lambda com um URL de função (AWS SAM)

Você também pode criar uma função do Lambda configurada com um endpoint de URL de função usando o 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
```

Limpe os recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Para excluir a função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Escolha Excluir.
4. Insira o nome do perfil no campo de entrada de texto e escolha Delete (Excluir).

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.

2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Gerenciar funções do AWS Lambda

Aprenda a ajustar e proteger os recursos associados à função do Lambda usando a API ou o console do Lambda.

Uso do Lambda com a AWS CLI

Você pode usar a AWS Command Line Interface para gerenciar funções e outros recursos do AWS Lambda. A AWS CLI usa o AWS SDK for Python (Boto) para interagir com a API do Lambda. Neste tutorial, você gerencia e invoca as funções do Lambda com a AWS CLI.

Escalabilidade de funções

É possível configurar dois controles de simultaneidade em nível de função: simultaneidade reservada e simultaneidade provisionada. A simultaneidade corresponde ao número de instâncias da função que estão ativas e podem ser configuradas para garantir que as funções fundamentais evitem o controle de utilização.

Assinatura de código

A assinatura de código do Lambda fornece controles de confiança e integridade que permitem verificar se apenas o código inalterado publicado pelos desenvolvedores aprovados foi implantado em suas funções do Lambda.

Organizar com etiquetas

É possível marcar as funções do Lambda para ativar o controle de acesso por atributo (ABAC) e organizá-las por proprietário, projeto ou departamento.

Uso de camadas

É possível aplicar camadas criadas anteriormente para reduzir o tamanho do pacote de implantação e promover o compartilhamento de código e a separação de responsabilidades para que você possa iterar mais rapidamente ao gravar a lógica de negócios.

Como usar o Lambda com o AWS CLI

Você pode usar a AWS Command Line Interface para gerenciar funções e outros recursos do AWS Lambda. A AWS CLI usa o AWS SDK for Python (Boto) para interagir com a API do Lambda. Você pode usá-la para aprender sobre a API e aplicar esse conhecimento na criação de aplicações que usam o Lambda com o AWS SDK.

Neste tutorial, você gerencia e invoca as funções do Lambda com a AWS CLI. Para obter mais informações, consulte O que é o AWS CLI? no Manual do usuário do AWS Command Line Interface.

Pré-requisitos

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Se ainda não tiver visto as instruções, acesse-as em the section called “Criar uma função do Lambda com o console”.

Para concluir as etapas a seguir, a AWS Command Line Interface (AWS CLI) versão 2 será necessária. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

A seguinte saída deverá ser mostrada:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell.

> **Note**
>
> No Windows, alguns comandos da CLI do Bash que você costuma usar com o Lambda (como `zip`) não são compatíveis com os terminais integrados do sistema operacional. Para obter uma versão do Ubuntu com o Bash integrada no Windows, instale o Subsistema do Windows para Linux. Os exemplos de comandos da CLI neste guia usam a formatação Linux. Os comandos que incluem documentos JSON em linha deverão ser reformatados se você estiver usando a CLI do Windows.

Criar a função de execução

Crie a função de execução que dá à sua função permissão para acessar recursos do AWS. Para criar uma função de execução com a AWS CLI, use o comando `create-role`.

No exemplo a seguir, você especifica a política de confiança em linha. Os requisitos para escapar de aspas na string JSON variam dependendo do seu 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"}]}'
```

Também é possível definir a política de confiança para a função usando um arquivo JSON. No exemplo a seguir, `trust-policy.json` é um arquivo no diretório atual. Essa política de confiança permite que o Lambda use as permissões da função fornecendo à entidade principal de serviço a permissão `lambda.amazonaws.com` para chamar a ação 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
```

A seguinte saída deverá ser mostrada:

```
{
```

```
    "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"
                }
            ]
        }
    }
}
```

Para adicionar permissões à função, use o comando attach-policy-to-role. Inicie adicionando o`AWSLambdaBasicExecutionRole`política gerenciada pela.

```
aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole
```

A AWSLambdaBasicExecutionRolepolítica tem as permissões que a função precisa para gravar registros em CloudWatch Logs.

Criar a função

O exemplo a seguir registra em log os valores das variáveis de ambiente e o objeto do 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
}
```

Para criar a função

1. Copie o código de amostra em um arquivo chamado `index.js`.
2. Crie um pacote de implantação.

```
zip function.zip index.js
```

3. Crie uma função do Lambda com o comando `create-function`. Substitua o texto realçado no ARN da função pelo ID da conta.

```
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
```

A seguinte saída deverá ser mostrada:

```
{
    "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",
    ...
}
```

Para obter logs para uma invocação a partir da linha de comando, use a opção `--log-type`. A resposta inclui um campo `LogResult` que contém até 4 KB de logs codificados em base64 da invocação.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

A seguinte saída deverá ser mostrada:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Você pode usar o utilitário `base64` para decodificar os logs.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text |  base64 -d
```

A seguinte saída deverá ser mostrada:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e Ubuntu no Windows. Para macOS, o comando é `base64 -D`.

Para obter eventos de log completos a partir da linha de comando, você pode incluir o nome do fluxo de logs na saída de sua função, conforme mostrado no exemplo anterior. O script de exemplo a seguir invoca uma função chamada `my-function` e faz o download dos últimos 5 eventos de log.

Example Script get-logs.sh

Este exemplo requer que `my-function` retorne um ID de fluxo de 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
```

O script usa `sed` para remover aspas do arquivo de saída e dorme por 15 segundos para permitir que os logs estejam disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

```
./get-logs.sh
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
}
```

Atualizar a função

Depois de criar uma função, você pode configurar recursos adicionais para a função, como acionadores, acesso à rede e acesso ao sistema de arquivos. Você também pode ajustar recursos associados à função, como memória e simultaneidade. Essas configurações se aplicam a funções definidas como arquivos.zip e a funções definidas como imagens de contêiner.

Use o update-function-configurationcomando para configurar funções. O exemplo a seguir define a memória de função como 256 MB.

Example update-function-configuration comando

```
aws lambda update-function-configuration \
--function-name my-function \
--memory-size 256
```

Listar as funções do Lambda na conta

Execute o comando da AWS CLI `list-functions` a seguir para recuperar uma lista de funções que você criou.

```
aws lambda list-functions --max-items 10
```

A seguinte saída deverá ser mostrada:

```
{
    "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="
}
```

Em resposta, o Lambda retorna uma lista de até 10 funções. Se há mais funções que você pode recuperar, o parâmetro `NextToken` fornece um marcador que você pode usar na próxima solicitação `list-functions`. O comando `list-functions` da AWS CLI a seguir é um exemplo que mostra o parâmetro `--starting-token`.

```
aws lambda list-functions --max-items 10 --starting-
token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxMH0=
```

Recuperar uma função do Lambda

O comando `get-function` da CLI do Lambda retorna metadados da função do Lambda e um URL pré-assinado que você pode usar para fazer download do pacote de implantação da função.

```
aws lambda get-function --function-name my-function
```

A seguinte saída deverá ser mostrada:

```
{
    "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-..."
    }
}
```

Para obter mais informações, consulte GetFunction.

Limpeza

Execute o comando `delete-function` a seguir para excluir a função `my-function`.

```
aws lambda delete-function --function-name my-function
```

Exclua a função do IAM que você criou no console do IAM. Para obter mais informações sobre como excluir uma função, consulte Excluir funções ou perfis de instância no Manual do usuário do IAM.

Como entender a escalabilidade da função do Lambda

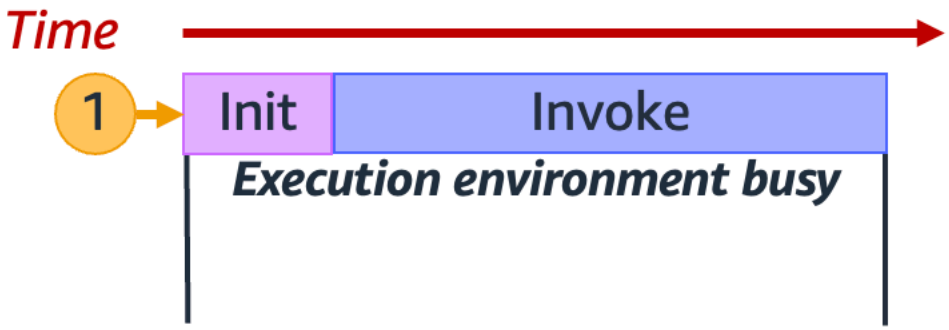
Simultaneidade corresponde ao número de solicitações em andamento que a função do AWS Lambda está processando ao mesmo tempo. Para cada solicitação simultânea, o Lambda provisiona uma instância separada do seu ambiente de execução. À medida que suas funções recebem mais solicitações, o Lambda gerencia a escalabilidade do número de ambientes de execução de forma automática até atingir o limite de simultaneidade da sua conta. Por padrão, o Lambda fornece à sua conta um limite total de simultaneidade de mil execuções simultâneas para todas as funções em uma Região da AWS. Para atender às necessidades específicas da sua conta, é possível solicitar um aumento de cotas e configurar controles de simultaneidade em nível de função para que as funções essenciais não sofram controle de utilização.

Este tópico explica conceitos de simultaneidade e escalabilidade de funções no Lambda. Ao final deste tópico, você será capaz de compreender como calcular a simultaneidade, visualizar as duas principais opções de controle de simultaneidade (reservada e provisionada), estimar as configurações apropriadas de controle de simultaneidade e visualizar métricas para otimização adicional.

Seções

Compreender e visualizar a simultaneidade

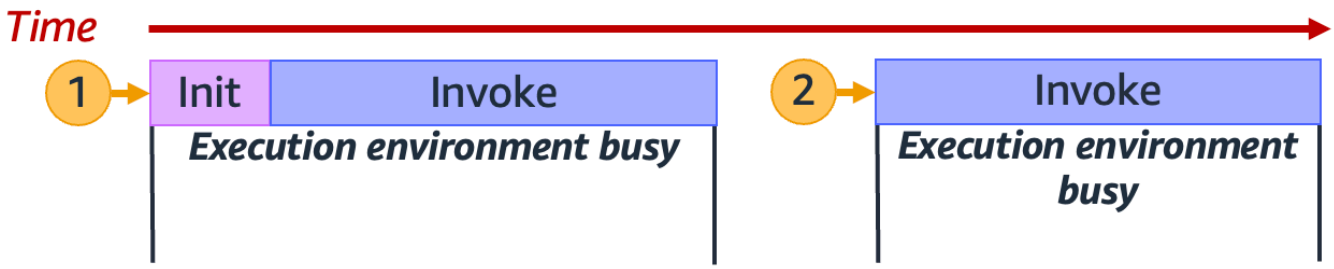
O Lambda invoca sua função em um ambiente de execução seguro e isolado. Para processar uma solicitação, o Lambda deve primeiro inicializar um ambiente de execução (a Init phase [Fase de inicialização]) antes de usá-lo para invocar sua função (a Invoke phase [Fase de invocação]):



> **Note**
>
> As durações reais de inicialização e invocação podem variar dependendo de diversos fatores, como o runtime escolhido e o código da função do Lambda. O diagrama anterior não pretende representar as proporções exatas das durações das fases de inicialização e de invocação.

O diagrama anterior usa um retângulo para representar um único ambiente de execução. Quando sua função recebe a primeira solicitação (representada pelo círculo amarelo com o rótulo 1), o Lambda cria um novo ambiente de execução e executa o código de forma externa ao manipulador principal durante a fase de inicialização. Em seguida, o Lambda executa o código do manipulador principal da sua função durante a fase de invocação. Durante todo esse processo, esse ambiente de execução fica ocupado e não consegue processar outras solicitações.

Quando o Lambda terminar de processar a primeira solicitação, o ambiente de execução poderá processar solicitações adicionais para a mesma função. Para solicitações subsequentes, o Lambda não precisará reinicializar o ambiente.

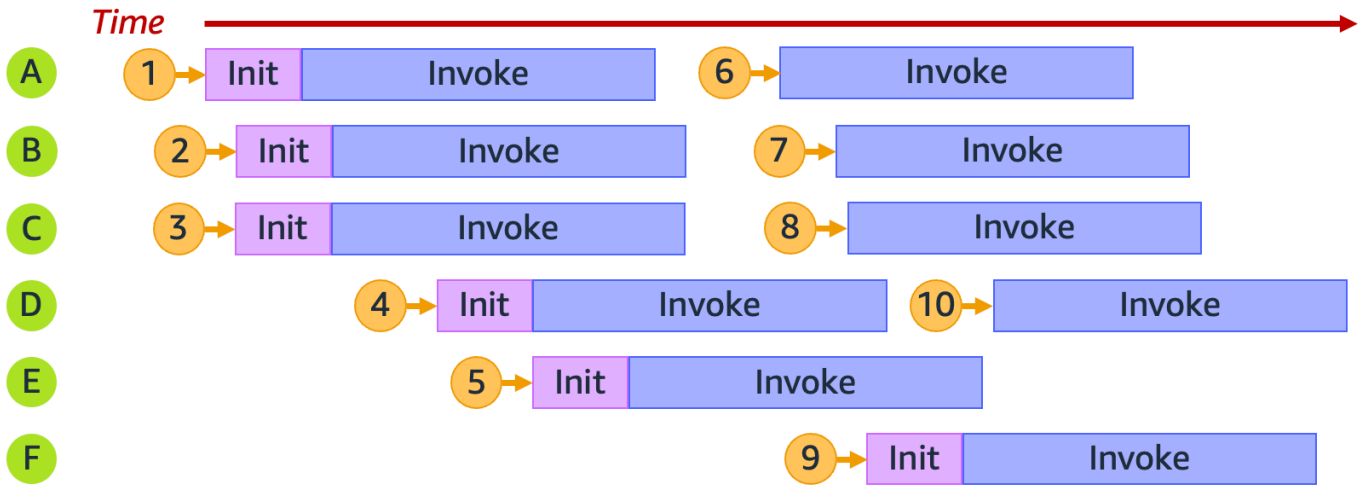


No diagrama anterior, o Lambda reutiliza o ambiente de execução para processar a segunda solicitação (representada pelo círculo amarelo com o rótulo 2).

Até o momento, nos concentramos em apenas uma única instância do seu ambiente de execução (ou seja, uma simultaneidade de um). Na prática, o Lambda pode precisar provisionar diversas instâncias do ambiente de execução em paralelo para processar todas as solicitações recebidas. Quando a função receber uma nova solicitação, uma das duas coisas poderá acontecer:

- Se uma instância do ambiente de execução inicializada previamente estiver disponível, o Lambda a usará para processar a solicitação.
- Caso contrário, o Lambda criará uma nova instância do ambiente de execução para processar a solicitação.

Por exemplo, vamos explorar o que acontece quando a função recebe dez solicitações:



No diagrama anterior, cada plano horizontal representa uma única instância do ambiente de execução (rotulada de A a F). Veja como o Lambda processa cada solicitação:

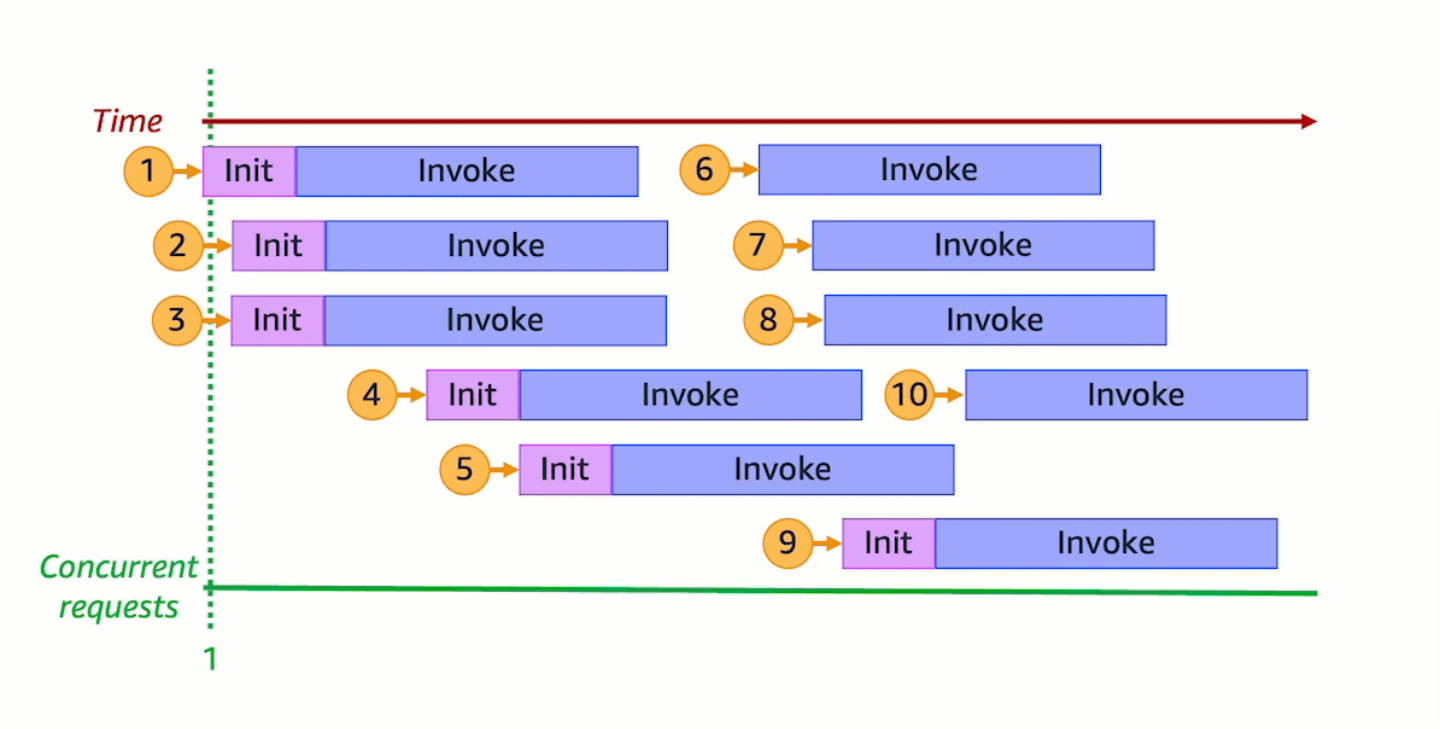
Comportamento do Lambda para solicitações de 1 a 10

Solicitação	Comportamento do Lambda	Reasoning
1	Provisiona novo ambiente A	Esta é a primeira solicitação; não há uma instância do ambiente de execução disponível.

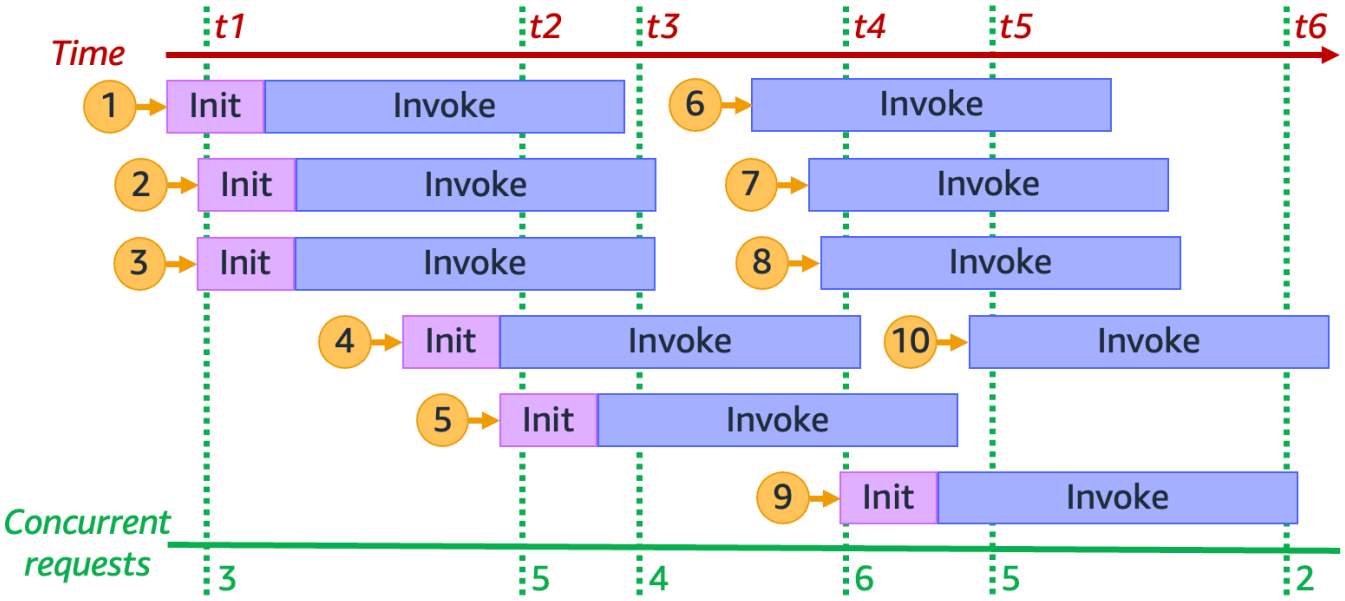
Solicitação	Comportamento do Lambda	Reasoning
2	Provisiona novo ambiente B	A instância A do ambiente de execução existente está ocupada.
3	Provisiona novo ambiente C	As instâncias A e B do ambiente de execução existente estão ambas ocupadas.
4	Provisiona novo ambiente D	As instâncias A, B e C do ambiente de execução existente estão todas ocupadas.
5	Provisiona novo ambiente E	As instâncias A, B, C e D do ambiente de execução existente estão todas ocupadas.
6	Reutiliza o ambiente A	A instância A do ambiente de execução concluiu o processamento da solicitação 1 e está disponível.
7	Reutiliza o ambiente B	A instância B do ambiente de execução concluiu o processamento da solicitação 2 e está disponível.
8	Reutiliza o ambiente C	A instância C do ambiente de execução concluiu o processamento da solicitação 3 e está disponível.

Solicitação	Comportamento do Lambda	Reasoning
9	Provisiona novo ambiente F	As instâncias A, B, C, D e E do ambiente de execução existente estão todas ocupadas.
10	Reutiliza o ambiente D	A instância D do ambiente de execução concluiu o processamento da solicitação 4 e está disponível.

À medida que a função recebe mais solicitações simultâneas, o Lambda aumenta a escala verticalmente do número de instâncias do ambiente de execução em resposta. A animação a seguir monitora o número de solicitações simultâneas ao longo do tempo:



Ao paralisar a animação anterior em seis pontos distintos no tempo, obtemos o diagrama a seguir:



No diagrama anterior, é possível traçar uma linha vertical em qualquer ponto no tempo e contar o número de ambientes que cruzam essa linha. Isso nos fornece o número de solicitações simultâneas naquele ponto no tempo. Por exemplo, no tempo `t1`, existem três ambientes ativos atendendo três solicitações simultâneas. O número máximo de solicitações simultâneas nessa simulação ocorre no tempo `t4`, quando existem seis ambientes ativos atendendo seis solicitações simultâneas.

Para resumir, a simultaneidade da sua função corresponde ao número de solicitações simultâneas que ela está processando ao mesmo tempo. Em resposta a um aumento na simultaneidade da sua função, o Lambda provisiona mais instâncias do ambiente de execução para atender à demanda de solicitações.

Calcular a simultaneidade para uma função

Em geral, a simultaneidade de um sistema corresponde à capacidade de processar mais de uma tarefa simultaneamente. No Lambda, simultaneidade corresponde ao número de solicitações em andamento que sua função está processando ao mesmo tempo. Uma maneira rápida e prática de medir a simultaneidade de uma função do Lambda é usar a fórmula a seguir:

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

Simultaneidade é diferente de solicitações por segundo. Por exemplo, suponha que a função receba 100 solicitações por segundo em média. Se a duração média da solicitação for de um segundo, é verdade que a simultaneidade também será cem:

```
Concurrency = (100 requests/second) * (1 second/request) = 100
```

No entanto, se a duração média da solicitação for de 500 ms, a simultaneidade será 50:

```
Concurrency = (100 requests/second) * (0.5 second/request) = 50
```

Na prática, o que significa uma simultaneidade de 50? Se a duração média da solicitação for de 500 ms, você pode pensar em uma instância da função que é capaz de processar duas solicitações por segundo. Ou seja, são necessárias 50 instâncias da função para processar uma carga de 100 solicitações por segundo. Uma simultaneidade de 50 significa que o Lambda deve provisionar 50 instâncias de ambiente de execução para processar essa workload com eficiência sem qualquer controle de utilização. Veja como expressar isso em forma de equação:

```
Concurrency = (100 requests/second) / (2 requests/second) = 50
```

Se a função receber o dobro do número de solicitações (200 solicitações por segundo), mas requerer apenas metade do tempo para processar cada uma (250 ms), a simultaneidade ainda será 50:

```
Concurrency = (200 requests/second) * (0.25 second/request) = 50
```

Teste sua compreensão sobre simultaneidade

Suponha que você tenha uma função que demora, em média, 200 ms para ser executada. Durante o pico de carga, você observa 5 mil solicitações por segundo. Qual será a simultaneidade da sua função durante o pico de carga?

Resposta

A duração média da função é de 200 ms, ou 0,2 segundo. Usando a fórmula da simultaneidade, é possível inserir os números para obter uma simultaneidade de mil:

```
Concurrency = (5,000 requests/second) * (0.2 seconds/request) = 1,000
```

Como alternativa, uma duração média da função de 200 ms significa que sua função pode processar 5 solicitações por segundo. Para lidar com a workload de 5 mil solicitações por segundo, você precisará de 1 mil instâncias do ambiente de execução. Portanto, a simultaneidade será 1 mil:

```
Concurrency = (5,000 requests/second) / (5 requests/second) = 1,000
```

Diferenciar entre simultaneidade e solicitações por segundo

Conforme mencionado na seção anterior, simultaneidade é diferente de solicitações por segundo. Essa é uma distinção especialmente importante a ser feita no trabalho com funções que têm uma duração média de solicitação inferior a 100 ms.

Em geral, cada instância do ambiente de execução pode processar no máximo dez solicitações por segundo. Esse limite se aplica às funções síncronas sob demanda, bem como às funções que usam simultaneidade provisionada. Se você não conhece esse limite, é possível que não saiba porque as funções podem apresentar controle de utilização em determinados cenários.

Por exemplo, considere uma função com uma duração média de solicitação de 50 ms. Com 200 solicitações por segundo, veja a simultaneidade dessa função:

```
Concurrency = (200 requests/second) * (0.05 second/request) = 10
```

Com base nesse resultado, você pode achar que precisa de apenas dez instâncias do ambiente de execução para processar essa carga. No entanto, cada ambiente de execução pode processar apenas dez execuções por segundo. Isso significa que, com dez ambientes de execução, você só pode lidar com cem solicitações por segundo do total de 200 solicitações. Essa função sofre controle de utilização.

A lição é que você precisa considerar a simultaneidade e as solicitações por segundo ao definir as configurações de simultaneidade para as funções. Nesse caso, você precisa de 20 ambientes de execução para a função, mesmo que ela tenha apenas uma simultaneidade de dez.

Teste sua compreensão da simultaneidade (funções abaixo de 100 ms)

Suponha que você tenha uma função que demora, em média, 20 ms para ser executada. Durante o pico de carga, você observa 3 mil solicitações por segundo. Qual será a simultaneidade da sua função durante o pico de carga?

Resposta

A duração média da função é de 20 ms ou 0,02 segundo. Usando a fórmula da simultaneidade, é possível inserir os números para obter uma simultaneidade de 60:

```
Concurrency = (3,000 requests/second) * (0.02 seconds/request) = 60
```

No entanto, cada ambiente de execução pode processar apenas dez solicitações por segundo. Com 60 ambientes de execução, a função pode processar no máximo 600 solicitações por segundo. Para acomodar totalmente as três mil solicitações, você precisará de pelo menos 300 instâncias do ambiente de execução.

Compreender a simultaneidade reservada e a simultaneidade provisionada

Por padrão, sua conta tem um limite de simultaneidade de mil execuções simultâneas para todas as funções em uma região. Suas funções compartilham esse grupo de mil para simultaneidade sob demanda. Suas funções poderão apresentar controle de utilização (ou seja, começarem a descartar solicitações) se a simultaneidade disponível for esgotada.

Algumas de suas funções podem ser mais essenciais do que outras. Como resultado, talvez seja necessário definir as configurações de simultaneidade para garantir que as funções essenciais obtenham a simultaneidade de que precisam. Existem dois tipos de controle de simultaneidade disponíveis: simultaneidade reservada e simultaneidade provisionada.

- Use a simultaneidade reservada para reservar uma parte da simultaneidade de sua conta para uma função. Isso é útil se você não deseja que outras funções ocupem toda a simultaneidade não reservada disponível.
- Use a simultaneidade provisionada para inicializar previamente diversas instâncias de ambiente para uma função. Isso é útil para reduzir as latências de inicialização a frio.

Simultaneidade reservada

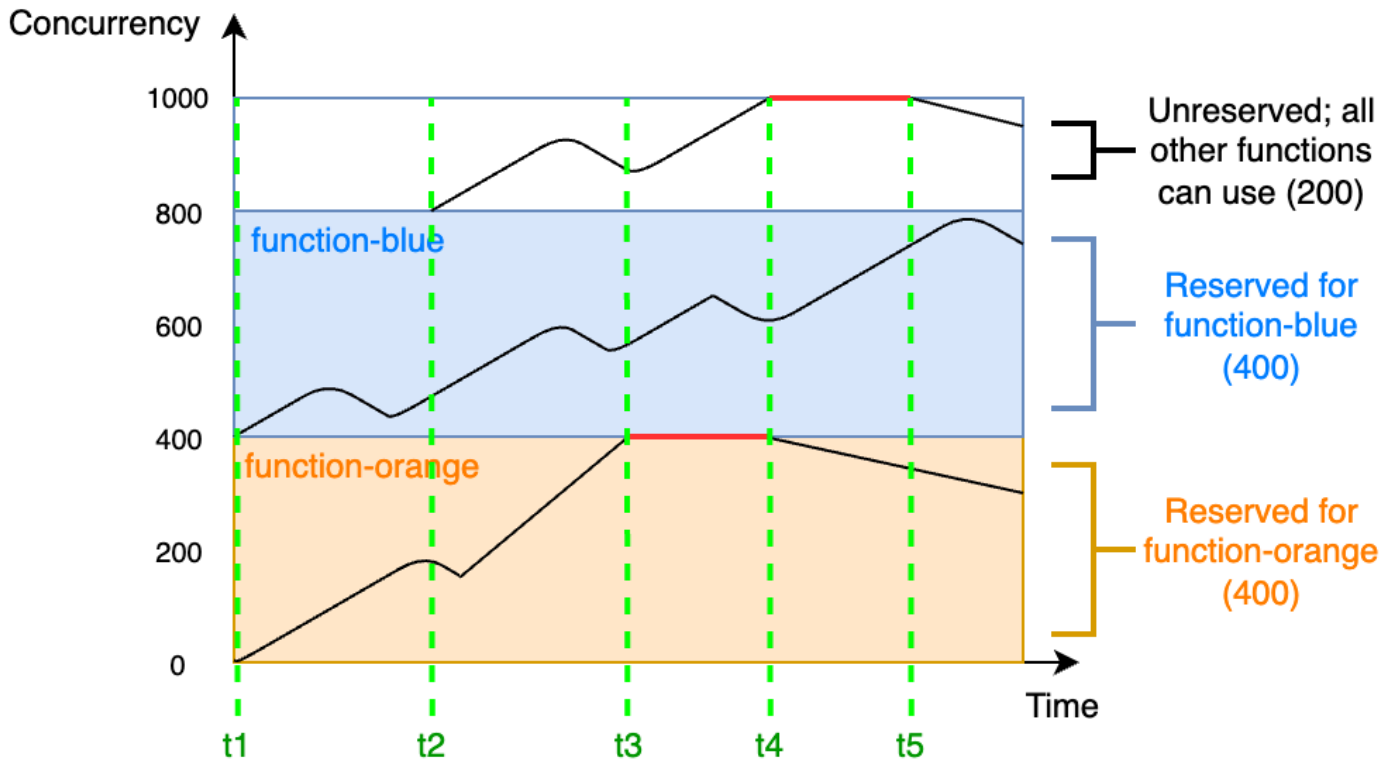
Se você deseja garantir que uma determinada quantidade de simultaneidade esteja disponível para sua função a qualquer momento, use a simultaneidade reservada.

A simultaneidade reservada corresponde ao número máximo de instâncias simultâneas que você deseja alocar para a função. Quando você dedica uma simultaneidade reservada para uma função, nenhuma outra função pode usar essa simultaneidade. Em outras palavras, definir a simultaneidade

reservada pode afetar o grupo de simultaneidade disponível para outras funções. As funções que não têm simultaneidade reservada compartilham o grupo restante de simultaneidade não reservada.

A configuração da simultaneidade reservada é contabilizada para o limite geral de simultaneidade da sua conta. Não há cobrança para configurar a simultaneidade reservada para uma função.

Para compreender melhor a simultaneidade reservada, considere o diagrama a seguir:



Neste diagrama, o limite de simultaneidade da sua conta para todas as funções nesta região está no limite padrão de mil. Suponha que você tenha duas funções essenciais, `function-blue` e `function-orange`, que rotineiramente espera obter altos volumes de invocação. Você decide oferecer 400 unidades de simultaneidade reservada para `function-blue` e 400 unidades de simultaneidade reservada para `function-orange`. Neste exemplo, todas as outras funções em sua conta devem compartilhar as 200 unidades restantes de simultaneidade não reservada.

O diagrama tem cinco pontos de interesse:

- Em `t1`, tanto `function-orange` quanto `function-blue` começam a receber solicitações. Cada função começa a usar sua parte alocada das unidades de simultaneidade reservadas.

- Em `t2`, `function-orange` e `function-blue`, elas estão recebendo constantemente mais solicitações. Ao mesmo tempo, você implanta algumas outras funções do Lambda, que começam a receber solicitações. Você não precisa alocar simultaneidade reservada para as outras funções. Elas começam a usar as 200 unidades restantes de simultaneidade não reservada.
- Em `t3`, `function-orange` atinge a simultaneidade máxima de 400. Embora haja simultaneidade não utilizada em outras partes da sua conta, `function-orange` não consegue acessá-la. A linha vermelha indica que `function-orange` está passando por um controle de utilização e o Lambda pode descartar solicitações.
- Em `t4`, `function-orange` começa a receber menos solicitações e não está mais sob o controle de utilização. No entanto, suas outras funções experimentam um aumento no tráfego e começam a sofrer o controle de utilização. Embora haja simultaneidade não utilizada em outras partes da sua conta, essas outras funções não conseguem acessá-la. A linha vermelha indica que suas outras funções estão passando pelo controle de utilização.
- Em `t5`, as outras funções começam a receber menos solicitações e não estão mais sob o controle de utilização.

Com base neste exemplo, observe que a reserva de simultaneidade tem os efeitos a seguir:

- Sua função pode ser escalada independentemente de outras funções em sua conta. Todas as funções da sua conta na mesma região que não têm simultaneidade reservada compartilham o grupo de simultaneidade não reservada. Sem a simultaneidade reservada, as outras funções podem potencialmente usar toda a simultaneidade disponível. Isso evita que funções essenciais aumentem a escala verticalmente, se necessário.
- Sua função não pode ser escalada para além do controle. A simultaneidade reservada limita a simultaneidade máxima da sua função. Isso significa que a função não pode usar a simultaneidade reservada para outras funções ou usar a simultaneidade do grupo não reservado. É possível reservar a simultaneidade para evitar que a função use toda a simultaneidade disponível em sua conta ou sobrecarregue os recursos downstream.
- Talvez você não consiga usar toda a simultaneidade disponível em sua conta. A reserva de simultaneidade é contabilizada para o limite de simultaneidade da sua conta, mas isso também significa que outras funções não podem usar essa parte da simultaneidade reservada. Se a função não usar toda a simultaneidade reservada para ela, você estará efetivamente desperdiçando essa simultaneidade. Isso não é um problema, caso outras funções em sua conta possam se beneficiar da simultaneidade desperdiçada.

Para saber como gerenciar as configurações de simultaneidade reservada das funções, consulte Configurar a simultaneidade reservada para uma função.

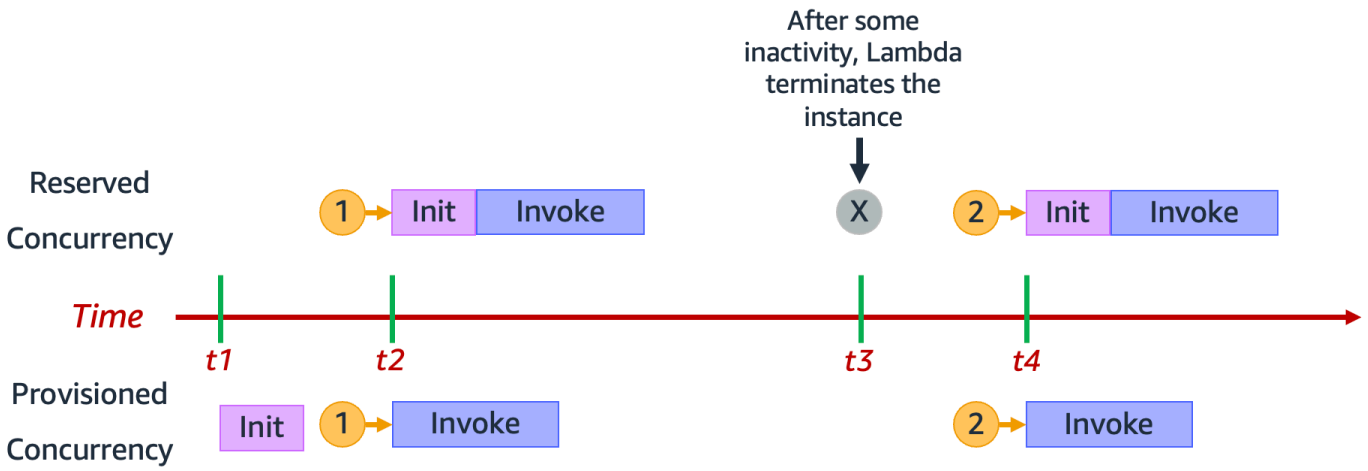
Simultaneidade provisionada

Você usa a simultaneidade reservada para definir o número máximo de ambientes de execução reservados para uma função do Lambda. No entanto, nenhum desses ambientes está inicializado previamente. Como resultado, as invocações da sua função podem demorar mais porque o Lambda deve primeiro inicializar o novo ambiente antes de poder usá-lo para invocar a função. Quando o Lambda precisa inicializar um ambiente novo para realizar uma invocação, isso é conhecido como inicialização a frio. Para mitigar as inicializações a frio, é possível usar a simultaneidade provisionada.

A simultaneidade provisionada corresponde ao número de ambientes de execução inicializados previamente que você deseja alocar para a função. Se você definir a simultaneidade provisionada em uma função, o Lambda inicializará esse número de ambientes de execução para que estejam preparados para responder imediatamente às solicitações da função.

> **Note**
>
> O uso de simultaneidade provisionada gera cobranças na sua conta. Se você estiver trabalhando com runtimes do Java 11 ou do Java 17, também poderá usar o Lambda SnapStart para mitigar problemas de inicialização a frio sem custo adicional. O SnapStart usa snapshots em cache do ambiente de execução para melhorar significativamente o desempenho de inicialização. Você não pode usar o SnapStart e a simultaneidade provisionada na mesma versão da função. Para obter mais informações sobre os recursos, limitações e regiões compatíveis com o SnapStart, consulte Aprimoramento da performance de inicialização com o Lambda SnapStart.

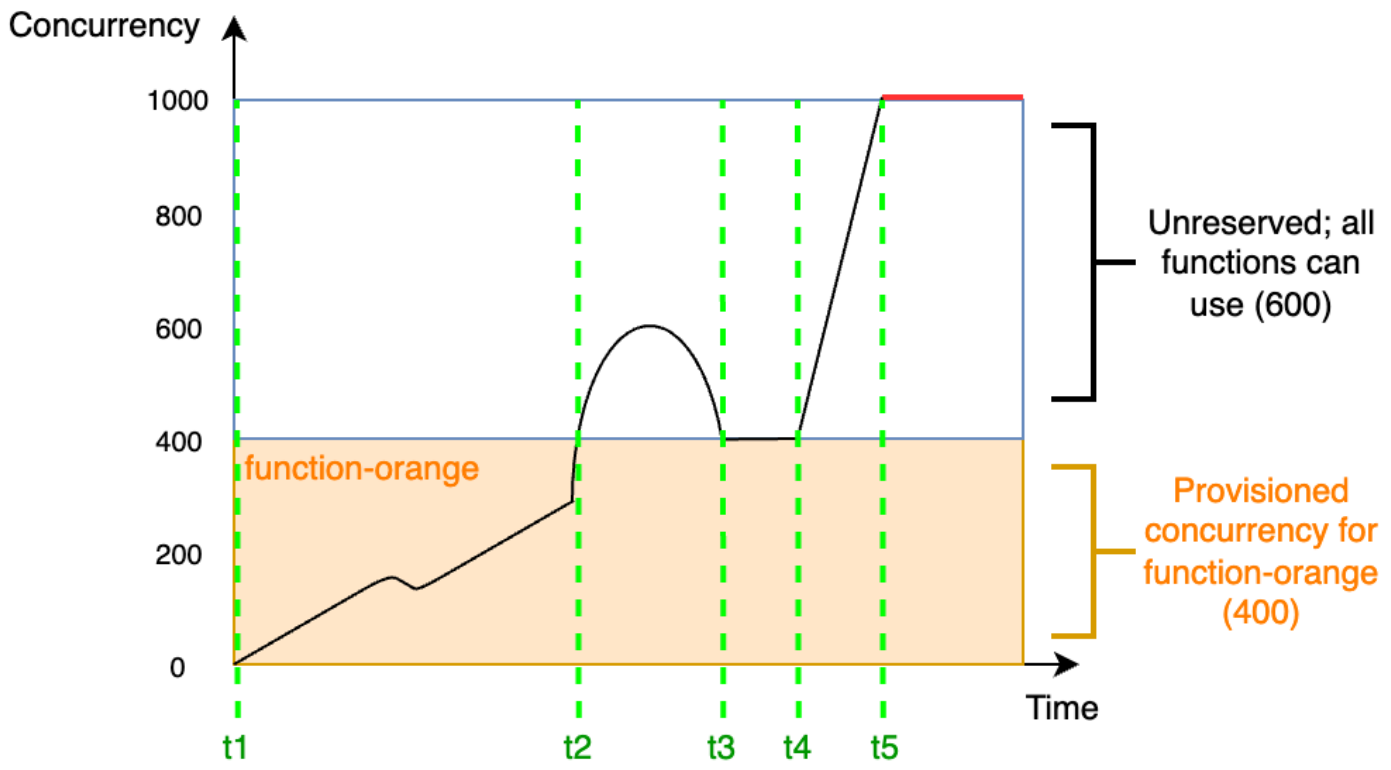
Ao usar a simultaneidade provisionada, o Lambda ainda recicla os ambientes de execução em segundo plano. No entanto, a qualquer momento, o Lambda garantirá que o número de ambientes inicializados previamente seja semelhante ao valor da configuração de simultaneidade provisionada da sua função. Esse comportamento difere da simultaneidade reservada, na qual o Lambda pode encerrar completamente um ambiente após um período de inatividade. O diagrama a seguir ilustra isso comparando o ciclo de vida de um único ambiente de execução quando você configura a função usando a simultaneidade reservada em comparação com a simultaneidade provisionada.



O diagrama tem quatro pontos de interesse:

Tempo	Simultaneidade reservada	Simultaneidade provisionada
t1	Nada acontece.	O Lambda inicializa previamente uma instância do ambiente de execução.
t2	A solicitação 1 é realizada. O Lambda deverá inicializar uma nova instância do ambiente de execução.	A solicitação 1 é realizada. O Lambda usa a instância do ambiente inicializada previamente.
t3	Após algum período de inatividade, o Lambda encerra a instância ativa do ambiente.	Nada acontece.
t4	A solicitação 2 é realizada. O Lambda deverá inicializar uma nova instância do ambiente de execução.	A solicitação 2 é realizada. O Lambda usa a instância do ambiente inicializada previamente.

Para compreender melhor a simultaneidade provisionada, considere o diagrama a seguir:



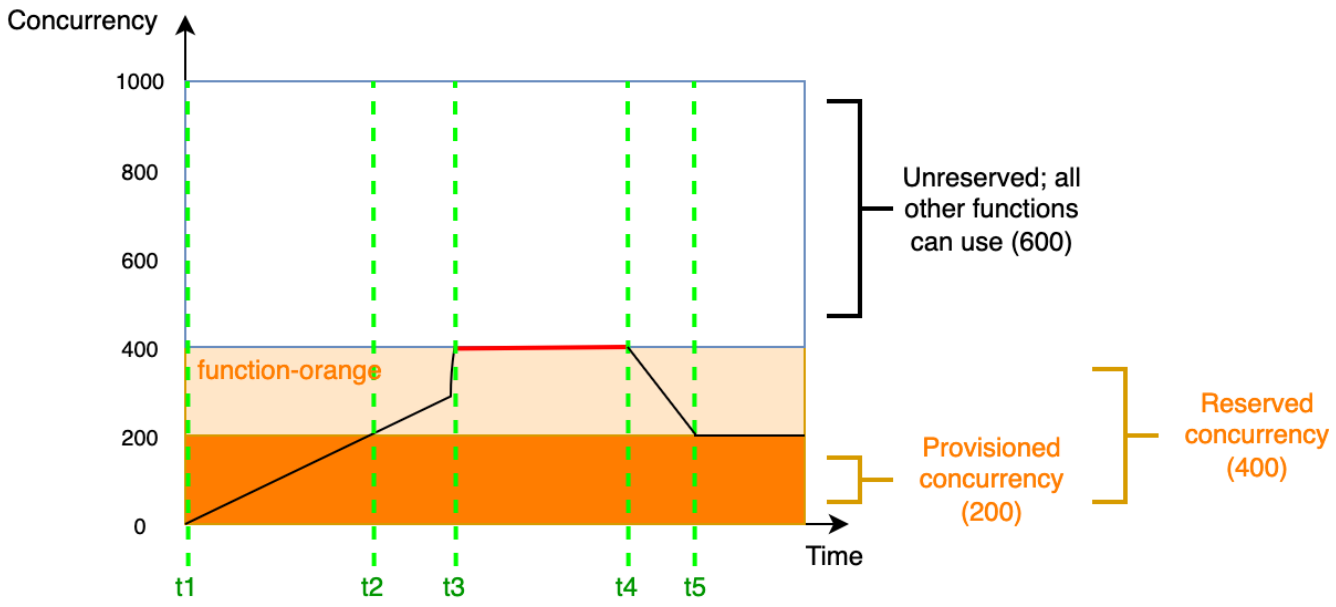
Neste diagrama, você tem um limite de simultaneidade de mil na conta. Você decide fornecer 400 unidades de simultaneidade provisionada para `function-orange`. Todas as funções em sua conta, incluindo `function-orange`, podem usar as 600 unidades restantes de simultaneidade não reservada.

O diagrama tem cinco pontos de interesse:

- Em `t1`, `function-orange` começa a receber solicitações. Como o Lambda inicializou previamente 400 instâncias do ambiente de execução, está tudo pronto para a invocação imediata de `function-orange`.
- Em `t2`, `function-orange` atinge 400 solicitações simultâneas. Como resultado, `function-orange` não é executada com simultaneidade provisionada. No entanto, como ainda há simultaneidade não reservada disponível, o Lambda pode usar isso para lidar com solicitações adicionais da `function-orange` (não há controle de utilização). O Lambda deve criar novas instâncias para atender a essas solicitações e sua função pode apresentar latências de inicialização a frio.

- Em `t3`, `function-orange` retorna para 400 solicitações simultâneas após um breve pico no tráfego. O Lambda é novamente capaz de processar todas as solicitações sem latências de inicialização a frio.
- Em `t4`, as funções da conta sofrem um pico de tráfego. Esse pico pode ser consequência da `function-orange` ou de qualquer outra função em sua conta. O Lambda usa simultaneidade não reservada para processar essas solicitações.
- Em `t5`, as funções em sua conta atingem o limite máximo de simultaneidade de mil e sofrem controle de utilização.

O exemplo anterior considerou apenas a simultaneidade provisionada. Na prática, é possível definir a simultaneidade provisionada e a simultaneidade reservada em uma função. Será possível fazer isso se você tiver uma função que processa uma carga consistente de invocações durante a semana, mas rotineiramente apresenta picos de tráfego nos finais de semana. Nesse caso, você pode usar a simultaneidade provisionada para definir uma quantidade de linha de base de ambientes para processar as solicitações durante a semana e usar a simultaneidade reservada para processar os picos no final de semana. Considere o diagrama a seguir:



Neste diagrama, suponha que você configure 200 unidades de simultaneidade provisionada e 400 unidades de simultaneidade reservada para `function-orange`. Como você configurou a simultaneidade reservada, `function-orange` não pode usar nenhuma das 600 unidades de simultaneidade não reservada.

Este diagrama tem cinco pontos de interesse:

- Em `t1`, `function-orange` começa a receber solicitações. Como o Lambda inicializou previamente 200 instâncias do ambiente de execução, está tudo pronto para a invocação imediata de `function-orange`.
- Em `t2`, `function-orange` usa toda a simultaneidade provisionada. `function-orange` pode continuar atendendo a solicitações usando simultaneidade reservada, mas essas solicitações podem apresentar latências de inicialização a frio.
- Em `t3`, `function-orange` atinge 400 solicitações simultâneas. Como resultado, `function-orange` usa toda a simultaneidade reservada. Como `function-orange` não pode usar a simultaneidade não reservada, as solicitações começam a sofrer controle de utilização.
- Em `t4`, `function-orange` começa a receber menos solicitações e não está mais sob o controle de utilização.
- Em `t5`, `function-orange` diminui para 200 solicitações simultâneas, portanto, todas as solicitações podem novamente usar a simultaneidade provisionada (ou seja, sem latências de inicialização a frio).

Tanto a simultaneidade reservada quanto a provisionada são contabilizadas para o limite de simultaneidade da sua conta e cotas regionais. Em outras palavras, alocar a simultaneidade reservada e a provisionada pode afetar o grupo de simultaneidade disponível para outras funções. A configuração da simultaneidade provisionada gera cobranças em sua conta da Conta da AWS.

> **Note**
>
> Se a quantidade de simultaneidade provisionada nas versões e aliases de uma função se somar à simultaneidade reservada da função, todas as invocações serão executadas na simultaneidade provisionada. Essa configuração também tem o efeito de controlar a utilização da versão não publicada da função (`$LATEST`), o que impede que ela seja executada. Não é possível alocar mais simultaneidade provisionada do que a simultaneidade reservada para uma função.

Para gerenciar as configurações de simultaneidade provisionada para as funções, consulte Configurar a simultaneidade provisionada para uma função. Para automatizar a escalabilidade da simultaneidade provisionada com base em uma programação ou utilização de aplicações,

consulte Usar o ajuste de escala automático de aplicações para automatizar o gerenciamento da simultaneidade provisionada.

Como o Lambda aloca a simultaneidade provisionada

A simultaneidade provisionada não fica online imediatamente depois que você a configura. O Lambda começa a alocar a simultaneidade provisionada após um ou dois minutos de preparação. Para cada função, o Lambda pode provisionar até 6 mil ambientes de execução a cada minuto, independentemente da Região da AWS. Isso é exatamente igual à taxa de escalabilidade de simultaneidade para funções.

Quando você envia uma solicitação para alocar a simultaneidade provisionada, não pode acessar qualquer um desses ambientes até que o Lambda termine de alocá-los. Por exemplo, se você solicitar 5 mil simultaneidades provisionadas, nenhuma das suas solicitações poderá usar a simultaneidade provisionada até que o Lambda conclua a alocação total dos 5 mil ambientes de execução.

Comparação entre a simultaneidade reservada e a simultaneidade provisionada

A tabela a seguir resume e compara as simultaneidades reservada e provisionada.

Tópico	Simultaneidade reservada	Simultaneidade provisionada
Definição	Número máximo de instâncias do ambiente de execução para a função.	Defina o número de instâncias do ambiente de execução pré-provisionadas para a função.
Comportamento de provisionamento	O Lambda provisiona novas instâncias sob demanda.	O Lambda provisiona as instâncias previamente (ou seja, antes que a função comece a receber solicitações).
Comportamento de inicialização a frio	Possibilidade de latência de inicialização a frio, pois o Lambda precisa criar novas instâncias sob demanda.	Não há latência de inicialização a frio, pois o Lambda não precisa criar instâncias sob demanda.
Comportamento de controle de utilização	Função sofre controle de utilização quando o limite de	Se a simultaneidade reservada não estiver definida:

Tópico	Simultaneidade reservada	Simultaneidade provisionada
	simultaneidade reservada é atingido.	a função usará a simultaneidade não reservada quando o limite de simultaneidade provisionada for atingido. Se a simultaneidade reservada estiver definida: a função sofrerá controle de utilização quando o limite de simultaneidade reservada for atingido.
Comportamento padrão, se não for definido	A função usa a simultaneidade não reservada disponível em sua conta.	O Lambda não provisiona previamente nenhuma instância. Em vez disso, se a simultaneidade reservada não estiver definida: a função usará a simultaneidade não reservada disponível em sua conta. Se a simultaneidade reservada estiver definida: a função usará a simultaneidade reservada.
Definição de preço	Sem custos adicionais.	Incorre em custos adicionais.

Cotas de simultaneidade

O Lambda define cotas para a quantidade total de simultaneidade que é possível usar em todas as funções de uma região. Essas cotas existem em dois níveis:

- Em nível de conta, suas funções podem ter até mil unidades de simultaneidade por padrão. Para aumentar esse limite, consulte Requesting a quota increase (Como solicitar um aumento de cota) no Guia do usuário do Service Quotas.

- Em nível de função, é possível reservar até 900 unidades de simultaneidade para todas as funções por padrão. Independentemente do limite total de simultaneidade da conta, o Lambda sempre reserva cem unidades de simultaneidade para suas funções que não reservam explicitamente a simultaneidade. Por exemplo, se você aumentou o limite de simultaneidade da sua conta para 2 mil, poderá reservar até 1,9 mil unidades de simultaneidade em nível de função.

Para verificar sua cota atual de simultaneidade no nível da conta, use a AWS Command Line Interface (AWS CLI) para executar o seguinte comando:

```
aws lambda get-account-settings
```

Você observará um resultado parecido com o seguinte:

```
{
    "AccountLimit": {
        "TotalCodeSize": 80530636800,
        "CodeSizeUnzipped": 262144000,
        "CodeSizeZipped": 52428800,
        "ConcurrentExecutions": 1000,
        "UnreservedConcurrentExecutions": 900
    },
    "AccountUsage": {
        "TotalCodeSize": 410759889,
        "FunctionCount": 8
    }
}
```

`ConcurrentExecutions` é sua cota total de simultaneidade no nível da conta. `UnreservedConcurrentExecutions` é a quantidade de simultaneidade reservada que você ainda pode alocar para suas funções.

À medida que sua função recebe mais solicitações, o Lambda aumenta automaticamente a escala do número de ambientes de execução para processar essas solicitações até que seja atingido o limite de simultaneidade da sua conta. No entanto, para evitar que o limite de escalabilidade seja ultrapassado em resposta a picos repentinos de tráfego, o Lambda limita a velocidade do ajuste de escala das funções. Essa taxa de escalabilidade de simultaneidade é a taxa máxima na qual as funções da conta podem reduzir a escala horizontalmente em resposta ao aumento de solicitações. Ou seja, a velocidade com que o Lambda é capaz de criar novos ambientes de execução. A taxa

de escalabilidade de simultaneidade difere do limite de simultaneidade no nível da conta, que é a quantidade total de simultaneidade disponível para suas funções.

Em cada Região da AWS, e para cada função, sua taxa de escalabilidade de simultaneidade é de mil instâncias de ambiente de execução a cada dez segundos. Em outras palavras, a cada dez segundos, o Lambda pode alocar no máximo mil instâncias adicionais de ambiente de execução para cada uma das suas funções.

Normalmente, você não precisa se preocupar com esse limite. A taxa de escalabilidade do Lambda é suficiente para a maioria dos casos de uso.

É importante ressaltar que a taxa de escalabilidade de simultaneidade é um limite aplicado por função. Isso significa que cada função da sua conta pode ajustar a escala independentemente de outras funções.

Para obter mais informações sobre comportamentos de escalabilidade, consulte Comportamento de escalabilidade do Lambda.

Configurar a simultaneidade reservada para uma função

No Lambda, simultaneidade corresponde ao número de solicitações em andamento que a função está processando, no momento. Há dois tipos de controle de simultaneidade disponíveis:

- Simultaneidade reservada: representa o número máximo de instâncias simultâneas alocadas para a função. Quando uma função tem simultaneidade reservada, nenhuma outra função pode usar essa simultaneidade. A simultaneidade reservada é útil para garantir que suas funções mais críticas sempre tenham simultaneidade suficiente para lidar com as solicitações recebidas. Configurar a simultaneidade reservada para uma função não acarreta cobranças adicionais.
- Simultaneidade provisionada: é o número de ambientes de execução inicializados previamente alocados para a função. Esses ambientes de execução estão prontos para responder imediatamente às solicitações de função de entrada. A simultaneidade provisionada é útil para reduzir as latências de inicialização a frio para funções. A configuração da simultaneidade provisionada gera cobranças na sua Conta da AWS.

Este tópico detalha como gerenciar e configurar a simultaneidade reservada. Para uma visão geral conceitual desses dois tipos de controles de simultaneidade, consulte Simultaneidade reservada e simultaneidade provisionada. Para obter informações, sobre a configuração da simultaneidade provisionada, consulte the section called “Configurar a simultaneidade provisionada”.

> **Note**
>
> As funções do Lambda vinculadas a um mapeamento da origem do evento do Amazon MQ têm uma simultaneidade máxima padrão. Para o Apache Active MQ, o número máximo de instâncias simultâneas é 5. Para o Rabbit MQ, o número máximo de instâncias simultâneas é 1. Definir a simultaneidade reservada ou provisionada para a função não altera esses limites. Para solicitar um aumento na simultaneidade máxima padrão ao usar o Amazon MQ, entre em contato com AWS Support.

Seções

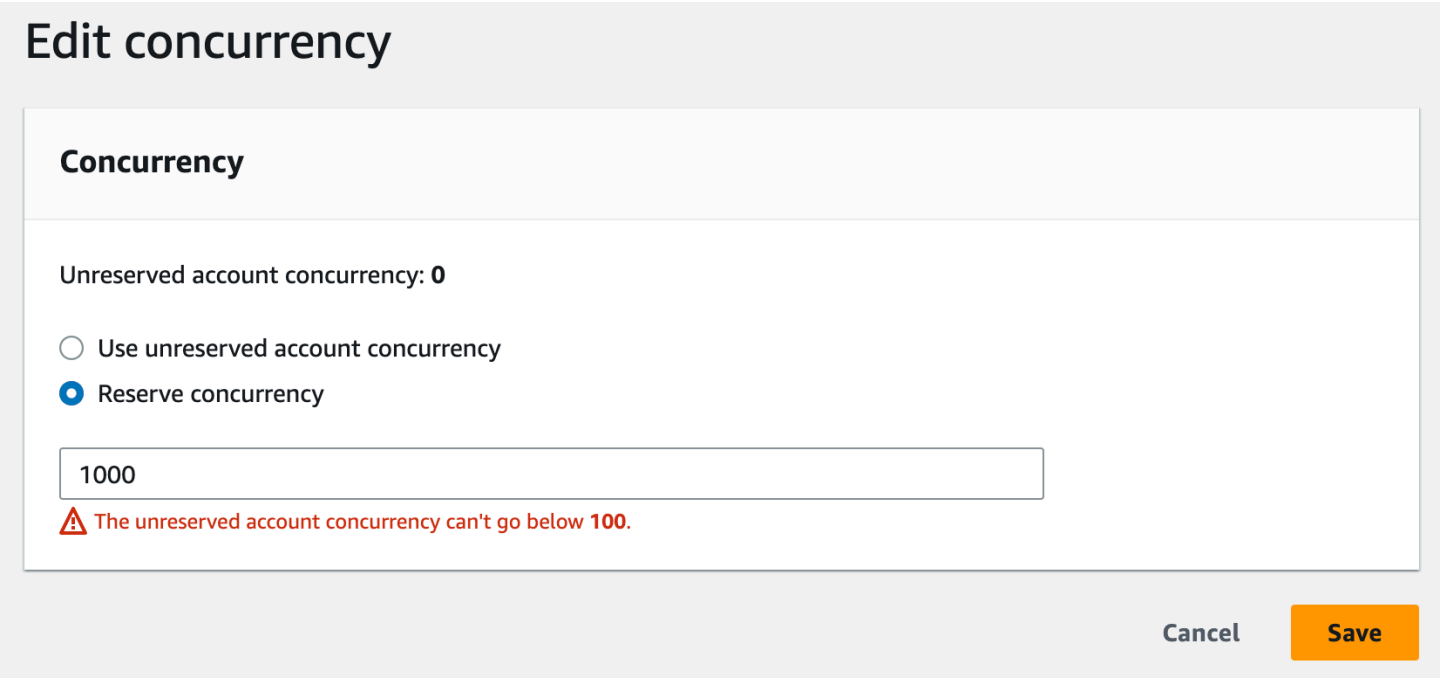
Configurar a simultaneidade reservada

Você pode definir configurações de simultaneidade reservada para uma função usando o console do Lambda ou a API do Lambda.

Para reservar simultaneidade para uma função (console)

1. Abra a página Funções do console do Lambda.
2. Escolha a função para a qual você deseja reservar a simultaneidade.
3. Escolha Configuration (Configuração) e escolha (Concurrency (Simultaneidade).
4. Em Concurrency (Simultaneidade), selecione Edit (Editar).
5. Selecione Reserve concurrency (Reservar simultaneidade). Insira a quantidade de simultaneidade para reservar para a função.
6. Escolha Salvar.

Você pode reservar até o valor da simultaneidade da conta não reservada menos 100. As 100 unidades restantes de simultaneidade se destinam a funções que não estão usando simultaneidade reservada. Por exemplo, se sua conta tiver um limite de simultaneidade de 1 mil, você não poderá reservar todas as 1 mil unidades de simultaneidade para uma única função.



Reservar simultaneidade para uma função afeta o grupo de simultaneidades disponíveis para outras funções. Por exemplo, se você reservar 100 unidades de simultaneidade para `function-a`, outras

funções da sua conta deverão compartilhar as 900 unidades restantes de simultaneidade, mesmo que `function-a` não use todas as 100 unidades de simultaneidade reservadas.

Para controlar intencionalmente a utilização de uma função, defina a simultaneidade reservada como zero. Isso impede que sua função processe todos os eventos até que você remova o limite.

Para configurar a simultaneidade reservada com a API do Lambda, use as operações da API a seguir.

- PutFunctionConcurrency
- GetFunctionConcurrency
- DeleteFunctionConcurrency

Por exemplo, para configurar a simultaneidade reservada com a AWS Command Line Interface (CLI), use o comando `put-function-concurrency`. O comando a seguir reserva 100 unidades de simultaneidade para uma função denominada `my-function`:

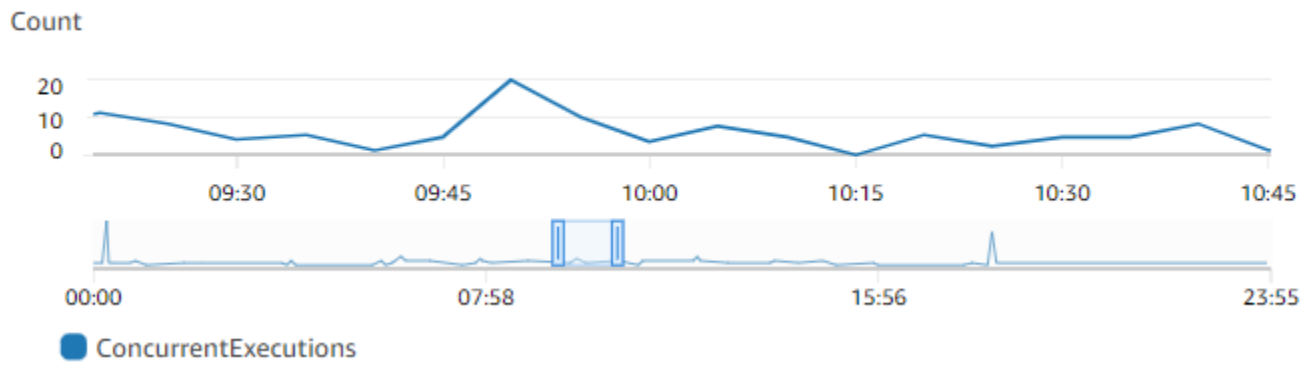
```
aws lambda put-function-concurrency --function-name my-function \
    --reserved-concurrent-executions 100
```

Você observará um resultado parecido com o seguinte:

```
{
    "ReservedConcurrentExecutions": 100
}
```

Estimar com precisão a simultaneidade reservada necessária para uma função

Se sua função estiver atendendo a tráfego no momento, você poderá visualizar facilmente as métricas de simultaneidade usando as métricas do CloudWatch. Especificamente, a métrica `ConcurrentExecutions` apresenta o número de invocações simultâneas para cada função em sua conta.



O gráfico anterior sugere que essa função atende a uma média de cinco a dez solicitações simultâneas e atinge o pico de 20 solicitações em um dia típico. Suponha que existam muitas outras funções na sua conta. Se essa função for essencial para sua aplicação e você não desejar descartar solicitação alguma, use um número maior ou igual a 20 como sua definição de simultaneidade reservada.

Com alternativa, lembre-se de que você também pode calcular a simultaneidade usando a fórmula a seguir:

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

Multiplicar a média de solicitações por segundo pela duração média da solicitação em segundos fornece uma estimativa aproximada do quanto de simultaneidade você precisa reservar. É possível estimar a média de solicitações por segundo usando a métrica `Invocation` e a duração média da solicitação em segundos usando a métrica `Duration`. Consulte Trabalhar com métricas de funções Lambda para obter mais detalhes.

Configurar a simultaneidade provisionada para uma função

No Lambda, simultaneidade corresponde ao número de solicitações em andamento que a função está processando, no momento. Há dois tipos de controle de simultaneidade disponíveis:

- Simultaneidade reservada: representa o número máximo de instâncias simultâneas alocadas para a função. Quando uma função tem simultaneidade reservada, nenhuma outra função pode usar essa simultaneidade. A simultaneidade reservada é útil para garantir que suas funções mais críticas sempre tenham simultaneidade suficiente para lidar com as solicitações recebidas. Configurar a simultaneidade reservada para uma função não acarreta cobranças adicionais.
- Simultaneidade provisionada: é o número de ambientes de execução inicializados previamente alocados para a função. Esses ambientes de execução estão prontos para responder imediatamente às solicitações de função de entrada. A simultaneidade provisionada é útil para reduzir as latências de inicialização a frio para funções. A configuração da simultaneidade provisionada gera cobranças na sua Conta da AWS.

Este tópico detalha como gerenciar e configurar a simultaneidade provisionada. Para uma visão geral conceitual desses dois tipos de controles de simultaneidade, consulte Simultaneidade reservada e simultaneidade provisionada. Para obter mais informações sobre a configuração de simultaneidade reservada, consulte the section called “Configurar a simultaneidade reservada”.

> **Note**
>
> As funções do Lambda vinculadas a um mapeamento da origem do evento do Amazon MQ têm uma simultaneidade máxima padrão. Para o Apache Active MQ, o número máximo de instâncias simultâneas é 5. Para o Rabbit MQ, o número máximo de instâncias simultâneas é 1. Definir a simultaneidade reservada ou provisionada para a função não altera esses limites. Para solicitar um aumento na simultaneidade máxima padrão ao usar o Amazon MQ, entre em contato com AWS Support.

Seções

- Usar variáveis de ambiente para visualizar e controlar o comportamento da simultaneidade provisionada
- Entender o comportamento de log e cobrança com simultaneidade provisionada
- Usar o ajuste de escala automático de aplicações para automatizar o gerenciamento da simultaneidade provisionada

Configurar a simultaneidade provisionada

Você pode definir configurações de simultaneidade provisionada para uma função usando o console do Lambda ou a API do Lambda.

Para alocar simultaneidade provisionada para uma função (console)

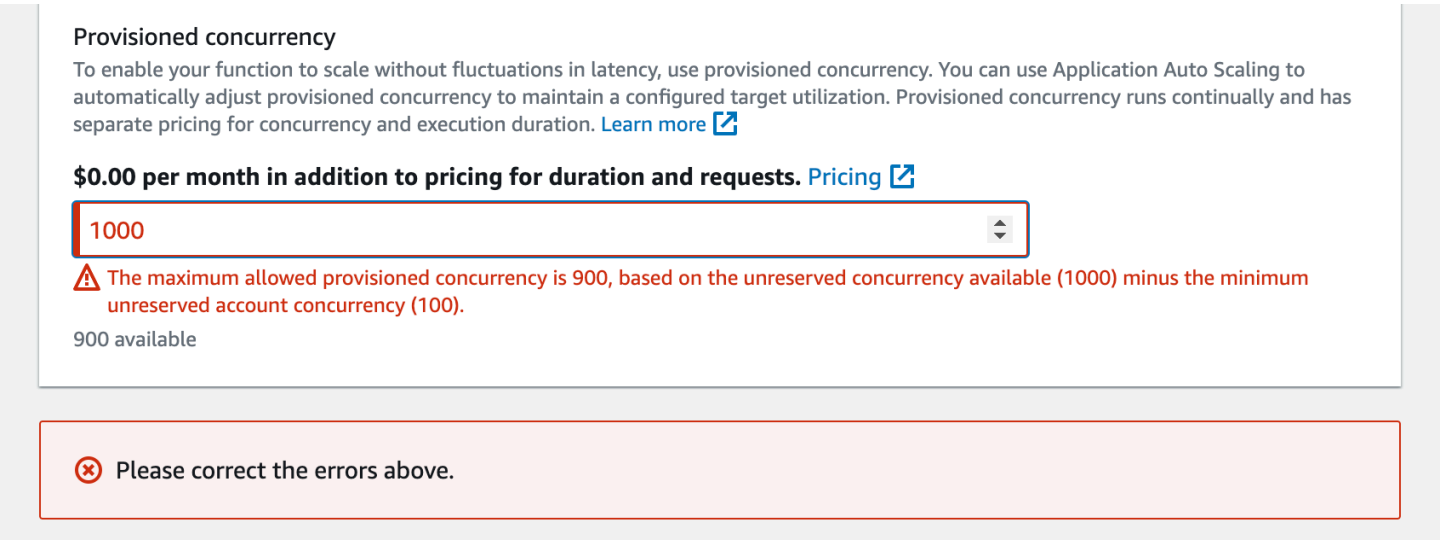
1. Abra a página Funções do console do Lambda.
2. Escolha a função para a qual você deseja alocar a simultaneidade provisionada.
3. Escolha Configuration (Configuração) e escolha (Concurrency (Simultaneidade).
4. Em Provisioned concurrency configurations (Configurações de simultaneidade provisionada), escolha Add configuration (Adicionar configuração).
5. Escolha o tipo de qualificador e o alias ou a versão.

> **Note**
>
> Você não pode usar simultaneidade provisionada com a versão $LATEST de qualquer função.
> Se a função tiver uma origem de evento, certifique-se de que a origem de evento aponte para o alias ou a versão correta da função. Caso contrário, a função não usará ambientes de simultaneidade provisionada.

6. Insira um número em Simultaneidade provisionada. O Lambda fornece uma estimativa dos custos mensais.
7. Escolha Salvar.

Você pode configurar até a Simultaneidade de conta não reservada na sua conta, menos 100. As 100 unidades restantes de simultaneidade se destinam a funções que não estão usando simultaneidade reservada. Por exemplo, se sua conta tem um limite de simultaneidade de 1 mil e você não atribuiu qualquer simultaneidade reservada ou provisionada a qualquer uma das outras

funções, você pode configurar no máximo 900 unidades de simultaneidade provisionada para uma única função.



Configurar simultaneidade provisionada para uma função causa um impacto no grupo de simultaneidades disponíveis para outras funções. Por exemplo, se você configurar 100 unidades de simultaneidade provisionada para a `function-a`, outras funções da sua conta deverão compartilhar as 900 unidades restantes de simultaneidade. Isso ocorrerá mesmo se a `function-a` não usar todas as 100 unidades.

É possível alocar tanto a simultaneidade reservada como a provisionada para a mesma função. Nesses casos, a simultaneidade provisionada não pode exceder a simultaneidade reservada.

Essa limitação se estende às versões da função. A simultaneidade máxima provisionada que você pode atribuir a uma versão de função específica é igual à simultaneidade reservada da função menos a simultaneidade provisionada em outras versões da função.

Para configurar a simultaneidade provisionada com a API do Lambda, use as operações da API a seguir.

- PutProvisionedConcurrencyConfig
- GetProvisionedConcurrencyConfig
- ListProvisionedConcurrencyConfigs
- DeleteProvisionedConcurrencyConfig

Por exemplo, para configurar a simultaneidade provisionada com a AWS Command Line Interface (CLI), use o comando `put-provisioned-concurrency-config`. O comando a seguir

aloca 100 unidades de simultaneidade provisionada para o alias BLUE de uma função chamada `my-function`:

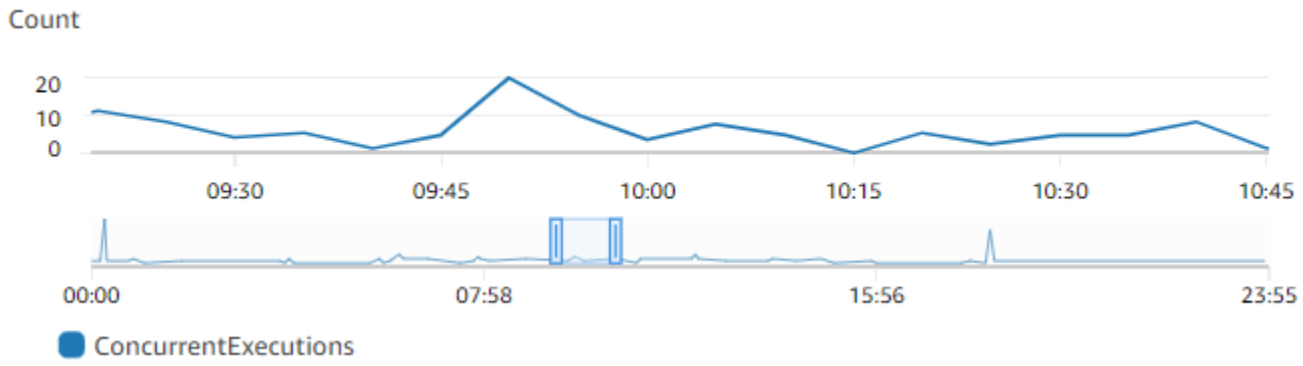
```
aws lambda put-provisioned-concurrency-config --function-name my-function \
  --qualifier BLUE \
  --provisioned-concurrent-executions 100
```

Você observará um resultado parecido com o seguinte:

```
{
  "Requested ProvisionedConcurrentExecutions": 100,
  "Allocated ProvisionedConcurrentExecutions": 0,
  "Status": "IN_PROGRESS",
  "LastModified": "2023-01-21T11:30:00+0000"
}
```

Estimar a simultaneidade provisionada necessária com precisão para uma função

Você pode visualizar as métricas de simultaneidade de qualquer função ativa usando as métricas do CloudWatch. Especificamente, a métrica `ConcurrentExecutions` apresenta o número de invocações simultâneas para as funções da conta.



O gráfico anterior sugere que essa função atende a uma média de cinco a dez solicitações simultâneas em qualquer ocasião e atinge o pico de 20 solicitações. Suponha que existam muitas outras funções na sua conta. Se essa função for essencial para sua aplicação e você precisar de uma resposta de baixa latência em cada invocação, configure pelo menos 20 unidades de simultaneidade provisionada.

Lembre-se de que você também pode calcular a simultaneidade usando a seguinte fórmula:

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

Para estimar quanto de simultaneidade você precisa, multiplique a média de solicitações por segundo pela duração média das solicitações em segundos. É possível estimar a média de solicitações por segundo usando a métrica `Invocation` e a duração média da solicitação em segundos usando a métrica `Duration`.

Quando você configura simultaneidade provisionada, o Lambda sugere a adição de uma reserva de 10% na quantidade de simultaneidade de que a função normalmente precisa. Por exemplo, se a função geralmente atinge o pico de 200 solicitações simultâneas, defina a simultaneidade provisionada como 220 (200 solicitações simultâneas + 10% = 220 de simultaneidade provisionada).

Otimizar o código da função ao usar a simultaneidade provisionada

Se você estiver usando simultaneidade provisionada, considere reestruturar o código da função para otimizar a baixa latência. Para funções que usam simultaneidade provisionada, o Lambda executa qualquer código de inicialização, como o carregamento de bibliotecas e a criação de instâncias de clientes) no momento da alocação. Portanto, é aconselhável mover o máximo de inicialização para fora do manipulador da função principal para evitar afetar a latência durante as invocações reais da função. Por outro lado, a inicialização de bibliotecas ou a criação de instâncias de clientes no código do manipulador principal significa que a função precisará executar isso toda vez que for invocada (isso ocorre independentemente de você estar ou não usando simultaneidade provisionada).

Para invocações sob demanda, o Lambda pode precisar executar novamente o código de inicialização sempre que a função passar por uma inicialização a frio. Para essas funções, você pode optar por adiar a inicialização de um recurso específico até que a função precise dele. Por exemplo, considere o seguinte fluxo de controle para um manipulador do Lambda:

```
def handler(event, context):
    ...
    if ( some_condition ):
        // Initialize CLIENT_A to perform a task
    else:
        // Do nothing
```

No exemplo anterior, em vez de inicializar o `CLIENT_A` fora do manipulador principal, o desenvolvedor realizou a inicialização com a instrução `if`. Ao fazer isso, o Lambda só executará esse código se `some_condition` for satisfeito. Se o autor inicializar `CLIENT_A` fora do manipulador

principal, o Lambda executará esse código em cada inicialização a frio. Isso pode aumentar a latência geral.

Usar variáveis de ambiente para visualizar e controlar o comportamento da simultaneidade provisionada

É possível que a função use toda a simultaneidade provisionada. O Lambda usa instâncias sob demanda para lidar com qualquer excesso de tráfego. Para determinar o tipo de inicialização que o Lambda usou para um ambiente específico, verifique o valor da variável de ambiente `AWS_LAMBDA_INITIALIZATION_TYPE`. Essa variável tem dois valores possíveis: `provisioned-concurrency` ou `on-demand`. O valor de `AWS_LAMBDA_INITIALIZATION_TYPE` é imutável e permanece constante durante toda a vida útil do ambiente. Para verificar o valor de uma variável de ambiente no código da função, consulte ???.

Se você estiver usando os runtimes do .NET 6 ou do .NET 7, poderá configurar a variável de ambiente `AWS_LAMBDA_DOTNET_PREJIT` para melhorar a latência das funções, mesmo se elas não usarem simultaneidade provisionada. O runtime do .NET emprega compilação e inicialização lentas para cada biblioteca que o código chama pela primeira vez. Como resultado, a primeira invocação de uma função do Lambda pode levar mais tempo do que as subsequentes. Para mitigar isso, é possível escolher um dos três valores para `AWS_LAMBDA_DOTNET_PREJIT`:

- `ProvisionedConcurrency`: o Lambda executa a compilação JIT antecipada para todos os ambientes usando simultaneidade provisionada. Este é o valor padrão.
- `Always`: o Lambda executa a compilação JIT antecipada para cada ambiente, mesmo que a função não use simultaneidade provisionada.
- `Never`: o Lambda desativa a compilação JIT antecipada para todos os ambientes.

Entender o comportamento de log e cobrança com simultaneidade provisionada

Para instâncias de simultaneidade provisionada, o código de inicialização da função é executado durante a alocação e periodicamente, quando o Lambda recicla as instâncias do ambiente. O tempo de inicialização pode ser visualizado em logs e rastreamentos depois que uma instância e ambiente processa uma solicitação. É importante notar que o Lambda cobra pela inicialização mesmo que a instância de ambiente nunca processe uma solicitação. A simultaneidade provisionada é executada continuamente e gera um faturamento separado dos custos de inicialização e de invocação. Para obter mais detalhes, consulte Preço do AWS Lambda.

Além disso, quando você configura uma função do Lambda com simultaneidade provisionada, o Lambda pré-inicializa esse ambiente de execução para que ele esteja disponível antes das solicitações de invocação da função. No entanto, sua função publica logs de invocação no CloudWatch somente quando a função é realmente invocada. Portanto, o campo Init Duration aparece na linha do log `REPORT` da primeira invocação de função, mesmo que a inicialização tenha ocorrido antes do tempo. Isso não significa que a função foi inicializada a frio.

Usar o ajuste de escala automático de aplicações para automatizar o gerenciamento da simultaneidade provisionada

Você pode usar o Application Auto Scaling para gerenciar a simultaneidade provisionada em uma programação ou com base na utilização. Se a função receber padrões previsíveis de tráfego, use ajuste de escala programado. Se você deseja que a função mantenha uma porcentagem de utilização específica, use uma política de ajuste de escala de rastreamento de destino.

Escalabilidade programada

Com o Application Auto Scaling, você pode definir sua própria programação de ajuste de escala de acordo com alterações de carga previsíveis. Para obter mais informações e exemplos, consulte Ajuste de escala programado para o Application Auto Scaling no Guia do usuário do Application Auto Scaling e Programar simultaneidade provisionada do AWS Lambda para picos recorrentes de uso no blog de computação da AWS.

Monitoramento do objetivo

Com o rastreamento de destino, o Application Auto Scaling cria e gerencia um conjunto de alarmes do CloudWatch com base na maneira como você define sua política de ajuste de escala. Quando esses alarmes são ativados, o Application Auto Scaling ajusta automaticamente a quantidade de ambientes alocados usando simultaneidade provisionada. Use rastreamento de destino para aplicações que não têm padrões de tráfego previsíveis.

Para ajustar a escala da simultaneidade provisionada usando o rastreamento de destino, use as operações da API do Application Auto Scaling `RegisterScalableTarget` e `PutScalingPolicy`. Por exemplo, se você estiver usando a AWS Command Line Interface (CLI), siga estas etapas:

1. Registre o alias de uma função como um destino de escalabilidade. O exemplo a seguir registra o alias BLUE de uma função denominada `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. Aplique uma política de escalabilidade ao destino. O exemplo a seguir configura o Application Auto Scaling para ajustar a configuração de simultaneidade provisionada para um alias a fim de manter a utilização próxima de 70%, mas é possível aplicar qualquer valor entre 10% e 90%.

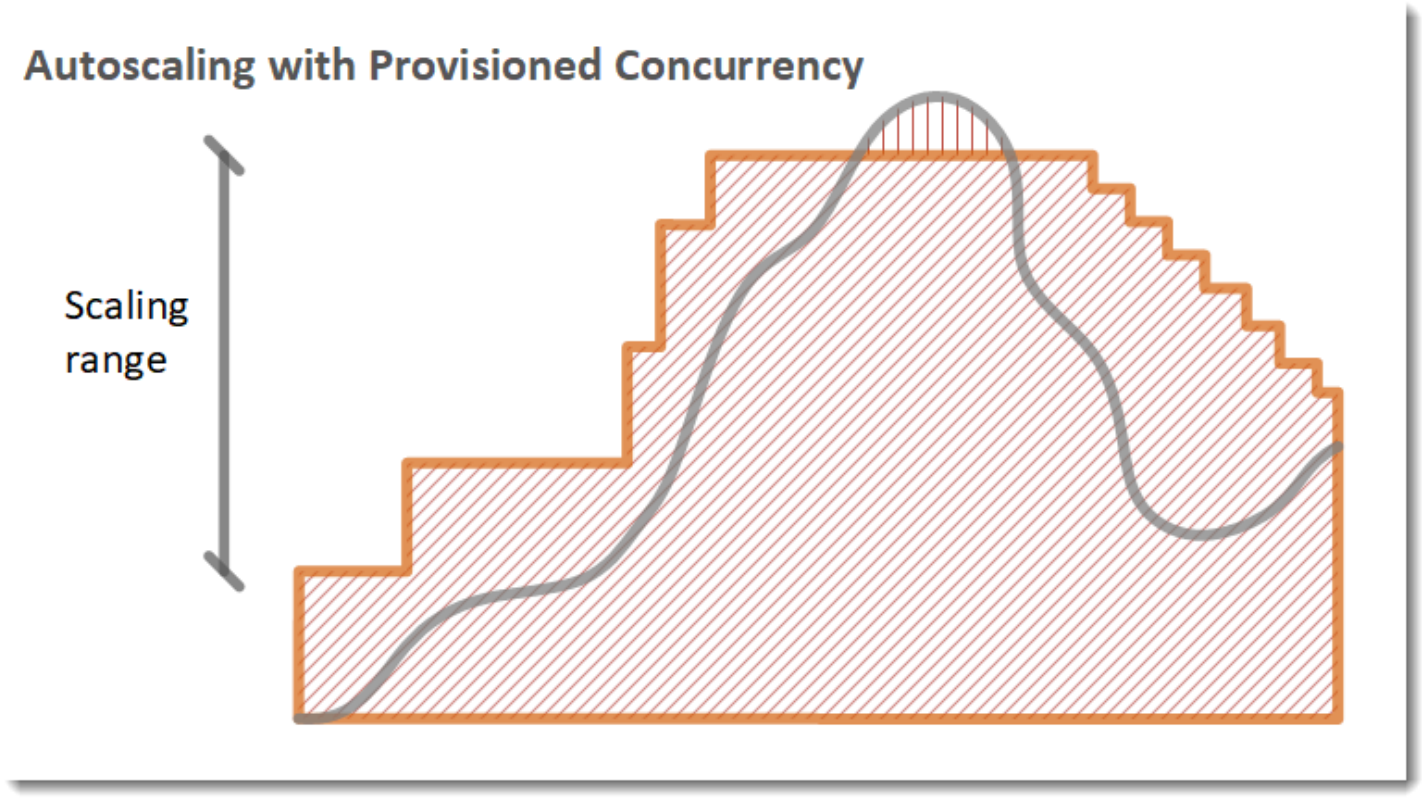
```
aws application-autoscaling put-scaling-policy \
    --service-namespace lambda \
    --scalable-dimension lambda:function:ProvisionedConcurrency \
    --resource-id function:my-function:BLUE \
    --policy-name my-policy \
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration '{ "TargetValue":
 0.7, "PredefinedMetricSpecification": { "PredefinedMetricType":
 "LambdaProvisionedConcurrencyUtilization" }}'
```

Você deve ver uma saída semelhante a:

```
{
    "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"
        }
    ]
}
```

O Application Auto Scaling cria dois alarmes no CloudWatch. O primeiro alarme será acionado quando a utilização da simultaneidade provisionada exceder consistentemente 70%. Quando isso acontecer, o Application Auto Scaling alocará mais simultaneidade provisionada para reduzir a utilização. O segundo alarme será acionado quando a utilização for consistentemente inferior a 63% (90% da meta de 70%). Quando isso acontecer, o Application Auto Scaling reduzirá a simultaneidade provisionada do alias.

No exemplo a seguir, uma função é dimensionada entre uma quantidade mínima e máxima de simultaneidade provisionada com base na utilização.



Legenda

- Instâncias de função
- Solicitações abertas
- Simultaneidade provisionada

- |||||||||||||||| Simulta
  padrão

Quando o número de solicitações abertas aumenta, o Application Auto Scaling aumenta a simultaneidade provisionada em grandes etapas até atingir o máximo configurado. Depois disso, a função poderá continuar a ajustar a escala com base na simultaneidade padrão não reservada se você não tiver atingido o limite de simultaneidade da conta. Quando a utilização diminui e permanece baixa, o Application Auto Scaling diminui a simultaneidade provisionada em etapas periódicas menores.

Os dois alarmes do Application Auto Scaling usam a estatística média por padrão. Funções que sofrem intermitências rápidas de tráfego podem não acionar esses alarmes. Por exemplo, suponha que a função do Lambda seja executada rapidamente (ou seja, entre 20 e 100 ms) e o tráfego ocorra em intermitências rápidas. Nesse caso, o número de solicitações excede a simultaneidade provisionada alocada durante a intermitência. No entanto, o Application Auto Scaling exige que a carga de intermitência seja mantida por pelo menos três minutos para provisionar ambientes adicionais. Além disso, os dois alarmes do CloudWatch exigem três pontos de dados que atinjam a média de destino para ativar a política de ajuste de escala automático. Se sua função apresentar picos rápidos de tráfego, o uso da estatística Máxima em vez da estatística Média pode ser mais eficaz no escalonamento da simultaneidade provisionada a fim de minimizar as inicializações a frio.

Para obter mais informações sobre políticas de ajuste de escala de rastreamento de destino, consulte Políticas de ajuste de escala de rastreamento de destino para o Application Auto Scaling.

Comportamento de escalabilidade do Lambda

À medida que sua função recebe mais solicitações, o Lambda aumenta automaticamente a escala do número de ambientes de execução para processar essas solicitações até que seja atingido o limite de simultaneidade da sua conta. No entanto, para evitar que o limite de escalabilidade seja ultrapassado em resposta a picos repentinos de tráfego, o Lambda limita a velocidade do ajuste de escala das funções. Essa taxa de escalabilidade de simultaneidade é a taxa máxima na qual as funções em sua conta podem ser escaladas em resposta ao aumento de solicitações. Ou seja, a velocidade com que o Lambda é capaz de criar novos ambientes de execução. A taxa de escalabilidade de simultaneidade difere do limite de simultaneidade no nível da conta, que é a quantidade total de simultaneidade disponível para suas funções.

Taxa de escalabilidade da simultaneidade

Em cada Região da AWS, e para cada função, sua taxa de escalabilidade de simultaneidade é de mil instâncias de ambiente de execução a cada dez segundos. Em outras palavras, a cada dez segundos, o Lambda pode alocar no máximo mil instâncias adicionais de ambiente de execução para cada uma das suas funções.

Normalmente, você não precisa se preocupar com esse limite. A taxa de escalabilidade do Lambda é suficiente para a maioria dos casos de uso.

É importante ressaltar que a taxa de escalabilidade de simultaneidade é um limite no nível da função. Isso significa que cada função da sua conta pode ajustar a escala independentemente de outras funções.

> **Note**
>
> Na prática, o Lambda faz o possível para reabastecer sua taxa de escalabilidade de simultaneidade continuamente, em vez de fazer uma única recarga de mil unidades a cada 10 segundos.

O Lambda não acumula partes não utilizadas da taxa de escalabilidade de simultaneidade. Isso significa que sua taxa de escalabilidade será sempre de mil unidades simultâneas, no máximo. Por exemplo, se você não usar nenhuma das mil unidades de simultaneidade disponíveis em um intervalo de 10 segundos, não acumulará mil unidades adicionais para o próximo intervalo de 10 segundos. Sua taxa de escalabilidade de simultaneidade ainda será de mil no próximo intervalo de 10 segundos.

Enquanto sua função continuar recebendo um número cada vez maior de solicitações, o Lambda ajustará a escala com a taxa mais rápida disponível para você, até o limite de simultaneidade da sua conta. Para limitar o valor de simultaneidade das funções individuais, é necessário configurar a simultaneidade reservada. Se a quantidade de solicitações recebidas for maior do que a capacidade da função de fazer o ajuste de escala ou se a função já estiver na simultaneidade máxima, as solicitações adicionais vão apresentar falha com um erro de controle de utilização (código de status 429).

Monitorar a simultaneidade

O Lambda emite métricas da CloudWatch Amazon para ajudar você a monitorar a simultaneidade de suas funções. Este tópico explica essas métricas e como interpretá-las.

Seções

Métricas gerais de simultaneidade

Use as métricas a seguir para monitorar a simultaneidade das funções do Lambda. A granularidade de cada métrica é de um minuto.

- `ConcurrentExecutions`: o número de invocações simultâneas ativas em um determinado ponto no tempo. O Lambda gera essa métrica para todas as funções, versões e aliases. Para qualquer função no console do Lambda, o Lambda exibe o grafo para `ConcurrentExecutions` nativamente na guia Monitoramento, em Métricas. Visualize essa métrica usando MAX.
- `UnreservedConcurrentExecutions`: o número de invocações simultâneas ativas que estão usando a simultaneidade não reservada. O Lambda gera essa métrica em todas as funções de uma região. Visualize essa métrica usando MAX.
- `ClaimedAccountConcurrency`: quantidade de simultaneidade que não está disponível para invocações sob demanda. `ClaimedAccountConcurrency` é igual a `UnreservedConcurrentExecutions` mais a quantidade de simultaneidade alocada (ou seja, a simultaneidade total reservada mais a simultaneidade total provisionada). Se `ClaimedAccountConcurrency` exceder o limite de simultaneidade da conta, você poderá solicitar um limite maior de simultaneidade para a conta. Visualize essa métrica usando MAX. Para ter mais informações, consulte Trabalhar com a métrica `ClaimedAccountConcurrency`.

Métricas de simultaneidade provisionada

Use as métricas a seguir para monitorar as funções do Lambda usando a simultaneidade provisionada. A granularidade de cada métrica é de um minuto.

- `ProvisionedConcurrentExecutions`: o número de instâncias do ambiente de execução que estão ativamente processando uma invocação na simultaneidade provisionada. O Lambda gera essa métrica para cada versão e alias da função com a simultaneidade provisionada configurada. Visualize essa métrica usando MAX.

`ProvisionedConcurrentExecutions` não é o mesmo que o número total de simultaneidade provisionada que você aloca. Por exemplo, suponha que você aloque 100 unidades de simultaneidade provisionada para uma versão de função. Durante um determinado minuto, se no máximo 50 desses 100 ambientes de execução estivessem lidando com invocações simultaneamente, o valor de MAX (`ProvisionedConcurrentExecutions`) seria 50.

- `ProvisionedConcurrentInvocations`: o número de vezes que o Lambda invoca o código da função usando a simultaneidade provisionada. O Lambda gera essa métrica para cada versão e alias da função com a simultaneidade provisionada configurada. Visualize essa métrica usando SUM.

`ProvisionedConcurrentInvocations` difere de `ProvisionedConcurrentExecutions` porque `ProvisionedConcurrentInvocations` conta o número total de invocações, enquanto `ProvisionedConcurrentExecutions` conta o número de ambientes ativos. Para entender essa distinção, considere o seguinte cenário:

Neste exemplo, suponha que você receba uma invocação por minuto e cada invocação leve dois minutos para ser concluída. Cada barra horizontal laranja representa uma única solicitação. Suponha que você aloque dez unidades de simultaneidade provisionada para essa função de forma que cada solicitação seja executada na simultaneidade provisionada.

Entre os minutos 0 e 1, entra `Request 1`. No minuto 1, o valor de MAX (`ProvisionedConcurrentExecutions`) é 1, pois no máximo um ambiente de execução estava ativo durante o último minuto. O valor de SUM (`ProvisionedConcurrentInvocations`) também é 1, pois uma nova solicitação foi recebida no último minuto.

Entre os minutos 1 e 2, `Request 2` entra e `Request 1` continua em execução. No minuto 2, o valor de MAX (`ProvisionedConcurrentExecutions`) é 2, pois no máximo dois ambientes de execução estavam ativos durante o último minuto. Entretanto, o valor de SUM (`ProvisionedConcurrentInvocations`) também é 1, pois somente uma nova solicitação foi recebida durante o último minuto. Esse comportamento métrico continua até o final do exemplo.

- `ProvisionedConcurrencySpilloverInvocations`: o número de vezes que o Lambda invoca a função na simultaneidade padrão (reservada ou não reservada) quando toda a simultaneidade provisionada está em uso. O Lambda gera essa métrica para cada versão e alias da função com a simultaneidade provisionada configurada. Visualize essa métrica usando SUM. O valor de `ProvisionedConcurrentInvocations` + `ProvisionedConcurrencySpilloverInvocations` deve ser igual ao número total de invocações da função (ou seja, a métrica `Invocations`).

  `ProvisionedConcurrencyUtilization`: a porcentagem de simultaneidade provisionada em uso (ou seja, o valor de `ProvisionedConcurrentExecutions` dividido pela quantidade total de simultaneidade provisionada alocada). O Lambda gera essa métrica para cada versão e alias da função com a simultaneidade provisionada configurada. Visualize essa métrica usando MAX.

Por exemplo, suponha que você provisione 100 unidades de simultaneidade provisionada para uma versão de função. Durante um determinado minuto, se no máximo 60 desses 100 ambientes de execução estivessem processando invocações simultaneamente, o valor de MAX (`ProvisionedConcurrentExecutions`) seria 60 e o valor de MAX (`ProvisionedConcurrentUtilization`) seria 0,6.

Um valor alto para `ProvisionedConcurrencySpilloverInvocations` pode indicar que você precisa alocar simultaneidade provisionada adicional para a função. Como alternativa, você pode configurar o Application Auto Scaling para processar o ajuste de escala automático da simultaneidade provisionada com base em limites predefinidos.

Por outro lado, valores consistentemente baixos para `ProvisionedConcurrencyUtilization` podem indicar que você superalocou a simultaneidade provisionada para a função.

Trabalhar com a métrica `ClaimedAccountConcurrency`

O Lambda usa a métrica `ClaimedAccountConcurrency` para determinar a quantidade de simultaneidade que está disponível na sua conta para invocações sob demanda. O Lambda calcula `ClaimedAccountConcurrency` usando a seguinte fórmula:

```
ClaimedAccountConcurrency = UnreservedConcurrentExecutions + (allocated concurrency)
```

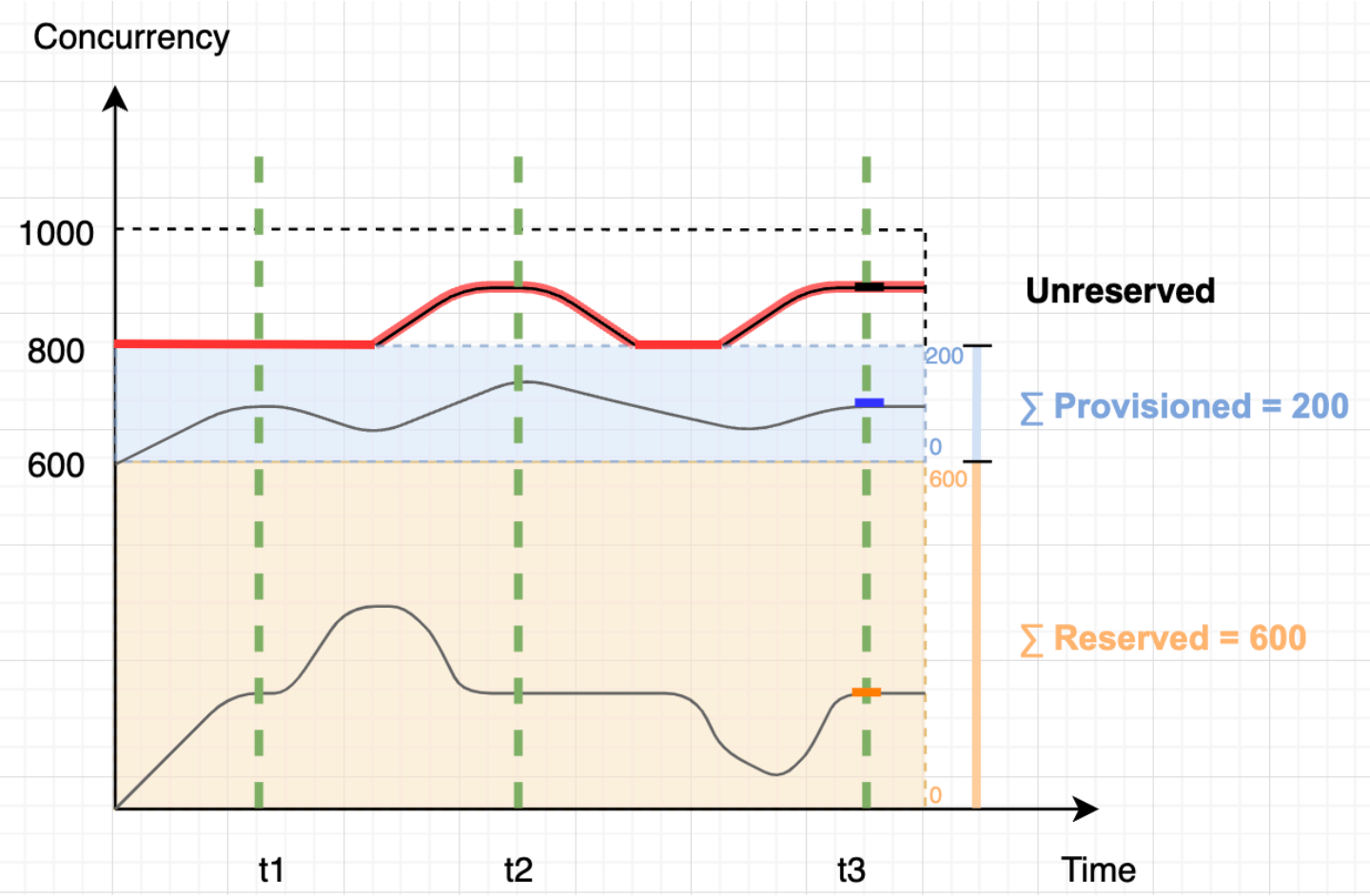
`UnreservedConcurrentExecutions` é o número de invocações simultâneas ativas que estão usando a simultaneidade não reservada. A simultaneidade alocada é a soma das duas partes a seguir (substituindo RC por “simultaneidade reservada” e PC por “simultaneidade provisionada”):

- O total de RC em todas as funções em uma região.
- O total de PC em todas as funções em uma região que usam PC, excluindo as funções que usam RC.

> **Note**
>
> Você não pode alocar mais PC do que RC para uma função. Portanto, a função RC é sempre maior ou igual à sua PC. Para calcular a contribuição para a simultaneidade alocada dessas funções com PC e RC, o Lambda considera apenas RC, que é o máximo das duas.

O Lambda usa a métrica `ClaimedAccountConcurrency` em vez de `ConcurrentExecutions` para determinar a quantidade de simultaneidade que está disponível para invocações sob demanda. Embora a métrica `ConcurrentExecutions` seja útil para rastrear o número de invocações simultâneas ativas, ela nem sempre reflete a verdadeira disponibilidade de simultaneidade. Isso ocorre porque o Lambda também considera a simultaneidade reservada e a simultaneidade provisionada para determinar a disponibilidade.

Para ilustrar `ClaimedAccountConcurrency`, considere um cenário em que você configura muitas simultaneidades reservadas e provisionadas nas funções que, em grande parte, não são usadas. No exemplo a seguir, suponha que o limite de simultaneidade da sua conta seja 1.000 e que você tenha duas funções principais na conta: `function-orange` e `function-blue`. Você aloca 600 unidades de simultaneidade reservada para `function-orange`. Você aloca 200 unidades de simultaneidade provisionada para `function-blue`. Suponha que, com o tempo, você implante funções adicionais e observe o seguinte padrão de tráfego:



No diagrama anterior, as linhas pretas indicam o uso real da simultaneidade ao longo do tempo, e a linha vermelha indica o valor de `ClaimedAccountConcurrency` ao longo do tempo. Em todo esse cenário, `ClaimedAccountConcurrency` no mínimo é 800, apesar da baixa utilização real da simultaneidade nas funções. Isso ocorre porque você alocou o total de 800 unidades de simultaneidade para `function-orange` e `function-blue`. Da perspectiva do Lambda, você "reivindicou" essa simultaneidade para uso, portanto você efetivamente só tem 200 unidades de simultaneidade restantes para outras funções.

Para esse cenário, a simultaneidade alocada é 800 na fórmula `ClaimedAccountConcurrency`. Podemos, então, calcular o valor de `ClaimedAccountConcurrency` em vários pontos do diagrama:

- Em `t1`, `ClaimedAccountConcurrency` é 800 (800 + 0 `UnreservedConcurrentExecutions`).
- Em `t2`, `ClaimedAccountConcurrency` é 900 (800 + 100 `UnreservedConcurrentExecutions`).

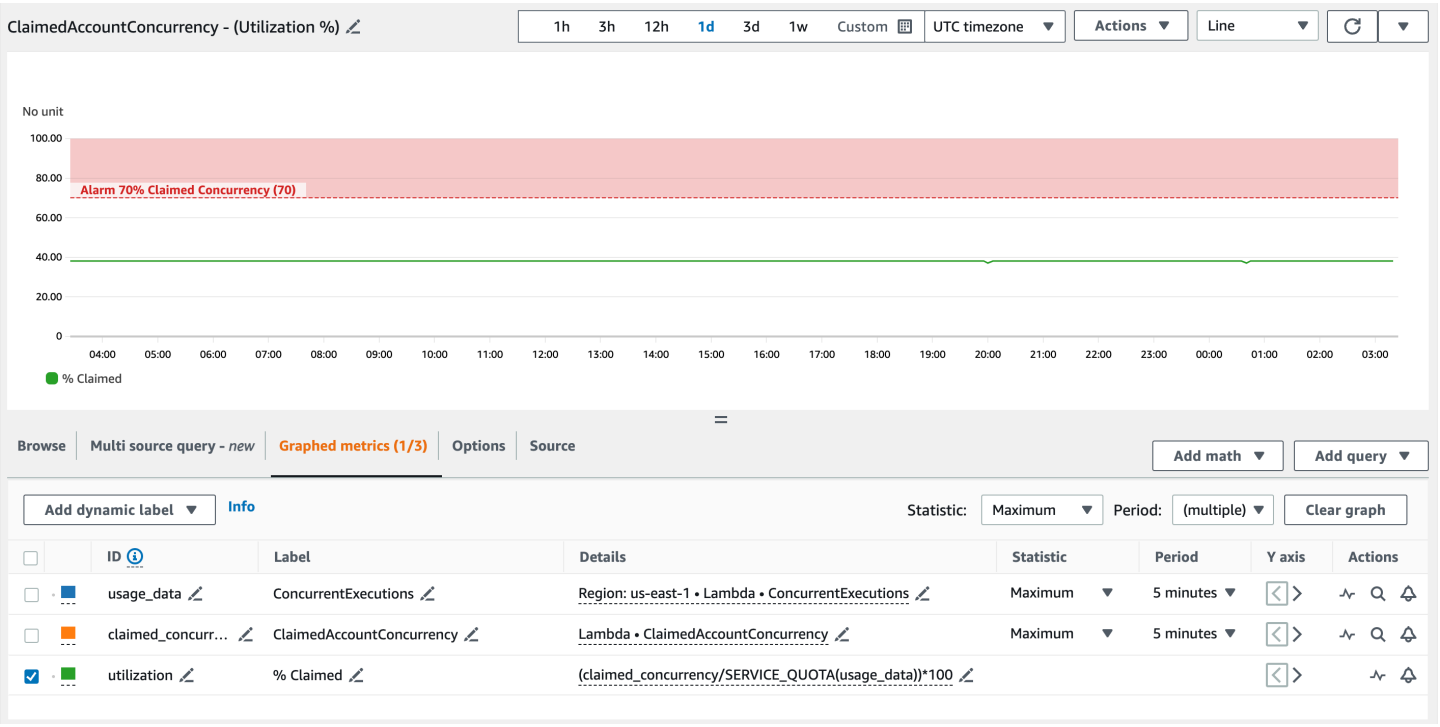
- Em `t3`, `ClaimedAccountConcurrency` é, novamente, 900 (800 + 100 `UnreservedConcurrentExecutions`).

Configurando a **`ClaimedAccountConcurrency`** métrica em CloudWatch

O Lambda emite a métrica em. `ClaimedAccountConcurrency` CloudWatch Use essa métrica com o valor de `SERVICE_QUOTA(ConcurrentExecutions)` para obter a utilização percentual da simultaneidade na conta, conforme mostrado na seguinte fórmula:

```
Utilization = (ClaimedAccountConcurrency/SERVICE_QUOTA(ConcurrentExecutions)) * 100%
```

A captura de tela a seguir ilustra como você pode representar graficamente essa fórmula. CloudWatch A linha verde `claim_utilization` representa a utilização da simultaneidade nessa conta, que está em torno de 40%:



A captura de tela anterior também inclui um CloudWatch alarme que entra em `ALARM` estado quando a utilização simultânea excede 70%. Você poderá usar a métrica `ClaimedAccountConcurrency` associada a alarmes semelhantes para determinar proativamente quando poderá ser preciso solicitar um limite maior de simultaneidade para a conta.

Configurar a assinatura de código para o AWS Lambda

A assinatura de código para AWS Lambda ajuda a garantir que apenas código confiável seja executado em suas funções do Lambda. Quando você habilita a assinatura de código para uma função, o Lambda verifica cada implantação de código e se o pacote de código é assinado por uma fonte confiável.

> **Note**
>
> Funções definidas como imagens de contêiner não são compatíveis com assinatura de código.

Para verificar a integridade do código, use o AWS Signer para criar pacotes de código assinados digitalmente para funções e camadas. Quando um usuário tenta implantar um pacote de código, o Lambda executa verificações de validação no pacote de código antes de aceitar a implantação. Como as verificações de validação de assinatura de código são executadas no momento da implantação, não há impacto na performance durante a execução da função.

Você também usa o AWS Signer para criar perfis de assinatura. Você usa um perfil de assinatura para criar o pacote de código assinado. Use o AWS Identity and Access Management (IAM) para controlar quem pode assinar pacotes de código e criar perfis de assinatura. Para obter mais informações, consulte Controle de acesso e autenticação no Guia do desenvolvedor do AWS Signer.

Para ativar a assinatura de código para uma função, crie uma configuração de assinatura de código e associe-a à função. Uma configuração de assinatura de código define uma lista de perfis de assinatura permitidos e a ação de política a ser executada se alguma das verificações de validação falhar.

As camadas do Lambda seguem o mesmo formato de pacote de código assinado que os pacotes de código de função. Quando você adiciona uma camada a uma função que tem a assinatura de código ativada, o Lambda verifica se a camada está assinada por um perfil de assinatura permitido. Quando você habilita a assinatura de código para uma função, todas as camadas adicionadas à função também devem ser assinadas por um dos perfis de assinatura permitidos.

Use o IAM para controlar quem pode criar configurações de assinatura de código. Normalmente, você permite que apenas usuários administrativos específicos tenham essa capacidade. Além disso, você pode configurar políticas do IAM para impor que os desenvolvedores criem apenas funções que tenham a assinatura de código ativada.

Você pode configurar a assinatura de código para registrar as alterações do AWS CloudTrail. Implantações bem-sucedidas e bloqueadas para funções são registradas no CloudTrail com informações sobre as verificações de assinatura e validação.

Você pode configurar a assinatura de código para suas funções usando o console do Lambda, a AWS Command Line Interface (AWS CLI), o AWS CloudFormation e o AWS Serverless Application Model (AWS SAM).

Não há custo adicional para usar o AWS Signer ou assinatura de código para o AWS Lambda.

Seções

Validação de assinatura

O Lambda executa as seguintes verificações de validação ao implantar um pacote de código assinado na sua função:

1. Integridade — valida se o pacote de código não foi modificado desde que foi assinado. O Lambda compara o hash do pacote com o hash da assinatura.
2. Expiração valida se a assinatura do pacote de código não expirou.
3. Incompatibilidade: valida se o pacote de código está assinado com um dos perfis de assinatura permitidos para a função do Lambda. Uma incompatibilidade também ocorre se uma assinatura não estiver presente.
4. Revogação: valida se a assinatura do pacote de código não foi revogada.

A política de validação de assinatura definida na configuração da assinatura de código determina qual das seguintes ações serão executadas pelo Lambda se alguma das verificações de validação falhar:

- Avisar — O Lambda permite a implantação do pacote de código, mas emite um aviso. O Lambda emite uma nova métrica do Amazon CloudWatch e também armazena o aviso no log do CloudTrail.
- Impor: o Lambda emite um aviso (o mesmo que para a ação Avisar) e bloqueia a implantação do pacote de código.

Você pode configurar a política para as verificações de validação de expiração, incompatibilidade e revogação. Observe que não é possível configurar uma política para a verificação de integridade. Se a verificação de integridade falhar, o Lambda bloqueia a implantação.

Pré-requisitos de configuração

Antes de configurar a assinatura de código para uma função do Lambda, use o AWS Signer para fazer o seguinte:

- Criar um ou mais perfis de assinatura.
- Usar um perfil de assinatura para criar um pacote de código assinado para sua função.

Para obter mais informações, consulte Criar perfis de assinatura (Console) no Guia do desenvolvedor doAWS Signer.

Criar configurações de assinatura de código

Uma configuração de assinatura de código define uma lista dos perfis de assinatura permitidos e a política de validação de assinatura.

Para criar uma configuração de assinatura de código (console)

1. Abra a página Code signing configurations (Configurações de assinatura de código) do console do Lambda.
2. Escolha Criar configuração.
3. Em Description (Descrição), insira um nome descritivo para a configuração.
4. Em Signing profiles (Perfis de assinatura), adicione até 20 perfis de assinatura à configuração.

a. Para Signing profile version ARN (ARN da versão do perfil de assinatura), escolha o nome de recurso da Amazon (ARN) de uma versão de perfil ou insira o ARN.

b. Para adicionar um perfil de assinatura adicional, escolha Add signing profiles (Adicionar perfis de assinatura).

5. Em Signature validation policy (Política de validação de assinatura), escolha Warn (Avisar) ou Enforce (Impor).
6. Escolha Criar configuração.

Atualizar uma configuração de assinatura de código

Quando você atualiza uma configuração de assinatura de código, as alterações afetam as implantações futuras de funções que têm a configuração de assinatura de código anexada.

Para atualizar uma configuração de assinatura de código (console)

1. Abra a página Code signing configurations (Configurações de assinatura de código) do console do Lambda.
2. Selecione uma configuração de assinatura de código a ser atualizada e escolha Edit (Editar).
3. Em Description (Descrição), insira um nome descritivo para a configuração.
4. Em Signing profiles (Perfis de assinatura), adicione até 20 perfis de assinatura à configuração.

   a. Para Signing profile version ARN (ARN da versão do perfil de assinatura), escolha o nome de recurso da Amazon (ARN) de uma versão de perfil ou insira o ARN.

   b. Para adicionar um perfil de assinatura adicional, escolha Add signing profiles (Adicionar perfis de assinatura).

5. Em Signature validation policy (Política de validação de assinatura), escolha Warn (Avisar) ou Enforce (Impor).
6. Escolha Salvar alterações.

Excluir uma configuração de assinatura de código

Uma configuração de assinatura de código pode ser excluída somente se nenhuma função estiver usando-a.

Para excluir uma configuração de assinatura de código (console)

1. Abra a página Code signing configurations (Configurações de assinatura de código) do console do Lambda.
2. Selecione uma configuração de assinatura de código a ser excluída e escolha Delete (Excluir).
3. Para confirmar, escolha Delete (Excluir) novamente.

Habilitar a assinatura de código para uma função

Para habilitar a assinatura de código para uma função, associe uma configuração de assinatura de código à função.

Para associar uma configuração de assinatura de código a uma função (console)

1. Abra a página Funções do console do Lambda.
2. Escolha a função para a qual deseja ativar a assinatura de código.
3. Abra a guia Configuration (Configuração).
4. Role para baixo e escolha Assinatura de código.
5. Selecione a opção Editar.
6. Em Edit code signing (Editar assinatura de código), escolha uma configuração de assinatura de código para esta função.
7. Escolha Salvar.

Configurar políticas do IAM

Para conceder permissão a um usuário para acessar as operações da API de assinatura de código, anexe uma ou mais instruções de política à política do usuário. Para obter mais informações sobre políticas de usuário, consulte Trabalhando com políticas do IAM baseadas em identidade no Lambda.

A instrução da política de exemplo a seguir concede permissão para criar, atualizar e recuperar configurações de assinatura de código.

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "lambda:CreateCodeSigningConfig",
          "lambda:UpdateCodeSigningConfig",
          "lambda:GetCodeSigningConfig"
        ],
      "Resource": "*"
    }
  ]
}
```

Os administradores podem usar a chave de condição `CodeSigningConfigArn` para especificar as configurações de assinatura de código que os desenvolvedores devem usar para criar ou atualizar suas funções.

O exemplo de declaração de política a seguir concede permissão para criar uma função. A declaração de política inclui um`lambda:CodeSigningConfigArn`para especificar a configuração de assinatura de código permitida. O Lambda bloqueia qualquer`CreateFunction`Solicitação de API se`CodeSigningConfigArn`está ausente ou não corresponde ao valor na condição.

```
{
  "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”
          }
      }
    }
  ]
}
```

Configurar assinatura de código com a API do Lambda

Para gerenciar configurações de assinatura de código com a AWS CLI ou o AWS SDK ou, use as seguintes operações de API:

- ListCodeSigningConfigs
- CreateCodeSigningConfig
- GetCodeSigningConfig
- UpdateCodeSigningConfig
- DeleteCodeSigningConfig

Para gerenciar a configuração de assinatura de código de uma função, use as seguintes operações de API:

- CreateFunction
- GetFunctionCodeSigningConfig
- PutFunctionCodeSigningConfig
- DeleteFunctionCodeSigningConfig
- ListFunctionsByCodeSigningConfig

Utilizar etiquetas em funções do Lambda

Você pode marcar funções do AWS Lambda para ativar o controle de acesso por atributo (ABAC) e organizá-las por proprietário, projeto ou departamento. Etiquetas são pares de chave/valor de formato livre, compatíveis com todos os serviços da AWS usados no ABAC, filtragem de recursos e adição de detalhes a relatórios de faturamento.

As tags se aplicam no nível da função, não em versões ou aliases. Etiquetas não fazem parte da configuração específica da versão da qual o Lambda cria um snapshot quando você publica uma versão.

Seções

Permissões necessárias para trabalhar com tags

Conceda permissões apropriadas à identidade AWS Identity and Access Management (IAM) (usuário, grupo ou função) para a pessoa que trabalha com a função:

- lambda: ListTags — Quando uma função tem tags, conceda essa permissão a qualquer pessoa que precise chamá-la `GetFunction` ou `ListTags` usá-la.
- lambda: TagResource — Conceda essa permissão a qualquer pessoa que precise ligar `CreateFunction` ou`TagResource`.

Para ter mais informações, consulte Trabalhando com políticas do IAM baseadas em identidade no Lambda.

Uso de tags usando o console do Lambda

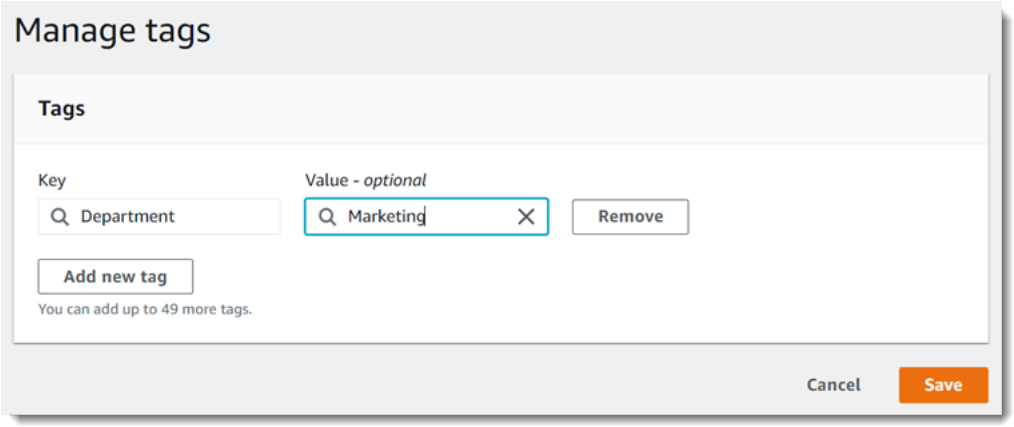
Você pode usar o console do Lambda para criar funções que têm etiquetas, adicionar etiquetas a funções existentes e filtrar funções pelas etiquetas adicionadas.

Para adicionar etiquetas ao criar uma função

1. Abra a página Funções do console do Lambda.
2. Escolha a opção Criar função.
3. Escolha Author from scratch (Criar do zero) ou Container image (Imagem de contêiner).
4. Em Basic information (Informações básicas), faça o seguinte:
   a. Em Function name (Nome da função), insira o nome da função. Os nomes das funções têm um limite de 64 caracteres de comprimento.
   b. Para o Runtime, escolha a versão do idioma a ser usada para sua função.
   c. (Opcional) Em Architecture (Arquitetura), escolha a arquitetura do conjunto de instruções a ser usada para sua função. O valor da arquitetura padrão é X86_64. Ao criar o pacote de implantação para sua função, verifique se ele é compatível com a arquitetura do conjunto de instruções que você escolheu.
5. Expanda Advanced settings (Configurações avançadas) e selecione Enable tags (Habilitar etiquetas).
6. Escolha Add new tag (Adicionar nova etiqueta) e insira Key (Chave) e uma opção de Value (Valor) opcional. Para adicionar mais tags, repita esta etapa.
7. Escolha a opção Criar função.

Para adicionar etiquetas a uma função existente

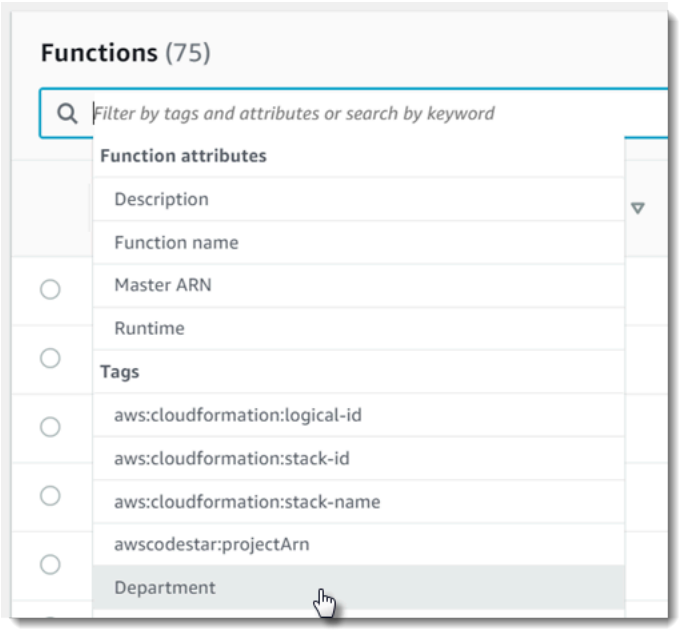
1. Abra a página Funções do console do Lambda.
2. Escolha o nome de uma função.
3. Escolha Configuration (Configuração) e depois Tags (Etiquetas).
4. Em Tags, selecione Gerenciar tags.
5. Escolha Add new tag (Adicionar nova etiqueta) e insira Key (Chave) e uma opção de Value (Valor) opcional. Para adicionar mais tags, repita esta etapa.



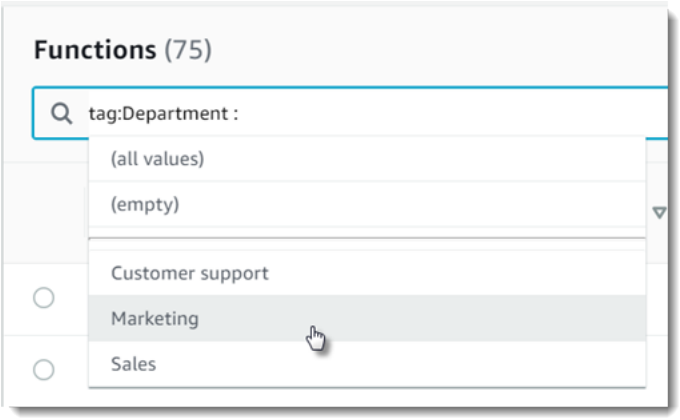
6. Escolha Salvar.

Como filtrar funções com tags

1. Abra a página Funções do console do Lambda.
2. Escolha a barra de pesquisa para ver uma lista de atributos de função e chaves de tag.



3. Selecione uma chave de tag para ver uma lista de valores que estão sendo usados na região da AWS atual.
4. Escolha um valor para ver funções correspondentes ou selecione (all values) (todos os valores) para ver todas as funções que têm uma tag com essa chave.



A barra de pesquisa também oferece suporte para a pesquisa de chaves de tag. Insira `tag` para ver somente uma lista de chaves de tag ou insira o nome de uma chave para encontrá-la na lista.

Uso de tags com a AWS CLI

Adicionar e remover etiquetas

Para criar uma nova função do Lambda com etiquetas, use o comando create-function com a opção --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
```

Para adicionar tags a uma função existente, use o comando tag-resource.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:function:my-function \
--tags Department=Marketing,CostCenter=1234ABCD
```

Para remover tags, use o comando untag-resource.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:function:my-
function \
--tag-keys Department
```

Visualização de tags em uma função

Se você deseja visualizar as etiquetas que são aplicadas à uma função do Lambda específica, você pode usar os seguintes comandos da AWS CLI:

- ListTags— Para ver uma lista das tags associadas a essa função, inclua o ARN (Amazon Resource Name) da função Lambda:

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:function:my-function
```

- GetFunction— Para ver uma lista das tags associadas a essa função, inclua o nome da função Lambda:

```
aws lambda get-function --function-name my-function
```

Filtragem de funções por tag

Você pode usar a operação AWS Resource Groups Tagging API GetResourcesda API para filtrar seus recursos por tags. A operação `GetResources` aceita até 10 filtros, cada um contendo uma chave de etiquetas e até 10 valores de etiquetas. Você fornece a `GetResources` um `ResourceType` para filtrar por tipos de recursos específicos.

Para obter mais informações sobre o AWS Resource Groups, consulte O que são grupos de recursos?, no Guia do usuário de AWS Resource Groups e etiquetas.

Requisitos de tags

Os seguintes requisitos são aplicáveis às tags:

- Número máximo de tags por recurso: 50
- Comprimento máximo da chave: 128 caracteres Unicode em UTF-8
- Valor máximo da chave: 256 caracteres Unicode em UTF-8
- As chaves e valores das tags diferenciam maiúsculas de minúsculas.
- Não use o prefixo `aws:` no nome nem no valor de suas tags, pois ele é reservado para uso da AWS. Você não pode editar nem excluir nomes ou valores de tag com esse prefixo. As tags com esse prefixo não contam para as tags por limite de recurso.

- Se você planeja usar o esquema de etiquetas em vários serviços e recursos, lembre-se de que outros serviços podem ter outras restrições sobre os caracteres permitidos. No geral, os caracteres permitidos são letras, espaços e números representáveis em UTF-8, além dos seguintes caracteres especiais: + - = . _ : / @.

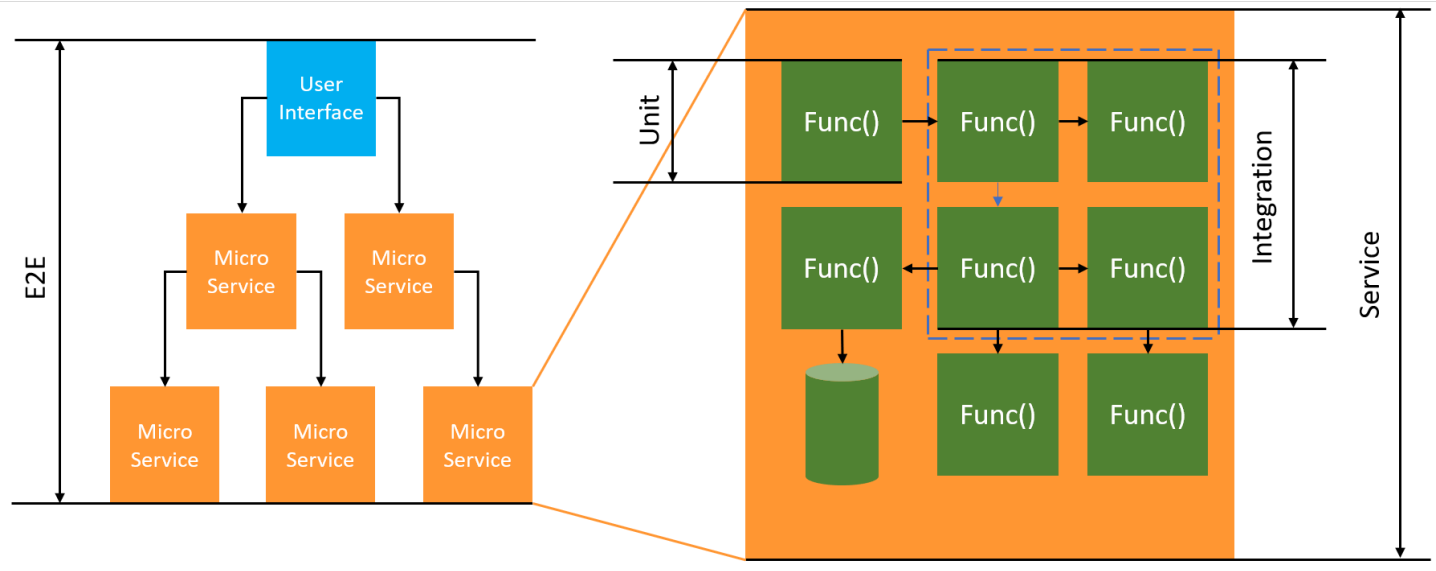
Como testar funções e aplicações com tecnologia sem servidor

Os testes de funções com tecnologia sem servidor usam tipos e técnicas de teste tradicionais, mas você também deve considerar testar aplicações com tecnologia sem servidor como um todo. Os testes baseados em nuvem fornecerão a medida mais precisa da qualidade das suas funções e aplicações com tecnologia sem servidor.

Uma arquitetura de aplicações com tecnologia sem servidor inclui serviços gerenciados que fornecem funcionalidade de aplicações críticas por meio de chamadas de API. Por esse motivo, seu ciclo de desenvolvimento deve incluir testes automatizados que verifiquem a funcionalidade quando sua função e seus serviços interagem.

Se você não criar testes baseados em nuvem, poderá encontrar problemas devido às diferenças entre o ambiente local e o ambiente implantado. Seu processo de integração contínua deve executar testes em comparação com um conjunto de recursos provisionados na nuvem antes de promover seu código para o próximo ambiente de implantação, como GQ, Preparação ou Produção.

Continue lendo este breve guia para aprender sobre estratégias de testes para aplicações com tecnologia sem servidor ou acesse o repositório de exemplos de testes com tecnologia sem servidor para conhecer exemplos práticos, específicos da linguagem e do runtime escolhidos.



Para testes com tecnologia sem servidor, você ainda escreverá testes unitários, de integração e de ponta a ponta.

- Testes unitários: testes executados em comparação com um bloco de código isolado. Por exemplo, verificar a lógica de negócios para calcular a taxa de entrega de acordo com um item e um destino determinado.
- Testes de integração: testes envolvendo dois ou mais componentes ou serviços que interagem, normalmente em um ambiente de nuvem. Por exemplo, a verificação de eventos de processos de uma função em uma fila.
- Testes de ponta a ponta: testes que verificam o comportamento em toda a aplicação. Por exemplo, garantir que a infraestrutura seja configurada corretamente e que os eventos fluam entre os serviços conforme o esperado para registrar o pedido de um cliente.

Resultados de negócios direcionados

Testar soluções com tecnologia sem servidor pode exigir um pouco mais de tempo para configurar testes que verifiquem as interações orientadas por eventos entre os serviços. Tenha em mente as seguintes questões práticas de negócios ao ler este guia:

- Aumentar a qualidade da sua aplicação
- Diminuir o tempo para criar recursos e corrigir bugs

A qualidade de uma aplicação depende do teste de vários cenários para verificar a funcionalidade. Considerar cuidadosamente os cenários de negócios e automatizar esses testes para serem executados em comparação com serviços em nuvem aumentará a qualidade da sua aplicação.

Bugs de software e problemas de configuração têm o menor impacto no custo e na programação quando detectados durante um ciclo de desenvolvimento iterativo. Se os problemas não forem detectados durante o desenvolvimento, a descoberta e a correção em produção exigirão mais esforços de mais pessoas.

Uma estratégia de teste com tecnologia sem servidor bem planejada aumentará a qualidade do software e melhorará o tempo de iteração ao verificar a performance das suas aplicações e funções do Lambda, conforme esperado em um ambiente de nuvem.

O que testar

Recomendamos a adoção de uma estratégia de testes que teste os comportamentos dos serviços gerenciados, a configuração da nuvem, as políticas de segurança e a integração com seu código para melhorar a qualidade do software. Os testes de comportamento, também conhecidos como

testes da caixa preta, verificam se um sistema funciona conforme o esperado sem o conhecimento de todos os componentes internos.

- Execute testes unitários para verificar a lógica de negócios dentro das funções do Lambda.
- Verifique se os serviços integrados foram realmente invocados e se os parâmetros de entrada estão corretos.
- Verifique se um evento passa por todos os serviços esperados de ponta a ponta em um fluxo de trabalho.

Na arquitetura tradicional baseada em servidor, as equipes frequentemente definem um escopo de testes para incluir somente o código executado no servidor de aplicações. Outros componentes, serviços ou dependências são frequentemente considerados externos e fora do escopo dos testes.

As aplicações com tecnologia sem servidor geralmente consistem em pequenas unidades de trabalho, como funções do Lambda que recuperam produtos de um banco de dados, processam itens de uma fila ou redimensionam uma imagem no armazenamento. Cada componente é executado no seu próprio ambiente. As equipes provavelmente serão responsáveis por muitas dessas pequenas unidades em uma única aplicação.

Algumas funcionalidades da aplicação podem ser delegadas inteiramente para serviços gerenciados, como o Amazon S3, ou criadas sem o uso de qualquer código desenvolvido internamente. Não há necessidade de testar esses serviços gerenciados, mas é necessário testar a integração com esses serviços.

Como testar com tecnologia sem servidor

Você provavelmente está familiarizado com a forma de testar aplicações implantadas localmente: você escreve testes que são executados em comparação com o código executado inteiramente no sistema operacional da área de trabalho ou dentro de contêineres. Por exemplo, você pode invocar um componente de serviço Web local com uma solicitação e, em seguida, fazer declarações sobre a resposta.

As soluções com tecnologia sem servidor são criadas com base no seu código de função e em serviços gerenciados baseados em nuvem, como filas, bancos de dados, barramentos de eventos e sistemas de mensagens. Esses componentes são todos conectados por meio de uma arquitetura orientada por eventos, na qual as mensagens, chamadas de eventos, fluem de um recurso para outro. Essas interações podem ser síncronas, como quando um serviço Web retorna

resultados imediatamente, ou uma ação assíncrona que é concluída posteriormente, como colocar itens em uma fila ou iniciar uma etapa do fluxo de trabalho. Sua estratégia de testes deve incluir ambos os cenários e testar as interações entre os serviços. Para interações assíncronas, talvez seja necessário detectar efeitos colaterais em componentes downstream que podem não ser imediatamente observáveis.

Replicar todo um ambiente de nuvem, incluindo filas, tabelas de banco de dados, barramentos de eventos, políticas de segurança e muito mais, não é prático. Você inevitavelmente encontrará problemas devido às diferenças entre seu ambiente local e seus ambientes implantados na nuvem. As variações entre seus ambientes aumentarão o tempo de reprodução e correção de bugs.

Em aplicações com tecnologia sem servidor, os componentes da arquitetura geralmente existem inteiramente na nuvem. Portanto, são necessários testes com códigos e serviços na nuvem para desenvolver recursos e corrigir bugs.

Técnicas de teste

Na realidade, sua estratégia de testes provavelmente incluirá uma combinação de técnicas para aumentar a qualidade das suas soluções. Você usará testes interativos rápidos para depurar funções no console, testes de unidade automatizados para verificar a lógica de negócios isolada, verificação de chamadas para serviços externos com simulações e testes ocasionais em comparação com emuladores que imitam um serviço.

- Testes na nuvem: você implanta infraestrutura e código para testar com serviços reais, políticas de segurança, configurações e parâmetros específicos da infraestrutura. Os testes baseados em nuvem fornecem a medida mais precisa da qualidade do seu código.

  Depurar uma função no console é uma forma rápida de testar na nuvem. Você pode escolher entre uma biblioteca de eventos de teste de exemplo ou criar um evento personalizado para testar uma função isoladamente. Você também pode compartilhar eventos de teste por meio do console com sua equipe.

  Para automatizar os testes no ciclo de vida de desenvolvimento e construção, você precisará testar fora do console. Consulte as seções de testes específicos da linguagem neste guia para obter estratégias e recursos de automação.
- Testar com simulações (também chamadas de imitações): as simulações são objetos no seu código que simulam e substituem um serviço externo. As simulações fornecem um comportamento predefinido para verificar as chamadas e parâmetros de serviços. Uma imitação é a implementação de uma simulação que usa atalhos para simplificar ou melhorar a performance.

Por exemplo, a imitação de um objeto de acesso a dados pode retornar dados de um datastore na memória. As simulações podem imitar e simplificar dependências complexas, mas também podem levar a mais simulações para substituir dependências aninhadas.

- Testar com emuladores: você pode configurar aplicações (às vezes, de terceiros) para imitar um serviço de nuvem no seu ambiente local. Velocidade é o ponto forte deles, mas a configuração e a paridade com os serviços de produção são o ponto fraco. Use emuladores com moderação.

Testes na nuvem

Os testes na nuvem são valiosos para todas as fases dos testes, incluindo testes unitários, testes de integração e testes de ponta a ponta. Quando você executa testes comparados com um código baseado na nuvem que também interage com serviços baseados na nuvem, você obtém a medida mais precisa da qualidade do seu código.

Uma maneira conveniente de executar uma função do Lambda na nuvem é por meio de um evento de teste no AWS Management Console. Um evento de teste é uma entrada JSON para sua função. Se a função não necessitar de entrada, o evento poderá ser um documento JSON vazio (`{}`). O console fornece eventos de exemplo para uma variedade de integrações de serviços. Depois de criar um evento no console, você também pode compartilhá-lo com sua equipe para tornar os testes mais fáceis e consistentes.

Saiba como depurar uma função de exemplo no console.

> **Note**
>
> Embora executar funções no console seja uma maneira rápida de depurar, a automatização dos ciclos de teste é essencial para aumentar a qualidade e a velocidade de desenvolvimento da aplicação.

Exemplos de automação de testes estão disponíveis no repositório de exemplos de testes com tecnologia sem servidor. A linha de comando a seguir executa um exemplo de teste de integração automatizado no Python:

```
python -m pytest -s tests/integration -v
```

Embora o teste seja executado localmente, ele interage com recursos baseados na nuvem. Esses recursos foram implantados usando a ferramenta da linha de comando do AWS SAM e AWS

Serverless Application Model. O código de teste primeiro recupera as saídas da pilha implantada, que inclui o endpoint da API, o ARN da função e a função de segurança. Em seguida, o teste envia uma solicitação ao endpoint da API, que responde com uma lista de buckets do Amazon S3. Esse teste é executado inteiramente com recursos baseados na nuvem para verificar se esses recursos estão implantados e protegidos e se funcionam conforme esperado.

```
========================= 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 ==========================
```

Para o desenvolvimento de aplicações nativas de nuvem, os testes na nuvem oferecem os seguintes benefícios:

- Você pode testar todos os serviços disponíveis.
- Você está sempre usando as APIs de serviço e os valores de retorno mais recentes.
- Um ambiente de teste na nuvem se assemelha muito ao seu ambiente de produção.
- Os testes podem abranger políticas de segurança, cotas de serviço, configurações e parâmetros específicos da infraestrutura.
- Cada desenvolvedor pode criar rapidamente um ou mais ambientes de teste na nuvem.
- Os testes na nuvem aumentam a confiança de que seu código será executado corretamente na produção.

Os testes na nuvem têm algumas desvantagens. O aspecto negativo mais óbvio dos testes na nuvem é que as implantações em ambientes de nuvem geralmente demoram mais do que as implantações em ambientes de área de trabalho locais.

Ferramentas, como o AWS Serverless Application Model (AWS SAM) Accelerate, o modo de observação do kit de desenvolvimento em nuvem (CDK da AWS) e o SST (terceiros) reduzem a latência envolvida nas iterações de implantação na nuvem. Essas ferramentas podem monitorar sua infraestrutura e seu código e implantar automaticamente atualizações incrementais no seu ambiente de nuvem.

> **Note**
>
> Veja como criar infraestrutura como código no Guia do desenvolvedor de tecnologia sem servidor para saber mais sobre o AWS Serverless Application Model, AWS CloudFormation e AWS Cloud Development Kit (AWS CDK).

Diferentemente dos testes locais, os testes na nuvem exigem recursos adicionais que podem gerar custos de serviço. A criação de ambientes de teste isolados pode aumentar a carga sobre suas equipes de DevOps, especialmente em organizações com controles rígidos sobre contas e infraestrutura. Mesmo assim, ao trabalhar com cenários de infraestrutura complexos, o custo do tempo do desenvolvedor para configurar e manter um ambiente local complexo pode ser semelhante (ou mais caro) ao de usar ambientes de teste descartáveis criados com ferramentas de automação de infraestrutura como código.

Testar na nuvem, mesmo com essas considerações, ainda é a melhor maneira de garantir a qualidade das suas soluções com tecnologia sem servidor.

Testar com simulações

Testar com simulações é uma técnica em que você cria objetos de substituição no seu código para simular o comportamento de um serviço em nuvem.

Por exemplo, você pode escrever um teste que usa uma simulação do serviço do Amazon S3 que retorna uma resposta específica sempre que o método CreateObject é chamado. Quando um teste é executado, a simulação retorna essa resposta programada sem chamar o Amazon S3 ou qualquer outro endpoint de serviço.

Objetos simulados são frequentemente gerados por uma estrutura simulada para reduzir o esforço de desenvolvimento. Algumas estruturas simuladas são genéricas e outras são criadas especificamente para AWS SDKs, como o Moto, uma biblioteca Python para simular serviços e recursos da AWS.

Observe que os objetos simulados diferem dos emuladores porque os simuladores normalmente são criados ou configurados por um desenvolvedor como parte do código de teste, enquanto os emuladores são aplicações autônomas que expõem a funcionalidade da mesma maneira que os sistemas que elas emulam.

As vantagens de usar simulações incluem o seguinte:

- Elas podem simular serviços de terceiros que estão além do controle da aplicação, como APIs e provedores de software como serviço (SaaS), sem precisar de acesso direto a esses serviços.
- As simulações são úteis para testar condições de falha, especialmente quando essas condições são difíceis de simular, como uma interrupção do serviço.
- A simulação pode fornecer testes locais rápidos depois de configurada.
- As simulações podem fornecer um comportamento substituto para praticamente qualquer tipo de objeto. Portanto, as estratégias de simulação podem criar cobertura para uma variedade mais ampla de serviços do que os emuladores.
- Quando novos recursos ou comportamentos são disponibilizados, os testes com simulações podem reagir mais rapidamente. Ao usar uma estrutura simulada genérica, você poderá simular novos recursos assim que o AWS SDK atualizado estiver disponível.

O teste com simulação tem as seguintes desvantagens:

- As simulações geralmente exigem uma quantidade significativa de esforços de definição e configuração, especificamente na tentativa de determinar valores de retorno de diferentes serviços para simular respostas adequadamente.
- As simulações são escritas, configuradas e devem ser mantidas pelos desenvolvedores, aumentando suas responsabilidades.
- Talvez você precise ter acesso à nuvem para entender as APIs e os valores de retorno dos serviços.
- As simulações podem ser difíceis de manter. Quando as assinaturas simuladas da API de nuvem mudam ou os esquemas de valor de retorno evoluem, é preciso atualizar as simulações. As simulações também exigem atualizações se você estender a lógica da aplicação para fazer chamadas para novas APIs.
- Os testes que usam simulações podem passar em ambientes de área de trabalho, mas falhar na nuvem. Os resultados podem não corresponder à API atual. A configuração e as cotas do serviço não podem ser testadas.
- As estruturas simuladas se limitam a testar ou detectar uma política do AWS Identity and Access Management (IAM). Embora as simulações sejam melhores para simular quando a autorização falha ou uma cota é excedida, os testes não podem determinar que resultado realmente ocorrerá em um ambiente de produção.

Testar com emulação

Os emuladores normalmente são uma aplicação executada localmente que imita um serviço da AWS de produção.

Os emuladores têm APIs semelhantes às APIs equivalentes da nuvem e fornecem valores de retorno semelhantes. Eles também podem simular mudanças de estado iniciadas por chamadas de API. Por exemplo, você pode usar o AWS SAM para executar uma função com o AWS SAM local para emular o serviço do Lambda para poder invocar rapidamente uma função. Consulte o AWS SAM local no Guia do desenvolvedor do AWS Serverless Application Model para obter detalhes.

As vantagens do teste com emuladores incluem o seguinte:

- Os emuladores podem facilitar iterações e testes rápidos de desenvolvimento local.
- Os emuladores fornecem um ambiente familiar para desenvolvedores acostumados a desenvolver código em um ambiente local. Por exemplo, se você estiver familiarizado com o desenvolvimento de uma aplicação de n camadas, talvez tenha um mecanismo de banco de dados e um servidor Web, semelhantes aos executados em produção na sua máquina local para fornecer uma capacidade de teste rápida, local e isolada.

- Os emuladores não exigem qualquer alteração na infraestrutura de nuvem (como contas de desenvolvedores na nuvem). Por isso, é fácil implementar com os padrões de teste existentes.

Testar com emuladores tem as seguintes desvantagens:

- Os emuladores podem ser difíceis de configurar e replicar, especialmente quando usados em pipelines de CI/CD. Isso pode aumentar a workload da equipe de TI ou dos desenvolvedores que gerenciam seu próprio software.
- Os recursos e APIs emulados geralmente ficam atrasados em relação às atualizações do serviço. Isso pode causar erros porque o código testado não corresponde à API real e impede a adoção de novos recursos.
- Os emuladores precisam de suporte, atualizações, correções de bugs e aprimoramentos de paridade de recursos. Eles são de responsabilidade do autor do emulador, que pode ser uma empresa externa.
- Os testes que dependem de emuladores podem fornecer resultados bem-sucedidos localmente, mas falham na nuvem devido a políticas de segurança de produção, configurações entre serviços ou excesso de cotas do Lambda.
- Muitos serviços da AWS não têm emuladores disponíveis. Se você depende da emulação, talvez não tenha uma opção de teste satisfatória para partes da sua aplicação.

Práticas recomendadas

As seções a seguir fornecem recomendações para testes bem-sucedidos de aplicações com tecnologia sem servidor.

Você pode encontrar exemplos práticos de testes e automação de testes no repositório de exemplos de testes com tecnologia sem servidor.

Priorize os testes na nuvem

Os testes na nuvem fornecem a cobertura de teste mais confiável, precisa e completa. A realização de testes no contexto da nuvem testará de forma abrangente não apenas a lógica de negócios, mas também as políticas de segurança, as configurações de serviços, as cotas, as assinaturas de API e os valores de retorno mais atualizados.

Estruture seu código quanto à capacidade de teste

Simplifique seus testes e funções do Lambda separando o código específico do Lambda da sua principal lógica de negócios.

Seu manipulador de funções do Lambda deve ser um pequeno adaptador que absorve dados de eventos e transmite somente os detalhes importantes para seus métodos de lógica de negócios. Com essa estratégia, você pode realizar testes abrangentes na sua lógica de negócios sem se preocupar com detalhes específicos do Lambda. Suas funções do AWS Lambda não devem exigir a configuração de um ambiente complexo ou uma grande quantidade de dependências para criar e inicializar o componente em teste.

De um modo geral, você deve escrever um manipulador que extraia e valide dados do evento de entrada e dos objetos de contexto e, em seguida, envie essa entrada aos métodos que executam sua lógica de negócios.

Acelerar loops de feedback de desenvolvimento

Existem ferramentas e técnicas para acelerar os loops de feedback de desenvolvimento. Por exemplo, o AWS SAM Accelerate e o modo de observação do AWS CDK diminuem o tempo necessário para atualizar os ambientes de nuvem.

Os exemplos no repositório de exemplos de testes com tecnologia sem servidor do GitHub exploram algumas dessas técnicas.

Também recomendamos que você crie e teste os recursos de nuvem o mais cedo possível durante o desenvolvimento, não somente após um registro no controle de origem. Essa prática permite uma exploração e experimentação mais rápidas no desenvolvimento de soluções. Além disso, automatizar a implantação em uma máquina de desenvolvimento ajuda a descobrir problemas de configuração da nuvem com mais rapidez e reduz o esforço desperdiçado em atualizações e processos de revisão de código.

Foco nos testes de integração

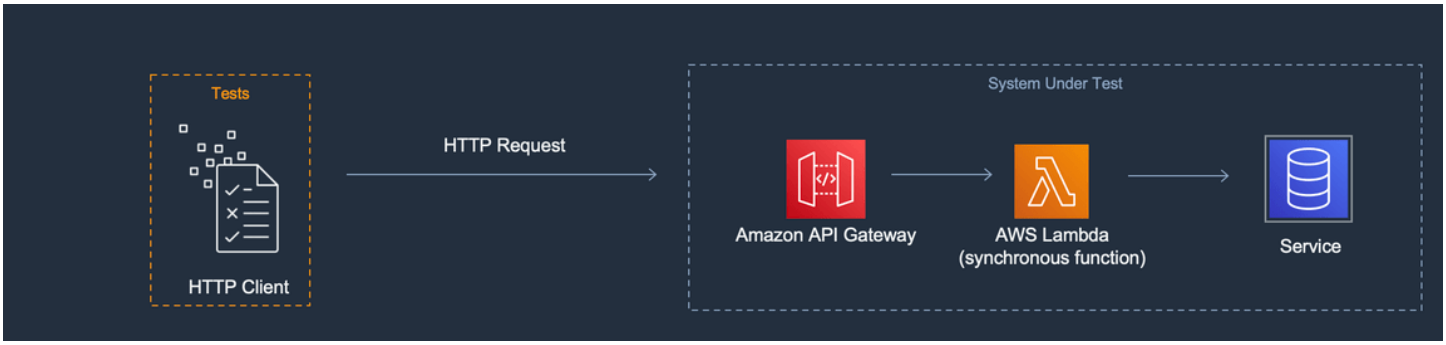
Ao criar aplicações com o Lambda, testar os componentes juntos é a melhor prática.

Os testes executados em comparação com dois ou mais componentes arquitetônicos são chamados de testes de integração. O objetivo dos testes de integração é entender não apenas como seu código será executado em todos os componentes, mas também como o ambiente que hospeda seu código

se comportará. Testes de ponta a ponta são tipos especiais de testes de integração que verificam o comportamento em toda uma aplicação.

Para criar testes de integração, implante sua aplicação em um ambiente de nuvem. Isso pode ser feito em um ambiente local ou por meio de um pipeline de CI/CD. Em seguida, escreva testes para exercitar o sistema em teste (SUT) e validar o comportamento esperado.

Por exemplo, o sistema em teste pode ser uma aplicação que usa o API Gateway, Lambda e DynamoDB. Um teste pode fazer uma chamada HTTP sintética para um endpoint do API Gateway e validar se a resposta incluiu a carga útil esperada. Esse teste valida se o código do AWS Lambda está correto e se cada serviço está configurado corretamente para processar a solicitação, incluindo as permissões do IAM entre eles. Além disso, você pode criar o teste para gravar registros de vários tamanhos para verificar se suas cotas de serviço, como o tamanho máximo do registro no DynamoDB, estão configuradas corretamente.



Criar ambientes de teste isolados

Os testes na nuvem normalmente exigem ambientes de desenvolvedores isolados para que não haja sobreposição de testes, dados e eventos.

Uma abordagem é fornecer a cada desenvolvedor uma conta da AWS dedicada. Isso evitará conflitos com a nomenclatura de recursos que podem ocorrer quando vários desenvolvedores trabalhando em uma base de código compartilhada tentarem implantar recursos ou invocar uma API.

Os processos de teste automatizados devem criar recursos com nomes exclusivos para cada pilha. Por exemplo, você pode configurar scripts ou arquivos de configuração TOML para que os comandos sam deploy ou sam sync da CLI do AWS SAM especifiquem automaticamente uma pilha com um prefixo exclusivo.

Em alguns casos, os desenvolvedores compartilham uma conta da AWS. Isso pode ocorrer devido à existência de recursos em sua pilha que são caros para operar ou de provisionar e configurar. Por

exemplo, um banco de dados pode ser compartilhado para facilitar a configuração e a propagação adequada dos dados.

Se os desenvolvedores compartilharem uma conta, você deverá definir limites para identificar a propriedade e eliminar a sobreposição. Uma maneira de fazer isso é prefixar os nomes das pilhas com os IDs de usuário dos desenvolvedores. Outra abordagem popular é configurar pilhas com base em ramificações de código. Com os limites das ramificações, os ambientes são isolados, mas os desenvolvedores ainda podem compartilhar recursos, como um banco de dados relacional. Essa abordagem é a melhor prática quando os desenvolvedores trabalham em mais de uma ramificação ao mesmo tempo.

Os testes na nuvem são valiosos para todas as fases dos testes, incluindo testes unitários, testes de integração e testes de ponta a ponta. Manter o isolamento adequado é essencial. Porém, você ainda quer que seu ambiente de GQ seja o mais parecido possível com seu ambiente de produção. Por esse motivo, as equipes adicionam processos de controle de mudanças para ambientes de GQ.

Para ambientes de pré-produção e de produção, geralmente são estabelecidos limites no nível da conta para isolar as workloads dos problemas de vizinho barulhento e implementar controles de segurança com privilégio mínimo para proteger dados confidenciais. As workloads têm cotas. Você não quer que seus testes consumam cotas alocadas para produção (vizinho barulhento) ou tenham acesso aos dados do cliente. O teste de carga é outra atividade que você deve isolar da sua pilha de produção.

Em todos os casos, os ambientes devem ser configurados com alertas e controles para evitar gastos desnecessários. Por exemplo, você pode limitar o tipo, a camada ou o tamanho dos recursos que podem ser criados e configurar alertas por e-mail quando os custos estimados excederem um determinado limite.

Use simulações para lógica de negócios isolada

As estruturas simuladas são uma ferramenta valiosa para escrever testes unitários rápidos. Elas são especialmente benéficas quando os testes abrangem lógicas de negócios internas complexas, como cálculos ou simulações matemáticas ou financeiras. Procure testes unitários que tenham um grande número de casos de teste ou variações de entrada, nos quais essas entradas não alterem o padrão ou o conteúdo das chamadas para outros serviços de nuvem.

O código coberto por testes unitários com simulações também deve ser abordado por testes na nuvem. Isso é recomendado porque o laptop de um desenvolvedor ou o ambiente de uma máquina de criação pode ser configurado de forma diferente de um ambiente de produção na nuvem. Por exemplo, suas funções do Lambda podem usar mais memória ou tempo do que o alocado quando

executadas com determinados parâmetros de entrada. Ou seu código pode incluir variáveis de ambiente que não estão configuradas da mesma forma (ou não estão configuradas) e as diferenças podem fazer com que o código se comporte de forma diferente ou apresente falha.

O benefício das simulações é menor para testes de integração porque o nível de esforço para implementar as simulações necessárias aumenta com o número de pontos de conexão. Os testes de ponta a ponta não devem usar simulações porque eles geralmente lidam com estados e lógica complexa que não podem ser facilmente simulados com estruturas de simulação.

Por fim, evite usar serviços de nuvem simulados para validar a implementação adequada das chamadas de serviço. Em vez disso, faça chamadas de serviço de nuvem na nuvem para validar o comportamento, a configuração e a implementação funcional.

Usar emuladores com moderação

Os emuladores podem ser convenientes em alguns casos de uso, por exemplo, para uma equipe de desenvolvimento com acesso limitado, não confiável ou lento à Internet. Mas, na maioria das circunstâncias, opte por usar emuladores com moderação.

Ao evitar emuladores, você poderá criar e inovar com os recursos de serviço mais recentes e APIs atualizadas. Você não ficará preso esperando os lançamentos de fornecedores para obter paridade de recursos. Você reduzirá suas despesas iniciais e contínuas de compra e configuração em vários sistemas de desenvolvimento e máquinas de criação. Além disso, você evitará o problema de muitos serviços em nuvem simplesmente não terem emuladores disponíveis. Uma estratégia de teste que depende de emulação impossibilitará o uso desses serviços (resultando em soluções alternativas potencialmente mais caras) ou produzirá código e configurações que não foram bem testados.

Ao usar a emulação para testes, você ainda deve testar na nuvem para verificar a configuração e testar as interações com serviços em nuvem que só podem ser simulados ou reproduzidos em um ambiente emulado.

Testes de desafios no local

Ao usar emuladores e chamadas simuladas para testar no desktop local, você pode observar inconsistências de teste à medida que seu código progride de ambiente para ambiente em seu pipeline de CI/CD. Os testes unitários para validar a lógica de negócios da aplicação no desktop podem não testar com precisão aspectos críticos dos serviços em nuvem.

Os exemplos a seguir fornecem casos a serem observados nos testes realizados no local com simulações e emuladores:

Exemplo: a função do Lambda cria um bucket do S3

Se a lógica de uma função do Lambda depender da criação de um bucket do S3, um teste completo deverá confirmar que o Amazon S3 foi chamado e que o bucket foi criado com sucesso.

- Em uma configuração de teste de simulação, você pode simular uma resposta bem-sucedida e potencialmente adicionar um caso de teste para lidar com uma resposta de falha.
- Em um cenário de teste de emulação, a API CreateBucket pode ser chamada, mas você precisa estar ciente de que a identidade que faz a chamada local não se origina do serviço do Lambda. A identidade da chamada não assumirá uma função de segurança como na nuvem. Portanto, uma autenticação de espaço reservado será usada em seu lugar, possivelmente com uma função ou identidade de usuário mais permissiva que será diferente quando executada na nuvem.

As configurações de simulação e emulação testarão o que a função do Lambda fará se chamar o Amazon S3; no entanto, esses testes não verificarão se a função do Lambda, conforme configurada, é capaz de criar com êxito o bucket do Amazon S3. Você deve garantir que a função atribuída à função tenha uma política de segurança anexada que permita que a função execute a ação `s3:CreateBucket`. Caso contrário, a função provavelmente falhará quando implantada em um ambiente de nuvem.

Exemplo: a função do Lambda processa mensagens de uma fila do Amazon SQS

Se uma fila do Amazon SQS for a origem de uma função do Lambda, um teste completo deverá verificar se a função do Lambda vai ser invocada com êxito quando uma mensagem for colocada em uma fila.

Os testes de emulação e simulação geralmente são configurados para executar o código da função do Lambda diretamente e para simular a integração do Amazon SQS passando uma carga útil de eventos JSON (ou um objeto desserializado) como entrada do manipulador da função.

O teste local que simula a integração do Amazon SQS testará o que a função do Lambda fará quando for chamada pelo Amazon SQS com uma determinada carga útil, mas o teste não verificará se o Amazon SQS invocará com êxito a função do Lambda quando ela for implantada em um ambiente de nuvem.

Alguns exemplos de problemas de configuração que você pode encontrar com o Amazon SQS e o Lambda incluem o seguinte:

- O tempo limite de visibilidade do Amazon SQS é muito baixo, resultando em várias invocações quando apenas uma foi planejada.
- A função de execução da função do Lambda não permite a leitura de mensagens da fila (por meio de `sqs:ReceiveMessage`, `sqs:DeleteMessage` ou `sqs:GetQueueAttributes`).
- O evento de exemplo que é passado para a função do Lambda excede a cota de tamanho de mensagem do Amazon SQS. Portanto, o teste é inválido porque o Amazon SQS nunca seria capaz de enviar uma mensagem desse tamanho.

Como mostrado nesses exemplos, os testes que abrangem a lógica de negócios, mas não as configurações entre os serviços de nuvem, provavelmente fornecerão resultados não confiáveis.

Perguntas frequentes

Tenho uma função do Lambda que executa cálculos e retorna um resultado sem chamar qualquer outro serviço. Preciso realmente testá-la na nuvem?

Sim. As funções do Lambda têm parâmetros de configuração que podem alterar o resultado do teste. Todo código de função do Lambda tem uma dependência nas configurações de tempo limite e memória, o que pode causar falha da função se essas configurações não forem definidas corretamente. As políticas do Lambda também permitem o registro de saída padrão para o Amazon CloudWatch. Mesmo que seu código não chame o CloudWatch diretamente, é necessária uma permissão para ativar o registro. Essa permissão necessária não pode ser simulada ou emulada com precisão.

Como os testes na nuvem podem ajudar nos testes unitários? Caso esteja na nuvem e se conecte a outros recursos, não é um teste de integração?

Nós definimos testes unitários como testes que operam isoladamente em componentes da arquitetura, mas isso não impede que os testes incluam componentes que possam chamar outros serviços ou usar alguma comunicação de rede.

Muitas aplicações com tecnologia sem servidor têm componentes da arquitetura que podem ser testados isoladamente, mesmo na nuvem. Um exemplo é uma função do Lambda que recebe entradas, processa os dados e envia uma mensagem para uma fila do Amazon SQS. Um teste unitário dessa função provavelmente testaria se os valores de entrada resultariam na presença de determinados valores na mensagem colocada em fila.

Considere um teste escrito usando o padrão Organizar, Agir, Declarar:

- Organizar: aloque recursos (uma fila para receber mensagens e a função em teste).
- Agir: chame a função em teste.
- Declarar: recupere a mensagem enviada pela função e valide a saída.

Uma abordagem de teste de simulação consistiria em simular a fila com um objeto simulado em andamento e em criar uma instância em andamento da classe ou do módulo que conteria o código da função do Lambda. Durante a fase Declarar, a mensagem colocada em fila seria recuperada do objeto simulado.

Em uma abordagem baseada em nuvem, o teste criaria uma fila do Amazon SQS para fins do teste e implantaria a função do Lambda com variáveis de ambiente configuradas para usar a fila isolada do Amazon SQS como destino de saída. Depois de executar a função do Lambda, o teste recuperaria a mensagem da fila do Amazon SQS.

O teste baseado em nuvem executaria o mesmo código, declararia o mesmo comportamento e validaria a correção funcional da aplicação. No entanto, ele teria a vantagem adicional de poder validar as configurações da função do Lambda: o perfil do IAM, as políticas do IAM e as configurações de tempo limite e memória da função.

Próximas etapas e recursos

Use os recursos a seguir para saber mais e explore os exemplos práticos dos testes.

Exemplos de implementações

O repositório de exemplos de testes com tecnologia sem servidor no GitHub contém exemplos concretos de testes que seguem os padrões e as práticas recomendadas descritos neste guia. O repositório contém exemplos de códigos e o passo a passo guiado dos processos de simulação, emulação e teste em nuvem descritos nas seções anteriores. Use esse repositório para se atualizar com as orientações mais recentes sobre testes com tecnologia sem servidor da AWS.

Outras fontes de leitura

Visite o Serverless Land para acessar os blogs, vídeos e treinamentos mais recentes sobre tecnologias sem servidor da AWS.

A leitura das seguintes publicações do blog da AWS também é recomendada:

- Acelerar o desenvolvimento com tecnologia sem servidor com o AWS SAM Accelerate (publicação do blog da AWS)

- Aumentar a velocidade do desenvolvimento com o CDK Watch (publicação no blog da AWS)
- Simular integrações de serviços com o AWS Step Functions Local (publicação no blog da AWS)
- Conceitos básicos dos testes de aplicações com tecnologia sem servidor (publicação no blog da AWS)

Ferramentas

- AWS SAM: testar e depurar aplicações com tecnologia sem servidor
- AWS SAM: integrar com testes automatizados
- Lambda: testar funções do Lambda no console do Lambda

Criar funções do Lambda com Node.js

Você pode executar o código JavaScript com Node.js noAWS Lambda. O Lambda fornece runtimes para Node.js que executam seu código para processar eventos. Seu código é executado em um ambiente que inclui o AWS SDK for JavaScript, com as credenciais de uma função do AWS Identity and Access Management (IAM) que você gerencia. Para saber mais sobre as versões do SDK incluídas nos runtimes do Node.js, consulte the section called “Versões do SDK incluídas no runtime”.

O Lambda oferece suporte aos runtimes Node.js a seguir.

Node.js

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
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 de junho de 2024	28 de fevereiro de 2025	31 de março de 2025

> **Note**
>
> Os runtimes do Node.js 18 e de versões posteriores usam o AWS SDK para JavaScript v3. Para fazer a migração de uma função de um runtime anterior, siga o workshop de migração no GitHub. Para obter mais informações sobre o SDK da AWS para a versão 3 do JavaScript, consulte a postagem no blog O SDK da AWS modular para JavaScript já está disponível para o público em geral.

Para criar uma função em Node.js.

1. Abra o console do lambda.
2. Escolha a opção Criar função.
3. Configure as seguintes opções:
   - Nome da função: digite um nome para a função.
   - Runtime: escolha Node.js 20.x.
4. Escolha a opção Criar função.
5. Para configurar um evento de teste, escolha Test (Testar).
6. Em Nome do evento, insira **`test`**.
7. Escolha Salvar alterações.
8. Escolha Testar para invocar a função.

O console cria uma função do Lambda com um único arquivo de origem denominado `index.js` ou `index.mjs`. Você pode editar esse arquivo e adicionar mais arquivos no editor de códigos integrado. Para salvar suas alterações, selecione Salvar. Em seguida, para executar seu código, escolhaTeste.

> **Note**
>
> O console do Lambda usa o AWS Cloud9 para fornecer um ambiente de desenvolvimento integrado no navegador. Você também pode usar o AWS Cloud9 para desenvolver funções do Lambda em seu próprio ambiente. Para obter mais informações, consulte Working with AWS Lambda functions using the AWS Toolkit no Guia do usuário do AWS Cloud9.

O arquivo `index.js` ou `index.mjs` exporta uma função denominada `handler` que assume um objeto de evento e um objeto de contexto. Esta é a função de manipulador que o Lambda chama quando a função é invocada. O runtime da função do Node.js recebe eventos de invocação do Lambda e os transmite ao manipulador. Na configuração da função, o valor do manipulador é `index.handler`.

Quando você salva seu código de função, o console do Lambda cria um pacote de implantação de arquivos .zip. Quando desenvolver o código de função fora do console (usando um IDE), você precisará criar um pacote de implantação para carregar o código na função do Lambda.

> **Note**
>
> Para começar com o desenvolvimento de aplicativos no ambiente local, implante um dos aplicativos de exemplo disponíveis no repositório do GitHub deste guia.
>
> Aplicações de exemplo do Lambda em Node.js
>
> - blank-nodejs: uma função do Node.js que mostra o uso do registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.
> - nodejs-apig: uma função com endpoint de API pública que processa um evento do API Gateway e retorna uma resposta HTTP.
> - efs-nodejs: uma função que usa um sistema de arquivos do Amazon EFS em uma Amazon VPC. Esse exemplo inclui uma VPC, um sistema de arquivos, destinos de montagem e ponto de acesso configurado para uso com o Lambda.

O runtime transmite um objeto de contexto para o manipulador, além do evento de invocação. O objeto de contexto contém informações adicionais sobre a invocação, a função e o ambiente de execução. Outras informações estão disponíveis de variáveis de ambiente.

Sua função do Lambda é fornecida com um grupo de logs do CloudWatch Logs. O runtime envia detalhes sobre cada invocação para o CloudWatch Logs. Ele retransmite quaisquer logs que sua função produz durante a invocação. Se a função retornar um erro, o Lambda formatará o erro e o retornará para o invocador.

Tópicos

- Instrumentação do código Node.js no AWS Lambda

Inicialização do Node.js

O Node.js tem um modelo de loop de eventos exclusivo que torna seu comportamento de inicialização diferente de outros runtimes. Especificamente, o Node.js utiliza um modelo de E/S sem bloqueio que oferece suporte a operações assíncronas. Esse modelo permite que o Node.js funcione de modo eficiente com a maioria das workloads. Por exemplo, se uma função Node.js fizer uma chamada de rede, a solicitação poderá ser designada como uma operação assíncrona e colocada em uma fila de retorno de chamada. A função poderá continuar processando outras operações dentro da pilha de chamadas principal sem ser bloqueada aguardando o retorno da chamada de rede. Após a chamada de rede ser concluída, o retorno de chamada é executado e, em seguida, removido da fila de retorno de chamada.

Algumas tarefas de inicialização podem ser executadas de modo assíncrono. Não há garantia de que a execução dessas tarefas assíncronas será concluída antes de uma invocação. Por exemplo, um código que faz uma chamada de rede para buscar um parâmetro do AWS Parameter Store poderá não estar concluído quando o Lambda executar a função de manipulador. Como resultado, a variável poderá ser nula durante uma invocação. Para evitar isso, certifique-se de que as variáveis e outros códigos assíncronos estejam totalmente inicializados antes de continuar com o restante da lógica de negócios principal da função.

Como alternativa, é possível designar o código da função como um módulo ES, o que permite que você use `await` no nível superior do arquivo, fora do escopo do manipulador de função. Quando você `await` cada `Promise`, o código de inicialização assíncrona é concluído antes das invocações do manipulador, maximizando a eficácia da simultaneidade provisionada ao reduzir a latência de inicialização a frio. Para obter mais informações e um exemplo, consulte Uso de módulos ES do Node.js e do “await” em nível superior no AWS Lambda.

Designar um manipulador de funções como módulo ES

Por padrão, o Lambda trata arquivos com o sufixo `.js` como módulos CommonJS. Como opção, é possível designar o código como um módulo ES. Isso pode ser feito de duas maneiras: especificando o `type` como `module` no arquivo da função `package.json` ou usando a extensão do nome do arquivo `.mjs`. Na primeira abordagem, o código da função trata todos os arquivos `.js` como módulos ES, enquanto no segundo cenário, somente o arquivo especificado com `.mjs` corresponde a um módulo ES. É possível mesclar módulos ES e módulos CommonJS nomeando-os `.mjs` e

`.cjs`, respectivamente, pois os arquivos `.mjs` serão sempre módulos ES, e os arquivos `.cjs` serão sempre módulos CommonJS.

O Lambda pesquisa pastas na variável de ambiente `NODE_PATH` ao carregar módulos de ES. É possível carregar o AWS SDK incluído no runtime ao usar as instruções `import` do módulo de ES. Também é possível carregar módulos ES usando camadas.

Versões do SDK incluídas no runtime

A versão do AWS SDK incluída no runtime do Node.js depende da versão do runtime e da Região da AWS. Para encontrar a versão do SDK incluída no runtime que você está usando, crie uma função do Lambda com o código a seguir.

> **Note**
>
> O exemplo de código mostrado abaixo para o Node.js versão 18 e acima usa o formato CommonJS. Se você criar a função no console do Lambda, não se esqueça de renomear o arquivo que contém o código como `index.js`.

Example Node.js 18 e acima

```
const { version } = require("@aws-sdk/client-s3/package.json");

exports.handler = async () => ({ version });
```

Uma resposta é retornada no seguinte formato:

```
{
  "version": "3.462.0"
}
```

Usar o keep-alive para conexões TCP

O agente Node.js HTTP/HTTPS padrão cria uma nova conexão TCP para cada nova solicitação. Para evitar o custo de estabelecer novas conexões, você pode usar `keepAlive: true` para reutilizar as conexões que sua função faz usando o AWS SDK para JavaScript. O Keep-alive pode

reduzir os tempos de solicitação de funções do Lambda que fazem várias chamadas de API usando o SDK.

No AWS SDK para JavaScript 3.x, que está incluído no runtime do Lambda para `nodejs18.x` e em versões posteriores, o recurso keep-alive está habilitado por padrão. Para desativar o keep-alive, consulte Reusing connections with keep-alive in Node.js no Guia do desenvolvedor do AWS SDK para JavaScript 3.x. Para obter mais informações sobre como usar o keep-alive, consulte HTTP keep-alive is on by default in modular AWS SDK for JavaScript no Blog de ferramentas da AWS para o desenvolvedor.

Carregamento de certificado de CA

Até a versão do runtime do Node.js 18, o Lambda carrega automaticamente certificados de CA (autoridade de certificação) específicos da Amazon para facilitar a criação de funções que interagem com outros serviços da AWS. Por exemplo, o Lambda inclui os certificados do Amazon RDS necessários para validar o certificado de identidade do servidor instalado no banco de dados do Amazon RDS. Esse comportamento pode ter um impacto na performance durante inicializações a frio.

A partir do Node.js 20, o Lambda não carrega mais certificados de CA adicionais por padrão. O runtime do Node.js 20 contém um arquivo de certificado com todos os certificados de CA da Amazon localizado em `/var/runtime/ca-cert.pem`. Para restaurar o mesmo comportamento do Node.js 18 e de runtimes anteriores, defina a variável de ambiente `NODE_EXTRA_CA_CERTS` como `/var/runtime/ca-cert.pem`.

Para otimizar a performance, recomendamos que você agrupe somente os certificados de que precisa com o pacote de implantação e carregue-os usando a variável de ambiente `NODE_EXTRA_CA_CERTS`. O arquivo de certificados deve consistir em um ou mais certificados de CA raiz ou intermediários confiáveis no formato PEM. Por exemplo, para o RDS, inclua os certificados necessários junto com seu código como `certificates/rds.pem`. Em seguida, carregue os certificados configurando `NODE_EXTRA_CA_CERTS` como`/var/task/certificates/rds.pem`.

Definir o manipulador de função do Lambda em Node.js

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. A função é executada até que o manipulador retorne uma resposta, seja encerrado ou atinja o tempo limite.

O exemplo de função a seguir registra em log o conteúdo do objeto de evento e retorna a localização dos logs.

> **Note**
>
> Esta página exibe exemplos de manipuladores de módulos CommonJS e ES. Para saber mais sobre a diferença entre esses dois tipos de manipulador, consulte Designar um manipulador de funções como módulo 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 você configura uma função, o valor da configuração do manipulador é o nome do arquivo e o nome do módulo do manipulador exportado, separados por um ponto. O padrão no console e nos exemplos deste guia é `index.handler`. Isso indica o método do `handler` que é exportado do arquivo `index.js`.

O runtime transmite argumentos para o método do handler. O primeiro argumento é o objeto `event`, que contém informações do chamador. O invocador passa essas informações como uma string no formato JSON ao chamar Invoke, e o runtime as converte em um objeto. Quando um serviço da AWS invoca a sua função, a estrutura do evento varia de acordo com o serviço.

O segundo argumento é o objeto de contexto, que contém informações sobre a invocação, a função e o ambiente de execução. No exemplo anterior, a função obtém o nome do fluxo de logs do objeto de contexto e o retorna para o invocador.

Também é possível usar um argumento de retorno de chamada, que é uma função que você pode chamar nos manipuladores não assíncronos para enviar uma resposta. Recomendamos o uso de async/await em vez dos retornos de chamada. Async/await oferece melhor legibilidade, tratamento de erros e eficiência. Para obter mais informações sobre as diferenças entre async/await e retornos de chamada, consulte Usar retornos de chamada.

Nomenclatura

Quando você configura uma função, o valor da configuração do manipulador é o nome do arquivo e o nome do módulo do manipulador exportado, separados por um ponto. O padrão para as funções criadas no console e para os exemplos deste guia é `index.handler`. Isso indica o método de `handler` que é exportado do arquivo `index.js` ou `index.mjs`.

Se você criar uma função no console usando um nome de arquivo ou nome de manipulador de funções diferente, deverá editar o nome do manipulador padrão.

Para alterar o nome do manipulador de funções (console)

1. Abra a página Funções do console do Lambda e escolha sua função.
2. Escolha a guia Código.
3. Role para baixo até o painel Configurações de runtime e escolha Editar.
4. Em Manipulador, insira o novo nome para seu manipulador de funções.
5. Escolha Salvar.

Usar async/await

Se o código realizar uma tarefa assíncrona, use o padrão async/await para garantir que a execução do manipulador seja finalizada. Async/await é uma forma concisa e legível de escrever código

assíncrono em Node.js, sem a necessidade de retornos de chamada aninhados ou promessas de encadeamento. Com async/await, é possível escrever um código que seja lido como código síncrono e, ao mesmo tempo, seja assíncrono e sem bloqueio.

A palavra-chave `async` marca uma função como assíncrona, e a palavra-chave `await` pausa a execução da função até que uma `Promise` seja resolvida.

> **Note**
>
> Aguarde a conclusão dos eventos assíncronos. Se a função retornar antes que os eventos assíncronos sejam concluídos, a função poderá falhar ou causar um comportamento inesperado em sua aplicação. Isso pode acontecer quando um loop `forEach` contiver um evento assíncrono. Os loops `forEach` esperam uma chamada síncrona. Para obter mais informações, consulte Array.prototype.forEach() na documentação do Mozilla.

ES module handler

Example – Solicitação HTTP com 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 – Solicitação HTTP com 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;
};
```

O próximo exemplo usa async/await para listar buckets do Amazon Simple Storage Service.

> **Note**
>
> Antes de usar esse exemplo, verifique se o perfil de execução da sua função tem permissões de leitura do Amazon S3.

ES module handler

Example – AWS SDK v3 com async/await

Este exemplo usa o AWS SDK for JavaScript v3, que está disponível no runtime `nodejs18.x` e em versões posteriores.

```
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 com async/await

Este exemplo usa o AWS SDK for JavaScript v3, que está disponível no runtime `nodejs18.x` e em versões posteriores.

```
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;
};
```

Usar retornos de chamada

Recomendamos usar async/await para declarar o manipulador da função em vez de usar retornos de chamada. Async/await é a melhor opção por vários motivos:

- Legibilidade: o código async/await é mais fácil de ler e entender do que o código de retorno de chamada, que pode logo se tornar difícil de seguir e transformar o retorno de chamada em um tormento.
- Depuração e tratamento de erros: pode ser difícil depurar código baseado em retorno de chamada. A pilha de chamadas pode se tornar difícil de acompanhar, e os erros podem ser facilmente ignorados. Com async/await, você pode usar blocos try/catch para manipular erros.
- Eficiência: retornos de chamada muitas vezes exigem alternância entre diferentes partes do código. O async/await pode reduzir o número de alternâncias de contexto, resultando em um código mais eficiente.

Quando você usa retornos de chamada no manipulador, a função continua em execução até que o loop de evento esteja vazio ou o tempo da função se esgote. A resposta não é enviada para o chamador até que todas as tarefas de loop de evento estejam concluídas. Se a função expirar, um erro será retornado. É possível configurar o runtime para enviar a resposta imediatamente, definindo context.callbackWaitsForEmptyEventLoop como false.

A função de retorno de chamada usa dois argumentos: um `Error` e uma resposta. O objeto de resposta deve ser compatível com `JSON.stringify`.

O exemplo de função a seguir verifica um URL e retorna o código de status para o invocador.

ES module handler

Example – Solicitação HTTP com 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 – Solicitação HTTP com 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));
    });
};
```

No próximo exemplo, a resposta do Amazon S3 é retornada ao invocador assim que fica disponível. O tempo limite em execução no loop de evento é congelado e continuará sendo executado na próxima vez que a função for invocada.

> **Note**
>
> Antes de usar esse exemplo, verifique se o perfil de execução da sua função tem permissões de leitura do Amazon S3.

ES module handler

Example – AWS SDK v3 com callbackWaitsForEmptyEventLoop

Este exemplo usa o AWS SDK for JavaScript v3, que está disponível no runtime `nodejs18.x` e em versões posteriores.

```
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 com callbackWaitsForEmptyEventLoop

Este exemplo usa o AWS SDK for JavaScript v3, que está disponível no runtime `nodejs18.x` e em versões posteriores.

```
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);
};
```

Implantar funções do Lambda em Node.js com arquivos .zip

O código da função do AWS Lambda compreende um arquivo .js ou .mjs contendo o código do manipulador da função, juntamente com quaisquer pacotes e módulos adicionais dos quais o código dependa. Para implantar o código dessa função no Lambda, você usa um pacote de implantação. Esse pacote pode ser um arquivo .zip ou uma imagem de contêiner. Para obter mais informações sobre como usar imagens de contêiner com Node.js, consulte Implantar funções do Lambda em Node.js com imagens de contêiner.

Para criar um pacote de implantação como arquivo .zip, você pode usar um utilitário de arquivo .zip integrado da ferramenta da linha de comando ou qualquer outro utilitário de arquivo .zip, como o 7zip. Os exemplos mostrados nas seções a seguir pressupõem que você esteja usando uma ferramenta `zip` da linha de comando em um ambiente Linux ou MacOS. Para usar os mesmos comandos no Windows, você pode instalar o Subsistema Windows para Linux para obter uma versão do Ubuntu e do Bash integrada ao Windows.

Observe que o Lambda usa permissões de arquivo POSIX, então pode ser necessário definir permissões para a pasta do pacote de implantação antes da criação do arquivo .zip.

Tópicos

Dependências de runtime em Node.js

Para funções do Lambda que usam o runtime do Node.js, uma dependência pode ser qualquer módulo Node.js. O runtime do Node.js inclui várias bibliotecas conhecidas, bem como uma versão do AWS SDK for JavaScript. O runtime do Lambda para `nodejs16.x` inclui a versão 2.x do SDK. As versões de runtime `nodejs18.x` e posteriores incluem a versão 3 do SDK. Para usar a versão 2 do SDK com as versões de runtime `nodejs18.x` e posteriores, adicione o SDK ao arquivo .zip. do pacote de implantação. Se o runtime escolhido incluir a versão do SDK que você está usando, não será necessário incluir a biblioteca do SDK no arquivo .zip. Para descobrir qual versão do SDK está

incluída no runtime que você está usando, consulte the section called “Versões do SDK incluídas no runtime”.

O Lambda atualiza periodicamente as bibliotecas do SDK no runtime do Node.js para incluir os recursos e upgrades de segurança mais recentes. O Lambda também aplica patches de segurança e atualizações às outras bibliotecas inclusas no runtime. Para ter controle total das dependências em seu pacote, é possível adicionar a versão preferencial de qualquer dependência inclusa no runtime ao pacote de implantação. Por exemplo, se você deseja usar uma versão específica do SDK para JavaScript, pode incluí-la no arquivo .zip como uma dependência. Para obter mais informações sobre como adicionar dependências inclusas no runtime ao arquivo .zip, consulte Caminho de pesquisa de dependências e bibliotecas incluídas no runtime.

No modelo de responsabilidade compartilhada da AWS, você é responsável pelo gerenciamento de todas as dependências dos pacotes de implantação das suas funções. Isso inclui a aplicação de atualizações e patches de segurança. Para atualizar as dependências no pacote de implantação da função, primeiro crie um novo arquivo .zip e depois carregue esse arquivo no Lambda. Consulte Criar um pacote de implantação .zip com dependências e Criação e atualização de funções do Lambda em Node.js usando arquivos .zip para obter mais informações.

Criar um pacote de implantação .zip sem dependências

Se o código de função não tiver dependências, exceto para as bibliotecas inclusas no runtime do Lambda, o arquivo .zip conterá somente o arquivo `index.js` ou `index.mjs` com o código do manipulador da função. Use seu utilitário zip preferencial para criar um arquivo .zip com o arquivo `index.js` ou `index.mjs` na raiz. Se o arquivo que contém o código do manipulador não estiver na raiz do arquivo .zip, o Lambda não poderá executar o código.

Para saber como implantar o arquivo .zip para criar uma função do Lambda ou atualizar uma já existente, consulte Criação e atualização de funções do Lambda em Node.js usando arquivos .zip.

Criar um pacote de implantação .zip com dependências

Se o código de função depender de pacotes ou módulos que não estejam inclusos no runtime do Node.js do Lambda, é possível adicionar essas dependências ao arquivo .zip com o código de função ou usar uma camada do Lambda. As instruções desta seção mostram como incluir as dependências no pacote de implantação .zip. Para obter instruções sobre como incluir suas dependências em uma camada, consulte the section called “Criar uma camada Node.js para suas dependências”.

Os comandos de exemplo da CLI a seguir criam um arquivo .zip denominado `my_deployment_package.zip` contendo o arquivo `index.js` ou `index.mjs` com o código do manipulador da função e as dependências. No exemplo, você instala as dependências usando o gerenciador de pacotes npm.

Para criar o pacote de implantação

1. Navegue até o diretório do projeto que contém seu arquivo de código-fonte `index.js` ou `index.mjs`. Neste exemplo, o diretório se chama `my_function`.

   ```
   cd my_function
   ```

2. Instale as bibliotecas necessárias para a função no diretório `node_modules` usando o comando `npm install`. Neste exemplo, você instala o AWS X-Ray SDK for Node.js.

   ```
   npm install aws-xray-sdk
   ```

   Isso cria uma estrutura de pastas semelhante à seguinte:

   ```
   ~/my_function
   ### index.mjs
   ### node_modules
       ### async
       ### async-listener
       ### atomic-batcher
       ### aws-sdk
       ### aws-xray-sdk
       ### aws-xray-sdk-core
   ```

   Também é possível adicionar módulos personalizados criados por você ao pacote de implantação. Crie um diretório em `node_modules` com o nome do seu módulo e salve seus pacotes personalizados gravados nesse diretório.

3. Crie um arquivo .zip que contenha o conteúdo da pasta do seu projeto na raiz. Use a opção `r` (recursiva) para garantir que zip compacte as subpastas.

   ```
   zip -r my_deployment_package.zip .
   ```

Criar uma camada Node.js para suas dependências

As instruções nesta seção mostram como incluir suas dependências em uma camada. Para obter instruções sobre como incluir suas dependências em seu pacote de implantação, consulte the section called “Criar um pacote de implantação .zip com dependências”.

Quando você adiciona uma camada a uma função, o Lambda carrega o conteúdo da camada no diretório `/opt` desse ambiente de execução. Para cada runtime do Lambda, a variável `PATH` já inclui caminhos de pasta específica no diretório `/opt`. Para garantir que a variável `PATH` colete o conteúdo da camada, o arquivo .zip da camada deve ter suas dependências nos seguintes caminhos de pasta:

- `nodejs/node_modules`
- `nodejs/node16/node_modules (NODE_PATH)`
- `nodejs/node18/node_modules (NODE_PATH)`
- `nodejs/node20/node_modules (NODE_PATH)`

Por exemplo, sua estrutura de arquivo .zip da sua camada pode ser assim:

```
xray-sdk.zip
# nodejs/node_modules/aws-xray-sdk
```

Além disso, o Lambda detecta automaticamente todas as bibliotecas no diretório `/opt/lib` e quaisquer binários no diretório `/opt/bin`. Para garantir que o Lambda encontre corretamente o conteúdo da sua camada, você também pode criar uma camada com a seguinte estrutura:

```
custom-layer.zip
# lib
    | lib_1
    | lib_2
# bin
    | bin_1
    | bin_2
```

Depois de empacotar sua camada, consulte the section called “Criar e excluir camadas” e the section called “Adicionar camadas” para concluir sua configuração de camada.

Caminho de pesquisa de dependências e bibliotecas incluídas no runtime

O runtime do Node.js inclui várias bibliotecas conhecidas, bem como uma versão do AWS SDK for JavaScript. Se você desejar usar uma versão diferente de uma biblioteca inclusa no runtime, poderá fazer isso ao empacotá-la com a função ou adicioná-la como uma dependência no pacote de implantação. Por exemplo, você pode usar uma versão diferente do SDK ao adicioná-la ao pacote de implantação .zip. Também é possível incluí-la em uma camada do Lambda para a função.

Quando você usa uma instrução `import` ou `require` em seu código, o runtime do Node.js pesquisa os diretórios no caminho `NODE_PATH` até localizar o módulo. Por padrão, o primeiro local que o runtime pesquisa é o diretório no qual o pacote de implantação .zip é descompactado e montado (`/var/task`). Se você incluir uma versão de uma biblioteca incluída no runtime do pacote de implantação, essa versão terá precedência sobre a versão incluída no runtime. As dependências do pacote de implantação também têm precedência sobre as dependências das camadas.

Quando você adiciona uma dependência a uma camada, o Lambda a extrai para `/opt/nodejs/nodexx/node_modules`, em que `nodexx` representa a versão do runtime que você está usando. No caminho de pesquisa, esse diretório tem precedência sobre o diretório que contém as bibliotecas incluídas no runtime (`/var/lang/lib/node_modules`). As bibliotecas em camadas da função, portanto, têm precedência sobre as versões incluídas no runtime.

É possível visualizar o caminho de pesquisa completo para a função do Lambda ao adicionar a linha de código a seguir.

```
console.log(process.env.NODE_PATH)
```

Você também pode adicionar dependências em uma pasta separada dentro do pacote .zip. Por exemplo, você pode adicionar um módulo personalizado a uma pasta no pacote .zip chamada `common`. Quando o pacote .zip é descompactado e montado, essa pasta é colocada dentro do diretório `/var/task`. Para usar uma dependência de uma pasta no pacote de implantação .zip em seu código, use uma instrução `import { } from` ou `const { } = require()`, com base no seu uso da resolução do módulo CJS ou ESM. Por exemplo:

```
import { myModule } from './common'
```

Se você empacotar o código com `esbuild`, `rollup` ou similares, as dependências usadas por sua função serão empacotadas em um ou mais arquivos. Recomendamos usar esse método para comercializar dependências sempre que possível. Em comparação com a adição de dependências

ao pacote de implantação, o empacotamento do código resulta em melhor performance devido à redução nas operações de E/S.

Criação e atualização de funções do Lambda em Node.js usando arquivos .zip

Depois de criar seu pacote de implantação .zip, você poderá usá-lo para criar uma nova função do Lambda ou atualizar uma existente. É possível implantar o pacote .zip usando o console do Lambda, a AWS Command Line Interface e a API do Lambda. Você também pode criar e atualizar funções do Lambda usando o AWS Serverless Application Model (AWS SAM) e o AWS CloudFormation.

O tamanho máximo de um pacote de implantação .zip para o Lambda é 250 MB (descompactado). Esse limite se aplica ao tamanho combinado de todos os arquivos que você carrega, inclusive qualquer camada do Lambda.

O runtime do Lambda precisa de permissão para ler os arquivos no pacote de implantação. Na notação octal de permissões do Linux, o Lambda precisa de 644 permissões para arquivos não executáveis (rw-r--r--) e 755 permissões (rwxr-xr-x) para diretórios e arquivos executáveis.

No Linux e no MacOS, use o comando `chmod` para alterar as permissões de arquivo em arquivos e diretórios do seu pacote de implantação. Por exemplo, para dar a um arquivo executável as permissões corretas, execute o comando a seguir.

```
chmod 755 <filepath>
```

Para alterar as permissões de arquivo no Windows, consulte Set, View, Change, or Remove Permissions on an Object na documentação do Microsoft Windows.

Criar e atualizar funções com arquivos .zip usando o console

Para criar uma nova função, você deve primeiro criar a função no console e depois carregar o arquivo .zip. Para atualizar uma função existente, abra a página da função e siga o mesmo procedimento para adicionar o arquivo .zip atualizado.

Se o arquivo .zip for menor que 50 MB, você poderá criar ou atualizar uma função carregando o arquivo diretamente da máquina local. Para arquivos .zip maiores que 50 MB, você deve primeiro carregar o pacote para um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando o AWS Management Console, consulte Conceitos básicos do Amazon S3. Para carregar arquivos usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

> **Note**
>
> Você não pode alterar o tipo de pacote de implantação (.zip ou imagem de contêiner) de uma função existente. Por exemplo, você não pode converter uma função de imagem de contêiner para usar um arquivo compactado .zip. É necessário criar uma nova função.

Para criar uma função (console)

1. Abra a página Funções do console do Lambda e escolha Criar função.
2. Escolha Author from scratch (Criar do zero).
3. Em Basic information (Informações básicas), faça o seguinte:
   a. Em Nome da função, insira o nome da função.
   b. Em Runtime, selecione o runtime que você deseja usar.
   c. (Opcional) Em Arquitetura, escolha a arquitetura do conjunto de instruções para a função. O valor da arquitetura padrão é X86_64. Certifique-se de que o pacote de implantação .zip da função seja compatível com a arquitetura do conjunto de instruções que você escolheu.
4. (Opcional) Em Permissões, expanda Alterar função de execução padrão. Crie uma função de execução ou use uma existente.
5. Escolha a opção Criar função. O Lambda cria uma função básica “Hello world” usando o runtime escolhido.

Você pode carregar o arquivo .zip da máquina local (console)

1. Na página Funções do console Lambda, escolha a função para a qual você deseja carregar o arquivo .zip.
2. Selecione a guia Código.
3. No painel do Código-fonte, escolha Carregar de.
4. Escolha o arquivo .zip.
5. Para carregar o arquivo .zip, faça o seguinte:
   a. Selecione Carregar e, em seguida, selecione o arquivo .zip no seletor de arquivos.
   b. Escolha Open (Abrir).
   c. Escolha Salvar.

Para carregar um arquivo .zip de um bucket do Amazon S3 (console)

1. Na página Funções do console do Lambda, escolha a função para a qual você deseja carregar um novo arquivo .zip.
2. Selecione a guia Código.
3. No painel do Código-fonte, escolha Carregar de.
4. Escolha Local do Amazon S3.
5. Cole o URL do link do Amazon S3 do arquivo .zip e escolha Salvar.

Atualizar funções do arquivo .zip usando o editor de código do console

Para algumas funções com pacotes de implantação .zip, você pode usar o editor de código integrado do console do Lambda para atualizar o código da função diretamente. Para usar esse recurso, a função deve atender aos seguintes critérios:

- A função deve usar um dos runtimes da linguagem interpretada (Python, Node.js ou Ruby)
- O pacote de implantação da função deve ser menor que 3 MB.

O código das funções com pacotes de implantação de imagens de contêiner não pode ser editado diretamente no console.

Para atualizar o código da função usando o editor de código do console

1. Abra a página Funções do console do Lambda e selecione a função.
2. Selecione a guia Código.
3. No painel Código-fonte, selecione o arquivo de código-fonte e edite-o no editor de código integrado.
4. Quando terminar de editar o código, escolha Implantar para salvar as alterações e atualizar a função.

Criar e atualizar funções com arquivos .zip usando a AWS CLI

Você pode usar a AWS CLI para criar uma função ou atualizar uma existente usando um arquivo .zip. Use os comandos create-function e update-function-code para implantar o pacote .zip. Se o arquivo .zip for menor que 50 MB, você poderá carregar o pacote .zip de um local do arquivo

na máquina de compilação local. Para arquivos .zip maiores, você deve carregar o pacote .zip de um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

> **Note**
>
> Se você carregar o arquivo .zip de um bucket do Amazon S3 usando a AWS CLI, o bucket deverá estar na mesma Região da AWS que sua função.

Para criar uma função usando um arquivo .zip com a AWS CLI, você deve especificar o seguinte:

- O nome da função (`--function-name`)
- O runtime da função (`--runtime`)
- O nome do recurso da Amazon (ARN) da função de execução da função (`--role`)
- O nome do método do manipulador no código da função (`--handler`)

Você também deve especificar a local do arquivo .zip. Se o arquivo .zip estiver localizado em uma pasta da máquina de compilação local, use a opção `--zip-file` para especificar o caminho do arquivo, conforme mostrado no comando do exemplo a seguir.

```
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
```

Para especificar o local do arquivo .zip em um bucket do Amazon S3, use a opção `--code` conforme mostrado no comando do exemplo a seguir. Você só precisa usar o parâmetro `S3ObjectVersion` para objetos com versionamento.

```
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
```

Para atualizar uma função existente usando a CLI, especifique o nome da função usando o parâmetro `--function-name`. Você também deve especificar o local do arquivo .zip que deseja usar para atualizar o código da função. Se o arquivo .zip estiver localizado em uma pasta da

máquina de compilação local, use a opção `--zip-file` para especificar o caminho do arquivo, conforme mostrado no comando do exemplo a seguir.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Para especificar o local do arquivo .zip em um bucket do Amazon S3, use as opções `--s3-bucket` e `--s3-key` conforme mostrado no comando do exemplo a seguir. Você só precisa usar o parâmetro `--s3-object-version` para objetos com versionamento.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket DOC-EXAMPLE-BUCKET --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Criar e atualizar funções com arquivos .zip usando a API do Lamba

Para criar e atualizar funções usando um arquivo .zip, use as seguintes operações de API:

- CreateFunction
- UpdateFunctionCode

Criar e atualizar funções com arquivos .zip usando o AWS SAM

O AWS Serverless Application Model (AWS SAM) é um kit de ferramentas que ajuda a simplificar o processo de criação e execução de aplicações com tecnologia sem servidor na AWS. Você define os recursos para a aplicação em um modelo YAML ou JSON e usa a interface da linha de comando do AWS SAM (CLI do AWS SAM) para criar, empacotar e implantar aplicações. Quando você cria uma função do Lambda com base em um modelo do AWS SAM, o AWS SAM cria automaticamente um pacote de implantação .zip ou uma imagem de contêiner com o código da função e quaisquer dependências que você especificar. Para saber mais sobre como usar o AWS SAM para criar e implantar funções do Lambda, consulte Conceitos básicos do AWS SAM no Guia do desenvolvedor do AWS Serverless Application Model.

Você também pode usar o AWS SAM para criar uma função do Lambda usando um arquivo .zip existente. Para criar uma função do Lambda usando o AWS SAM, salve o arquivo .zip em um bucket do Amazon S3 ou em uma pasta local na máquina de compilação. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

No modelo do AWS SAM, o recurso `AWS::Serverless::Function` especifica a função do Lambda. Nesse recurso, defina as seguintes propriedades para criar uma função usando um arquivo .zip:

- `PackageType`: definir como `Zip`
- `CodeUri`: definir como o URI do Amazon S3 do código da função, o caminho para a pasta local ou o objeto FunctionCode
- `Runtime`: definir como o runtime escolhido

Com o AWS SAM, se o arquivo .zip for maior que 50 MB, você não precisará carregá-lo primeiro em um bucket do Amazon S3. O AWS SAM poderá carregar pacotes .zip com o tamanho máximo permitido de 250 MB (descompactados) de um local da máquina de compilação local.

Para saber mais sobre a implantação de funções usando o arquivo .zip no AWS SAM, consulte AWS::Serverless::Function no Guia do desenvolvedor do AWS SAM.

Criar e atualizar funções com arquivos .zip usando o AWS CloudFormation

Você pode usar o AWS CloudFormation para criar uma função do Lambda usando um arquivo .zip. Para criar uma função do Lambda de um arquivo.zip, primeiro carregue o arquivo em um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

No modelo do AWS CloudFormation, o recurso `AWS::Lambda::Function` especifica a função do Lambda. Nesse recurso, defina as seguintes propriedades para criar uma função usando um arquivo .zip:

- `PackageType`: definir como `Zip`
- `Code`: inserir o nome do bucket do Amazon S3 e o nome do arquivo .zip nos campos `S3Bucket` e `S3Key`
- `Runtime`: definir como o runtime escolhido

O arquivo .zip que o AWS CloudFormation gera não pode exceder 4 MB. Para saber mais sobre a implantação de funções usando o arquivo .zip no AWS CloudFormation, consulte AWS::Lambda::Function no Guia do desenvolvedor do AWS CloudFormation.

Implantar funções do Lambda em Node.js com imagens de contêiner

Existem três maneiras de criar uma imagem de contêiner para uma função do Lambda em Node.js:

- Usar uma imagem base da AWS para Node.js

  As imagens base da AWS são pré-carregadas com um runtime de linguagem, um cliente de interface de runtime para gerenciar a interação entre o Lambda e o código da sua função e um emulador de interface de runtime para testes locais.
- Usar uma imagem base somente para sistema operacional da AWS

  As imagens base somente para sistema operacional da AWS contêm uma distribuição do Amazon Linux e o emulador de interface de runtime. Essas imagens são comumente usadas para criar imagens de contêiner para linguagens compiladas, como Go e Rust e para uma linguagem ou versão de linguagem para a qual o Lambda não fornece uma imagem base, como Node.js 19. Você também pode usar imagens base somente para sistema operacional para implementar um runtime personalizado. Para tornar a imagem compatível com o Lambda, você deve incluir o cliente de interface de runtime para Node.js na imagem.
- Usar uma imagem base que não é da AWS

  Você também pode usar uma imagem base alternativa de outro registro de contêiner, como Alpine Linux ou Debian. Você também pode usar uma imagem personalizada criada por sua organização. Para tornar a imagem compatível com o Lambda, você deve incluir o cliente de interface de runtime para Node.js na imagem.

> **Tip**
>
> Para reduzir o tempo necessário para que as funções do contêiner do Lambda se tornem ativas, consulte Use multi-stage builds na documentação do Docker. Para criar imagens de contêiner eficientes, siga as Melhores práticas para gravar Dockerfiles.

Esta página explica como criar, testar e implantar imagens de contêiner para o Lambda.

Tópicos

- Imagens base da AWS para Node.js

- Usar uma imagem base da AWS para Node.js
- Usar uma imagem base alternativa com o cliente da interface de runtime

Imagens base da AWS para Node.js

A AWS oferece as seguintes imagens base para Node.js:

Tags	Runtime	Sistema operacional	Dockerfile	Desaprovação
20	Node.js 20	Amazon Linux 2023	Dockerfile para Node.js 20 no GitHub	
18	Node.js 18	Amazon Linux 2	Dockerfile para Node.js 18 no GitHub	
16	Node.js 16	Amazon Linux 2	Dockerfile para Node.js 16 no GitHub	12 de junho de 2024

Repositório do Amazon ECR: gallery.ecr.aws/lambda/nodejs

As imagens base do Node.js 20 e posteriores são baseadas na imagem de contêiner mínimo do Amazon Linux 2023. Imagens base anteriores usam o Amazon Linux 2. O AL2023 oferece várias vantagens em relação ao Amazon Linux 2, incluindo uma área de implantação menor e versões atualizadas de bibliotecas, como `glibc`.

As imagens baseadas no AL2023 usam o `microdnf` (com link simbólico `dnf`) como o gerenciador de pacotes, em vez do `yum`, que é o gerenciador de pacotes padrão no Amazon Linux 2. O `microdnf` é uma implementação autônoma do `dnf`. Para obter uma lista dos pacotes incluídos nas imagens baseadas no AL2023, consulte as colunas Contêiner mínimo em Comparar pacotes instalados em imagens de contêiner do Amazon Linux 2023. Para obter mais informações sobre as diferenças entre o AL2023 e o Amazon Linux 2, consulte Introdução ao runtime do Amazon Linux 2023 para AWS Lambda no blog AWS Compute.

> **Note**
>
> Para executar imagens baseadas no AL2023 localmente, inclusive com o AWS Serverless Application Model (AWS SAM), você deve usar o Docker versão 20.10.10 ou posterior.

Usar uma imagem base da AWS para Node.js

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- Docker (versão mínima 20.10.10 para Node.js 20 e imagens base posteriores)
- Node.js

Criação de uma imagem a partir de uma imagem base

Para criar uma imagem de contêiner a partir de uma imagem base da AWS para Node.js

1. Crie um diretório para o projeto e depois mude para esse diretório.

```
mkdir example
cd example
```

2. Crie um novo projeto Node.js com o `npm`. Para aceitar as opções padrão fornecidas na experiência interativa, pressione `Enter`.

```
npm init
```

3. Crie um novo arquivo chamado `index.js`. É possível adicionar o exemplo de código de função a seguir ao arquivo para testes ou usar o seu próprio código.

   Example Manipulador CommonJS

```
exports.handler = async (event) => {
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
```

```
    };
    return response;
};
```

4. Se a função depender de outras bibliotecas que não o AWS SDK for JavaScript, use o npm para adicioná-las ao seu pacote.
5. Crie um novo Dockerfile com a seguinte configuração:

   - Defina a propriedade `FROM` como o URI da imagem base.
   - Use o comando COPY para copiar o código da função e as dependências do runtime para `{LAMBDA_TASK_ROOT}`, uma variável de ambiente definida pelo Lambda.
   - Defina o argumento `CMD` como o manipulador de funções do 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. Crie a imagem do Docker com o comando docker build. O exemplo a seguir nomeia a imagem como `docker-image` e atribui a ela a tag `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> O comando especifica a opção `--platform linux/amd64` para garantir que seu contêiner seja compatível com o ambiente de execução do Lambda, independentemente da arquitetura da sua máquina de compilação. Se você pretende criar uma função do Lambda usando a arquitetura do conjunto de instruções ARM64, certifique-se de alterar o comando para usar a opção `--platform linux/arm64` em vez disso.

(Opcional) Teste a imagem localmente

1. Inicie a imagem do Docker com o comando docker run. Neste exemplo, `docker-image` é o nome da imagem e `test` é a tag.

   ```
   docker run --platform linux/amd64 -p 9000:8080 docker-image:test
   ```

   Esse comando executa a imagem como um contêiner e cria um endpoint local em `localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Se você criou a imagem do Docker para a arquitetura do conjunto de instruções ARM64, certifique-se de usar a opção `--platform linux/arm64`, em vez de `--platform linux/amd64`.

2. Em uma nova janela de terminal, publique um evento no endpoint local.

   Linux/macOS

   No Linux e no MacOS, execute o seguinte comando `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   No PowerShell, execute o seguinte comando `Invoke-WebRequest`:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Obtenha o ID do contêiner.

```
docker ps
```

4. Use o comando docker kill para parar o contêiner. Nesse comando, substitua `3766c4ab331c` pelo ID do contêiner da etapa anterior.

```
docker kill 3766c4ab331c
```

Implantação da imagem

Para enviar a imagem ao Amazon ECR e criar a função do Lambda

1. Execute o comando get-login-password para autenticar a CLI do Docker no seu registro do Amazon ECR.

   - Defina o valor `--region` para a Região da AWS onde você deseja criar o repositório do Amazon ECR.
   - Substituir `111122223333` por seu ID da Conta da 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. Crie um repositório no Amazon ECR usando o 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**
>
> O repositório do Amazon ECR deve estar na mesma Região da AWS que a função do Lambda.

Se tiver êxito, você verá uma resposta como esta:

```
{
    "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. Copie o `repositoryUri` da saída na etapa anterior.
4. Execute o comando docker tag para aplicar uma tag na sua imagem local em seu repositório do Amazon ECR como a versão mais recente. Neste comando:

   - Substitua `docker-image:test` pelo nome e tag da sua imagem do Docker.
   - Substitua `<ECRrepositoryUri>` pelo `repositoryUri` que você copiou. Certifique-se de incluir `:latest` no final do URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Exemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Execute o comando docker push para implantar a imagem local no repositório do Amazon ECR. Certifique-se de incluir `:latest` no final do URI do repositório.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crie um perfil de execução para a função, caso ainda não tenha um. Você precisará do nome do recurso da Amazon (ARN) do perfil na próxima etapa.
7. Criar a função do Lambda. Em `ImageUri`, especifique o URI do repositório anterior. Certifique-se de incluir `:latest` no final do 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**
>
> É possível criar uma função usando uma imagem em uma conta da AWS diferente desde que a imagem esteja na mesma região da função do Lambda. Para ter mais informações, consulte Permissões entre contas do Amazon ECR.

8. Invoque a função.

```
aws lambda invoke --function-name hello-world response.json
```

Você obterá uma resposta parecida com esta:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Para ver a saída da função, verifique o arquivo `response.json`.

Para atualizar o código da função, você deve criar a imagem novamente, fazer upload da nova imagem no repositório do Amazon ECR e, em seguida, usar o comando update-function-code para implantar a imagem na função do Lambda.

O Lambda resolve a tag de imagem em um resumo de imagem específico. Isso significa que, se você apontar a tag de imagem que foi usada para implantar a função em uma nova imagem no Amazon ECR, o Lambda não atualizará automaticamente a função para usar a nova imagem. Para implantar a nova imagem na mesma função do Lambda, você deverá usar o comando `update-function-code`, mesmo que a tag da imagem no Amazon ECR permaneça a mesma.

Usar uma imagem base alternativa com o cliente da interface de runtime

Se você usar uma imagem base somente para sistema operacional ou uma imagem base alternativa, deverá incluir o cliente de interface de runtime na imagem. O cliente de interface de runtime estende API de tempo de execução do Lambda, que gerencia a interação entre o Lambda e o código da sua função.

Instale o cliente de interface de runtime em Node.js usando o gerenciador de pacotes npm:

```
npm install aws-lambda-ric
```

Você também pode fazer download do cliente de interface de runtime Node.js no GitHub. O cliente da interface de runtime é compatível com as seguintes versões do Node.js:

- 14.x
- 16.x
- 18.x
- 20.x

O exemplo a seguir demonstra como criar uma imagem de contêiner para Node.js usando uma imagem base que não é da AWS. O exemplo de Dockerfile usa a imagem base da `buster`. O Dockerfile inclui o cliente de interface de runtime.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- Docker

- Node.js

Criar uma imagem de uma imagem base alternativa

Para criar uma imagem de contêiner de uma imagem base que não é da AWS

1. Crie um diretório para o projeto e depois mude para esse diretório.

```
mkdir example
cd example
```

2. Crie um novo projeto Node.js com o `npm`. Para aceitar as opções padrão fornecidas na experiência interativa, pressione `Enter`.

```
npm init
```

3. Crie um novo arquivo chamado `index.js`. É possível adicionar o exemplo de código de função a seguir ao arquivo para testes ou usar o seu próprio código.

   Example Manipulador CommonJS

```
exports.handler = async (event) => {
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
    return response;
};
```

4. Crie um novo Dockerfile. O Dockerfile a seguir usa uma imagem base de `buster` em vez de uma imagem base da AWS. O Dockerfile inclui o cliente de interface de runtime, o que torna a imagem compatível com o Lambda. O Dockerfile usa uma compilação em vários estágios. O primeiro estágio cria uma imagem de compilação, que é um ambiente Node.js padrão em que as dependências da função são instaladas. O segundo estágio cria uma imagem mais fina que inclui o código da função e suas dependências. Isso reduz o tamanho final da imagem.

   - Defina a propriedade `FROM` como o identificador da imagem base.
   - Use o comando `COPY` para copiar o código da função e as dependências do runtime.
   - Defina o `ENTRYPOINT` como o módulo em que você deseja que o contêiner do Docker seja executado quando for iniciado. Nesse caso, o módulo é o cliente de interface de runtime.

- Defina o argumento `CMD` como o manipulador de funções do 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. Crie a imagem do Docker com o comando docker build. O exemplo a seguir nomeia a imagem como `docker-image` e atribui a ela a tag `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> O comando especifica a opção `--platform linux/amd64` para garantir que seu contêiner seja compatível com o ambiente de execução do Lambda, independentemente da arquitetura da sua máquina de compilação. Se você pretende criar uma função do Lambda usando a arquitetura do conjunto de instruções ARM64, certifique-se de alterar o comando para usar a opção `--platform linux/arm64` em vez disso.

(Opcional) Teste a imagem localmente

Use o emulador de interface de runtime para testar a imagem localmente. Você pode compilar o emulador em sua imagem ou usar o procedimento a seguir instalá-lo na sua máquina local.

Para instalar o emulador de interface de runtime na sua máquina local

1. No diretório do projeto, execute o comando a seguir para baixar o emulador de interface de runtime (arquitetura x86-64) do GitHub e instalá-lo na sua máquina local.

   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
```

Para instalar o emulador arm64, substitua o URL do repositório do GitHub no comando anterior pelo seguinte:

```
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
```

Para instalar o emulador de arm64, substitua `$downloadLink` pelo seguinte:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Inicie a imagem do Docker com o comando docker run. Observe o seguinte:

   - `docker-image` é o nome da imagem e `test` é a tag.
   - `/usr/local/bin/npx aws-lambda-ric index.handler` é o ENTRYPOINT seguido pelo CMD do 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
```

Esse comando executa a imagem como um contêiner e cria um endpoint local em `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se você criou a imagem do Docker para a arquitetura do conjunto de instruções ARM64, certifique-se de usar a opção `--platform linux/arm64`, em vez de `--platform linux/amd64`.

3. Publique um evento no endpoint local.

Linux/macOS

No Linux e no MacOS, execute o seguinte comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

No PowerShell, execute o seguinte comando `Invoke-WebRequest`:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Obtenha o ID do contêiner.

```
docker ps
```

5. Use o comando docker kill para parar o contêiner. Nesse comando, substitua `3766c4ab331c` pelo ID do contêiner da etapa anterior.

```
docker kill 3766c4ab331c
```

Implantação da imagem

Para enviar a imagem ao Amazon ECR e criar a função do Lambda

1. Execute o comando get-login-password para autenticar a CLI do Docker no seu registro do Amazon ECR.

   - Defina o valor `--region` para a Região da AWS onde você deseja criar o repositório do Amazon ECR.
   - Substituir `111122223333` por seu ID da Conta da 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. Crie um repositório no Amazon ECR usando o 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**
>
> O repositório do Amazon ECR deve estar na mesma Região da AWS que a função do Lambda.

Se tiver êxito, você verá uma resposta como esta:

```
{
    "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. Copie o `repositoryUri` da saída na etapa anterior.
4. Execute o comando docker tag para aplicar uma tag na sua imagem local em seu repositório do Amazon ECR como a versão mais recente. Neste comando:

   - Substitua `docker-image:test` pelo nome e tag da sua imagem do Docker.
   - Substitua `<ECRrepositoryUri>` pelo `repositoryUri` que você copiou. Certifique-se de incluir `:latest` no final do URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Exemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Execute o comando docker push para implantar a imagem local no repositório do Amazon ECR. Certifique-se de incluir `:latest` no final do URI do repositório.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crie um perfil de execução para a função, caso ainda não tenha um. Você precisará do nome do recurso da Amazon (ARN) do perfil na próxima etapa.
7. Criar a função do Lambda. Em `ImageUri`, especifique o URI do repositório anterior. Certifique-se de incluir `:latest` no final do 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**
>
> É possível criar uma função usando uma imagem em uma conta da AWS diferente desde que a imagem esteja na mesma região da função do Lambda. Para ter mais informações, consulte Permissões entre contas do Amazon ECR.

8. Invoque a função.

```
aws lambda invoke --function-name hello-world response.json
```

Você obterá uma resposta parecida com esta:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Para ver a saída da função, verifique o arquivo `response.json`.

Para atualizar o código da função, você deve criar a imagem novamente, fazer upload da nova imagem no repositório do Amazon ECR e, em seguida, usar o comando update-function-code para implantar a imagem na função do Lambda.

O Lambda resolve a tag de imagem em um resumo de imagem específico. Isso significa que, se você apontar a tag de imagem que foi usada para implantar a função em uma nova imagem no Amazon ECR, o Lambda não atualizará automaticamente a função para usar a nova imagem. Para implantar a nova imagem na mesma função do Lambda, você deverá usar o comando `update-function-code`, mesmo que a tag da imagem no Amazon ECR permaneça a mesma.

Objeto de contexto do AWS Lambda em Node.js

Quando o Lambda executa a função, ele transmite um objeto de contexto para o handler. Esse objeto fornece métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de execução.

Métodos de contexto

- `getRemainingTimeInMillis()`: retorna o número de milissegundos restantes antes do tempo limite da execução.

Propriedades de contexto

- `functionName`: o nome da função do Lambda.
- `functionVersion`: a versão da função.
- `invokedFunctionArn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `memoryLimitInMB`: a quantidade de memória alocada para a função.
- `awsRequestId`: o identificador da solicitação de invocação.
- `logGroupName`: o grupo de logs da função.
- `logStreamName`: a transmissão de log para a instância da função.
- `identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
  - `cognitoIdentityId`: a identidade autenticada do Amazon Cognito.
  - `cognitoIdentityPoolId`: o grupo de identidades do Amazon Cognito que autorizou a invocação.
- `clientContext`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.
  - `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`: os valores personalizados que são definidos pela aplicação cliente.
- `callbackWaitsForEmptyEventLoop`: definido como falso para enviar a resposta imediatamente quando o retorno de chamada é executado, em vez de aguardar até que o loop de eventos do Node.js esteja vazio. Se for falso, todos os eventos pendentes continuarão a ser executados durante a próxima invocação.

O exemplo a seguir registra informações de contexto da função e retorna a localização dos logs.

Example Arquivo index.js

```
exports.handler = async function(event, context) {
  console.log('Remaining time: ', context.getRemainingTimeInMillis())
  console.log('Function name: ', context.functionName)
  return context.logStreamName
}
```

Registro em log da função do AWS Lambda em Node.js

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome e envia logs para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente do runtime do Lambda envia detalhes sobre cada invocação à transmissão de logs e transmite os logs e outras saídas do código de sua função. Para ter mais informações, consulte Usar logs do Amazon CloudWatch com o AWS Lambda.

Esta página descreve como produzir a saída de logs usando o código de sua função do Lambda ou acessar os logs usando a AWS Command Line Interface, o console do Lambda ou o console do CloudWatch.

Seções

Criar uma função que retorna logs

Para gerar os logs do código de função, você pode usar métodos no objeto do console ou qualquer biblioteca de logs que grave no `stdout` ou `stderr`. O exemplo a seguir registra em log os valores das variáveis de ambiente e o objeto do evento.

Example Arquivo index.js: registro em 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 do 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
```

O runtime do Node.js registra as linhas `START`, `END` e `REPORT` para cada invocação. Ele adiciona um carimbo de data e hora, o ID da solicitação e o nível de log em cada entrada registrada pela função. A linha do relatório fornece os detalhes a seguir.

RELATAR campos de dados de linha

- RequestId: o ID de solicitação exclusivo para a invocação.
- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.

- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o runtime levou para carregar a função e executar o código fora do método do handler.
- XRAY TraceId: para solicitações rastreadas, o ID de rastreamento do AWS X-Ray.
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Você pode visualizar logs no console do Lambda, no console do CloudWatch Logs ou na linha de comando.

Usar controles avançados de registro em log do Lambda com Node.js

Para dar mais controle sobre como os logs das funções são capturados, processados e consumidos, você pode configurar as seguintes opções de log para runtimes compatíveis do Node.js:

- Formato do log: selecione entre texto simples e formato JSON estruturado para os logs da sua função.
- Nível de log: para logs no formato JSON, escolha o nível de detalhe dos logs enviados pelo Lambda para o Amazon CloudWatch, como ERROR, DEBUG ou INFO.
- Grupo de logs: escolha o grupo de logs do CloudWatch para o qual sua função envia logs.

Para obter mais informações sobre essas opções de registro em log e instruções sobre como configurar a função para usá-las, consulte the section called “Configurar controles avançados de registro em log para a função do Lambda”.

Para usar as opções de formato de log e nível de log com as funções do Lambda para Node.js, consulte as orientações nas seções a seguir.

Usar logs JSON estruturados com Node.js

Se você selecionar JSON para o formato de log da função, o Lambda enviará a saída de logs usando os métodos do console `console.trace`, `console.debug`, `console.log`, `console.info`, `console.error` e `console.warn` para o CloudWatch como JSON estruturado. Cada objeto de log JSON contém pelo menos quatro pares de valores-chave com as seguintes chaves:

- `"timestamp"`: o horário em que a mensagem de log foi gerada.
- `"level"`: o nível de log atribuído à mensagem.
- `"message"`: o conteúdo da mensagem de log.

- `"requestId"`: o ID de solicitação exclusivo para invocar a função.

Dependendo do método de registro em log que a função usa, esse objeto JSON também pode conter pares de chaves adicionais. Por exemplo, se a função usa métodos do `console` para registrar em log objetos de erro usando vários argumentos, o objeto JSON conterá pares adicionais de valores de chave com as chaves `errorMessage`, `errorType` e `stackTrace`.

Se o código já usa outra biblioteca de logs, como o Powertools para AWS Lambda, para produzir logs em JSON estruturados, você não precisa fazer nenhuma alteração. O Lambda não codifica duas vezes nenhum log que já esteja codificado em JSON. Então, os logs de aplicações da função continuarão sendo capturados como antes.

Para obter mais informações sobre como usar o pacote Powertools for AWS Lambda para criar logs em JSON estruturados no runtime do Node.js, consulte the section called “Registro em log”.

Exemplos de saídas de log formatadas em JSON

Os exemplos a seguir mostram como várias saídas de log geradas usando os métodos do `console` com argumentos únicos e múltiplos são capturadas no CloudWatch Logs quando você define o formato de log da função como JSON.

O primeiro exemplo usa o método do `console.error` para gerar uma string simples.

Example Código de registro em log do Node.js

```
export const handler = async (event) => {
  console.error("This is a warning message");
  ...
}
```

Example Registro em log JSON

```
{
    "timestamp":"2023-11-01T00:21:51.358Z",
    "level":"ERROR",
    "message":"This is a warning message",
    "requestId":"93f25699-2cbf-4976-8f94-336a0aa98c6f"
}
```

Você também pode gerar mensagens de log estruturadas mais complexas usando argumentos únicos ou múltiplos com os métodos do `console`. No próximo exemplo, você usará `console.log`

para gerar dois pares de valores de chave usando um único argumento. Observe que o campo "message" no objeto JSON que o Lambda envia para o CloudWatch Logs não é convertido em strings.

Example Código de registro em log do Node.js

```
export const handler = async (event) => {
  console.log({data: 12.3, flag: false});
  ...
}
```

Example Registro em log JSON

```
{
    "timestamp": "2023-12-08T23:21:04.664Z",
    "level": "INFO",
    "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
    "message": {
        "data": 12.3,
        "flag": false
    }
}
```

No próximo exemplo, você usará novamente o método do `console.log` para criar uma saída de log. Dessa vez, o método usa dois argumentos, um mapa contendo dois pares de valores de chave e uma string identificadora. Observe que, nesse caso, como você forneceu dois argumentos, o Lambda converte o campo "message" em strings.

Example Código de registro em log do Node.js

```
export const handler = async (event) => {
  console.log('Some object - ', {data: 12.3, flag: false});
  ...
}
```

Example Registro em 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 }"
}
```

O Lambda atribui saídas geradas usando `console.log` no nível de log INFO.

O exemplo final mostra como objetos de erro podem ser enviados para o CloudWatch Logs usando os métodos do `console`. Observe que, quando são registrados objetos de erro usando vários argumentos, o Lambda adiciona os campos `errorMessage`, `errorType` e `stackTrace` à saída do log.

Example Código de registro em log do 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 Registro em 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)"
    ]
}
```

Ao registrar em log vários tipos de erro, os campos adicionais `errorMessage`, `errorType` e `stackTrace` são extraídos do primeiro tipo de erro fornecido ao método do `console`.

Usar bibliotecas cliente com logs de formato de métricas incorporadas (EMF) em JSON estruturado

A AWS fornece bibliotecas cliente de código aberto para o Node.js que podem ser usadas para criar logs de formato de métricas incorporadas (EMF). Se você já tiver funções que usam essas bibliotecas e alterar o formato de log da função para JSON, é possível que o CloudWatch não reconheça mais as métricas emitidas pelo código.

Se o seu código atualmente emite logs em EMF diretamente usando `console.log` ou o Powertools para AWS Lambda (TypeScript), o CloudWatch também não poderá analisá-los se você alterar o formato de log da função para JSON.

> **Important**
>
> Para garantir que os logs em EMF das funções continuem sendo analisados adequadamente pelo CloudWatch, atualize as bibliotecas de EMF e do Powertools para AWS Lambda para as versões mais recentes. Ao mudar para logs em formato JSON, também recomendamos que você realize testes a fim de garantir a compatibilidade com as métricas incorporadas da função. Se o seu código emitir logs em EMF diretamente usando `console.log`, altere o código para gerar essas métricas diretamente em `stdout`, conforme mostrado no código de exemplo a seguir.

Example código que emite métricas incorporadas para **`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")
```

Usar a filtragem em nível de log com Node.js

Para o AWS Lambda filtrar os logs de aplicação de acordo com o nível de log, a função precisa usar logs em formato JSON. Isso pode ser feito de duas maneiras:

- Crie saídas de log usando os métodos padrão do console e configure a função para usar logs em formato JSON. Dessa forma, o AWS Lambda filtra suas saídas de logs usando o par de valores-chave “nível” no objeto JSON descrito em the section called “Usar logs JSON estruturados com Node.js”. Para saber como configurar o formato de log da função, consulte the section called “Configurar controles avançados de registro em log para a função do Lambda”.
- Use outro método ou biblioteca de logs para criar logs JSON estruturados no código, de modo que incluam um par de valores-chave de “nível” para definir o nível da saída de log. Por exemplo, você pode usar o Powertools para AWS Lambda para gerar saídas de log JSON estruturado do seu código. Consulte the section called “Registro em log” para saber mais sobre como usar o Powertools com o runtime do Node.js.

  Para que o Lambda filtre os logs da função, você também precisa incluir um par de valores-chave `"timestamp"` na saída do log JSON. A hora deve ser especificada em um formato RFC 3339

válido de carimbo de data/hora. Se você não fornecer um carimbo de data/hora válido, o Lambda atribuirá ao log o nível INFO e adicionará um carimbo de data/hora.

Ao configurar a função para usar a filtragem em nível de log, você seleciona uma das seguintes opções para o nível de logs enviados pelo AWS Lambda para o CloudWatch Logs:

Nível de log	Uso padrão
TRACE (mais detalhes)	As informações mais detalhadas usadas para rastrear o caminho da execução do código
DEBUG	Informações detalhadas para depuração do sistema
INFO	Mensagens que registram a operação normal da função
WARN	Mensagens sobre possíveis erros que podem levar a um comportamento inesperado se não forem corrigidos
ERRO	Mensagens sobre problemas que impedem que o código funcione conforme o esperado
FATAL (menos detalhes)	Mensagens sobre erros graves que fazem a aplicação parar de funcionar

O Lambda envia logs do nível selecionado, e dos níveis inferiores, para o CloudWatch. Por exemplo, se você configurar um nível de log de WARN, o Lambda enviará logs correspondentes aos níveis WARN, ERROR e FATAL.

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda.

Se seu código puder ser testado no editor de Código incorporado, você encontrará logs nos resultados de execução. Ao usar o recurso de teste do console para invocar uma função, você encontrará Saída de log na seção Detalhes.

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a página Log groups (Grupos de log).
2. Escolha o grupo de logs de sua função (/aws/lambda/***nome-de-sua-função***).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma instância da sua função. Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos intrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- AWS CLI – Configuração rápida com `aws configure`

Você pode usar a AWS CLI para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

A seguinte saída deverá ser mostrada:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `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
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

A seguinte saída deverá ser mostrada:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e Ubuntu no Windows. Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa `sed` para remover as aspas do arquivo de saída e fica inativo por 15 segundos

para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 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 macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod -R 755 get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
}
```


Excluir logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ouConfigurar um período de retençãoapós o qual os logs são excluídos automaticamente.

Instrumentação do código Node.js no AWS Lambda

O Lambda se integra ao AWS X-Ray para ajudar você a rastrear, depurar e otimizar aplicações do Lambda. É possível usar o X-Ray para rastrear uma solicitação enquanto ela atravessa recursos na aplicação, o que pode incluir funções Lambda e outros produtos da AWS.

Para enviar dados de rastreamento ao X-Ray, você pode usar uma das duas bibliotecas SDK:

- AWS Distro for OpenTelemetry (ADOT): uma distribuição segura, pronta para produção e com suporte na AWS do SDK OpenTelemetry (OTel).
- AWS X-Ray SDK for Node.js: um SDK para geração e envio de dados de rastreamento ao X-Ray.

Cada um dos SDKs oferece maneiras de enviar dados de telemetria ao serviço do X-Ray. Em seguida, é possível usar o X-Ray para visualizar, filtrar e obter insights sobre as métricas de performance da aplicação para identificar problemas e oportunidades de otimização.

> **Important**
>
> Os SDKs do X-Ray e do Powertools para AWS Lambda fazem parte de uma solução de instrumentação totalmente integrada oferecida pela AWS. As camadas do Lambda para ADOT fazem parte de um padrão em todo o setor para instrumentação de rastreamento que coleta mais dados em geral, mas pode não ser adequado para todos os casos de uso. É possível implementar o rastreamento de ponta a ponta no X-Ray usando ambas as soluções. Para saber mais sobre como escolher entre elas, consulte Como escolher entre os SDKs do AWS Distro para OpenTelemetry e do X-Ray.

Seções

Usar o ADOT para instrumentar funções do Node.js

O ADOT fornece camadas do Lambda totalmente gerenciadas que empacotam tudo o que você precisa para coletar dados de telemetria usando o SDK do OTel. Ao consumir essa camada, é possível instrumentar suas funções Lambda sem precisar modificar nenhum código de função. Você também pode configurar sua camada para fazer a inicialização personalizada do OTel. Para obter mais informações, consulte Custom configuration for the ADOT Collector on Lambda (Configuração personalizada para o ADOT Collector no Lambda) na documentação do ADOT.

Para runtimes Node.js, você pode adicionar a camada do Lambda gerenciada pela AWS para ADOT Javascript a fim de instrumentar suas funções automaticamente. Para obter instruções detalhadas sobre como adicionar essa camada, consulte Suporte do AWS Distro for OpenTelemetry Lambda para JavaScript, na documentação do ADOT.

Usar o SDK do X-Ray para instrumentar suas funções Node.js

Para registrar detalhes sobre as chamadas feitas pela sua função do Lambda para outros recursos na sua aplicação, você também pode usar o AWS X-Ray SDK for Node.js. Para obter o SDK, adicione o pacote `aws-xray-sdk-core` às dependências do aplicativo.

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"
  }
}
```

Para instrumentar clientes do AWS SDK no AWS SDK for JavaScript v3, envolva a instância do cliente com o método `captureAWSv3Client`.

Example blank-nodejs/function/index.js: rastrear um cliente do 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 => {
  ...
```

O runtime do Lambda define algumas variáveis de ambiente para configurar o X-Ray SDK. Por exemplo, o Lambda define `AWS_XRAY_CONTEXT_MISSING` como `LOG_ERROR` para evitar lançar erros de runtime no X-Ray SDK. Para definir uma estratégia de contexto ausente personalizada, substitua a variável de ambiente na configuração da função para que ela não tenha valores e, depois, defina a estratégia de contexto ausente de forma programática.

Example Exemplo de código de inicialização

```
const AWSXRay = require('aws-xray-sdk-core');

// Configure the context missing strategy to do nothing
AWSXRay.setContextMissingStrategy(() => {});
```

Para ter mais informações, consulte the section called “Configurar variáveis de ambiente”.

Depois de adicionar as dependências corretas e fazer as devidas mudanças de código, ative o rastreamento na configuração da sua função usando o console do Lambda ou a API.

Ativar o rastreamento com o console do Lambda

Para alternar o rastreamento ativo na sua função do Lambda usando o console, siga as etapas abaixo:

Para ativar o rastreamento ativo

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.

3. Escolha Configuration (Configuração) e depois Monitoring and operations tools (Ferramentas de monitoramento e operações).
4. Selecione a opção Editar.
5. Em X-Ray, ative a opção Active tracing (Rastreamento ativo).
6. Escolha Salvar.

Ativar o rastreamento com a API do Lambda

Configure o rastreamento na sua função do Lambda com aAWS CLI ou o AWS SDK, usando as seguintes operações de API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration \
--function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Ativar o rastreamento com o AWS CloudFormation

Para ativar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example function-inline.yml: configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
```

```
      ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example template.yml: configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretar um rastreamento do X-Ray

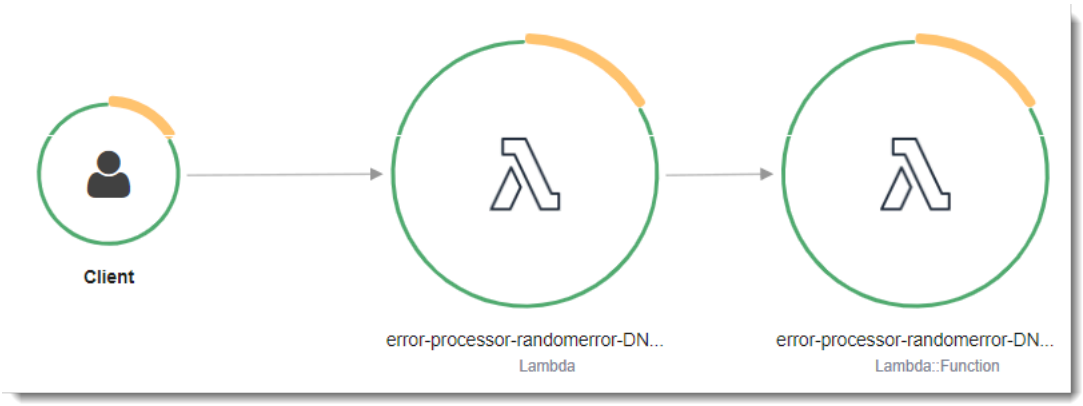
Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você ativa o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à função de execução da função. Caso contrário, adicione a política AWSXRayDaemonWriteAccess à função de execução.

Após configurar o rastreamento ativo, você pode observar solicitações específicas por meio da aplicação. O grafo de serviço do X-Ray exibe informações sobre sua aplicação e todos os componentes. A imagem a seguir demonstra uma aplicação com duas funções. A função principal processa eventos e, às vezes, retorna erros. A segunda função de cima para baixo processa erros que aparecem no primeiro grupo de logs e usa o AWS SDK para chamar o X-Ray, o Amazon Simple Storage Service (Amazon S3) e o Amazon CloudWatch Logs.

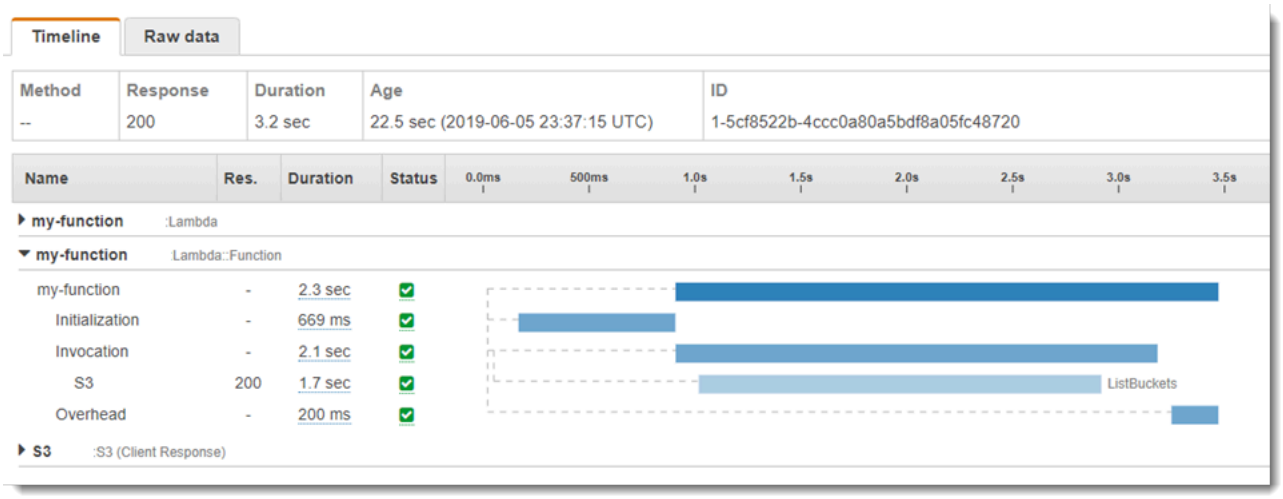
O X-Ray não rastreia todas as solicitações para sua aplicação. O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento seja eficiente, enquanto ainda fornece uma amostra representativa das solicitações. A taxa de amostragem é uma solicitação por segundo e 5% de solicitações adicionais.

> **Note**
>
> Não é possível configurar a taxa de amostragem do X-Ray para suas funções.

No X-Ray, um rastreamento registra informações sobre uma solicitação que é processada por um ou mais serviços. O Lambda registra dois segmentos por rastreamento, o que cria dois nós no gráfico de serviços. A imagem a seguir destaca esses dois nós:



O primeiro nó à esquerda representa o serviço do Lambda, que recebe a solicitação de invocação. O segundo nó representa a sua função do Lambda específica. O exemplo a seguir mostra um rastreamento com esses dois segmentos. Ambos têm o nome my-function, mas um tem a origem `AWS::Lambda` e o outro, a origem `AWS::Lambda::Function`. Se o segmento `AWS::Lambda` mostrar um erro, o serviço Lambda teve um problema. Se o segmento `AWS::Lambda::Function` mostrar um erro, sua função teve um problema.



Este exemplo expande o segmento `AWS::Lambda::Function` para mostrar seus três subsegmentos:

- Inicialização: representa o tempo gasto carregando a função e executando o código de inicialização. Esse subsegmento aparece somente para o primeiro evento que cada instância da função processa.
- Invocação: representa o tempo gasto na execução do código do manipulador.
- Sobrecarga: representa o tempo gasto pelo runtime do Lambda preparando-se para lidar com o próximo evento.

Você também pode instrumentar clientes HTTP, registrar consultas SQL e criar subsegmentos personalizados com anotações e metadados. Para obter mais informações, consulte AWS X-Ray SDK for Node.js no Guia do desenvolvedor do AWS X-Ray.

> **Definição de preço**
>
> Você pode usar o rastreamento do X-Ray gratuitamente todos os meses até determinado limite como parte do nível gratuito da AWS. Além do limite, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter mais informações, consulte Preços do AWS X-Ray.

Armazenar dependências de runtime em uma camada (SDK do X-Ray)

Se você usar o X-Ray SDK para instrumentar os clientes do AWS SDK com seu código de função, seu pacote de implantação poderá se tornar bastante grande. Para evitar o upload de dependências de runtime todas as vezes que você atualizar seu código de função, empacote o SDK do X-Ray em uma camada do Lambda.

O exemplo a seguir mostra um recurso `AWS::Serverless::LayerVersion` que armazena o AWS X-Ray SDK for Node.js.

Example template.yml: camada de dependências

```
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
```

Com essa configuração, você atualizará a camada de biblioteca somente se alterar as dependências de runtime. Já que o pacote de implantação de função inclui apenas o seu código, isso pode ajudar a reduzir o tempo de upload.

A criação de uma camada de dependências requer alterações de compilação para gerar o arquivo da camada antes da implantação. Para obter um exemplo funcional, consulte o aplicativo de exemplo blank-nodejs .

Criar funções Lambda com TypeScript

Você pode usar o runtime Node.js para executar o código TypeScript no AWS Lambda. Como o Node.js não executa o código TypeScript de modo nativo, você deve primeiro transcompilar seu código TypeScript em JavaScript. Em seguida, use os arquivos JavaScript para implantar o código de função no Lambda. Seu código é executado em um ambiente que inclui o AWS SDK for JavaScript, com as credenciais de uma função do AWS Identity and Access Management (IAM) que você gerencia. Para saber mais sobre as versões do SDK incluídas nos runtimes do Node.js, consulte the section called “Versões do SDK incluídas no runtime”.

O Lambda oferece suporte aos runtimes Node.js a seguir.

Node.js

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
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 de junho de 2024	28 de fevereiro de 2025	31 de março de 2025

Tópicos

Configurar seu ambiente de desenvolvimento de TypeScript

Use um ambiente de desenvolvimento integrado (IDE) local, um editor de texto ou o AWS Cloud9 para gravar o código de função TypeScript. Não é possível criar código TypeScript no console do Lambda.

Para transcompilar seu código TypeScript, configure um compilador como o esbuild ou o compilador de TypeScript da Microsoft (`tsc`), que é empacotado com a distribuição do TypeScript. Você pode usar o AWS Serverless Application Model (AWS SAM) ou o AWS Cloud Development Kit (AWS CDK) para simplificar a criação e a implantação de código TypeScript. Ambas as ferramentas usam esbuild para transcompilar código TypeScript em JavaScript.

Ao usar o esbuild, considere o seguinte:

- Existem várias advertências de TypeScript.
- É necessário definir as configurações de transcompilação do TypeScript para corresponder ao runtime Node.js que você pretende usar. Para obter mais informações, consulte Target (Destino) na documentação do esbuild. Para ver um exemplo de arquivo tsconfig.json que mostra como direcionar uma versão específica do Node.js compatível com o Lambda, consulte o repositório GitHub do TypeScript.
- O esbuild não executa verificações de tipo. Para verificar tipos, use o compilador `tsc`. Execute `tsc -noEmit` ou adicione um parâmetro `"noEmit"` no arquivo tsconfig.json, conforme mostrado no exemplo a seguir. Isso configura `tsc` para não emitir arquivos JavaScript. Após verificar os tipos, use esbuild para converter os arquivos TypeScript em 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"]
}
```

Definir o manipulador de função do Lambda em TypeScript

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. A função é executada até que o manipulador retorne uma resposta, seja encerrado ou atinja o tempo limite.

Example Manipulador TypeScript

Essa função de exemplo registra o conteúdo do objeto de evento e retorna a localização dos logs. Observe o seguinte:

- Antes de usar esse código em uma função do Lambda, você deve adicionar o pacote @types / aws-lambda como uma dependência de desenvolvimento. Esse pacote contém as definições de tipo para o Lambda. Quando `@types/aws-lambda` está instalado, a instrução `import` (`import ... from 'aws-lambda'`) importa as definições de tipo. Ela não importa o pacote `aws-lambda` do NPM, que é uma ferramenta de terceiros não relacionada. Para obter mais informações, consulte aws-lambda no repositório DefinitelyTyped do GitHub.
- O manipulador neste exemplo é um módulo ES e deve ser designado como tal em seu arquivo `package.json` ou usando a extensão de arquivo `.mjs`. Para obter mais informações, consulte Designar um manipulador de funções como módulo 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;
};
```

O runtime transmite argumentos para o método do handler. O primeiro argumento é o objeto `event`, que contém informações do chamador. O invocador passa essas informações como uma string no formato JSON ao chamar Invoke, e o runtime as converte em um objeto. Quando um serviço da AWS invoca a sua função, a estrutura do evento varia de acordo com o serviço. Com o TypeScript, recomendamos o uso de anotações de tipo para o objeto de evento. Para ter mais informações, consulte Usar tipos para o objeto de evento.

O segundo argumento é o objeto de contexto, que contém informações sobre a invocação, a função e o ambiente de execução. No exemplo anterior, a função obtém o nome do fluxo de logs do objeto de contexto e o retorna para o invocador.

Também é possível usar um argumento de retorno de chamada, que é uma função que você pode chamar nos manipuladores não assíncronos para enviar uma resposta. Recomendamos o uso de async/await em vez dos retornos de chamada. Async/await oferece melhor legibilidade, tratamento de erros e eficiência. Para obter mais informações sobre as diferenças entre async/await e retornos de chamada, consulte Usar retornos de chamada.

Usar async/await

Se o código realizar uma tarefa assíncrona, use o padrão async/await para garantir que a execução do manipulador seja finalizada. Async/await é uma forma concisa e legível de escrever código assíncrono em Node.js, sem a necessidade de retornos de chamada aninhados ou promessas de encadeamento. Com async/await, é possível escrever um código que seja lido como código síncrono e, ao mesmo tempo, seja assíncrono e sem bloqueio.

A palavra-chave `async` marca uma função como assíncrona, e a palavra-chave `await` pausa a execução da função até que uma `Promise` seja resolvida.

Example Função TypeScript: assíncrona

Este exemplo usa `fetch`, que está disponível no runtime `nodejs18.x`. Observe o seguinte:

- Antes de usar esse código em uma função do Lambda, você deve adicionar o pacote @types / aws-lambda como uma dependência de desenvolvimento. Esse pacote contém as definições de tipo para o Lambda. Quando `@types/aws-lambda` está instalado, a instrução `import` (`import ... from 'aws-lambda'`) importa as definições de tipo. Ela não importa o pacote `aws-lambda` do NPM, que é uma ferramenta de terceiros não relacionada. Para obter mais informações, consulte aws-lambda no repositório DefinitelyTyped do GitHub.
- O manipulador neste exemplo é um módulo ES e deve ser designado como tal em seu arquivo `package.json` ou usando a extensão de arquivo `.mjs`. Para obter mais informações, consulte Designar um manipulador de funções como módulo 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',
            }),
        };
    }
};
```

Usar retornos de chamada

Recomendamos usar async/await para declarar o manipulador da função em vez de usar retornos de chamada. Async/await é a melhor opção por vários motivos:

- Legibilidade: o código async/await é mais fácil de ler e entender do que o código de retorno de chamada, que pode logo se tornar difícil de seguir e transformar o retorno de chamada em um tormento.
- Depuração e tratamento de erros: pode ser difícil depurar código baseado em retorno de chamada. A pilha de chamadas pode se tornar difícil de acompanhar, e os erros podem ser facilmente ignorados. Com async/await, você pode usar blocos try/catch para manipular erros.
- Eficiência: retornos de chamada muitas vezes exigem alternância entre diferentes partes do código. O async/await pode reduzir o número de alternâncias de contexto, resultando em um código mais eficiente.

Quando você usa retornos de chamada no manipulador, a função continua em execução até que o loop de evento esteja vazio ou o tempo da função se esgote. A resposta não é enviada para o chamador até que todas as tarefas de loop de evento estejam concluídas. Se a função expirar, um erro será retornado. É possível configurar o runtime para enviar a resposta imediatamente, definindo context.callbackWaitsForEmptyEventLoop como false.

A função de retorno de chamada usa dois argumentos: um `Error` e uma resposta. O objeto de resposta deve ser compatível com `JSON.stringify`.

Example Função TypeScript com retorno de chamada

Essa função de exemplo recebe um evento do Amazon API Gateway, registra o evento e os objetos de contexto e, em seguida, retorna uma resposta ao API Gateway. Observe o seguinte:

- Antes de usar esse código em uma função do Lambda, você deve adicionar o pacote @types / aws-lambda como uma dependência de desenvolvimento. Esse pacote contém as definições de tipo para o Lambda. Quando `@types/aws-lambda` está instalado, a instrução `import` (`import ... from 'aws-lambda'`) importa as definições de tipo. Ela não importa o pacote `aws-lambda` do NPM, que é uma ferramenta de terceiros não relacionada. Para obter mais informações, consulte aws-lambda no repositório DefinitelyTyped do GitHub.
- O manipulador neste exemplo é um módulo ES e deve ser designado como tal em seu arquivo `package.json` ou usando a extensão de arquivo `.mjs`. Para obter mais informações, consulte Designar um manipulador de funções como módulo 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',
        }),
    });
};
```

Usar tipos para o objeto de evento

Recomendamos não usar qualquer tipo para os argumentos do manipulador e tipo de retorno, pois você perde a capacidade de verificar tipos. Em vez disso, gere um evento usando o comando da CLI evento gerador local sam do AWS Serverless Application Model ou use uma definição de código aberto do pacote @types/aws-lambda.

Gerar um evento usando o comando sam local generate-event

1. Gerar um evento proxy do Amazon Simple Storage Service (Amazon S3).

```
sam local generate-event s3 put >> S3PutEvent.json
```

2. Usar o utilitário quicktype para gerar definições de tipo do arquivo S3PutEvent.json.

```
npm install -g quicktype
quicktype S3PutEvent.json -o S3PutEvent.ts
```

3. Use os tipos gerados no código.

```
import { S3PutEvent } from './S3PutEvent';

export const lambdaHandler = async (event: S3PutEvent): Promise<void> => {
  event.Records.map((record) => console.log(record.s3.object.key));
};
```

Gerar um evento usando uma definição de código aberto do pacote @types/aws-lambda

1. Adicione o pacote @types/aws-lambda como uma dependência de desenvolvimento.

```
npm install -D @types/aws-lambda
```

2. Use os tipos no código.

```
import { S3Event } from "aws-lambda";

export const lambdaHandler = async (event: S3Event): Promise<void> => {
  event.Records.map((record) => console.log(record.s3.object.key));
};
```

Implemente TypeScript código transpilado no Lambda com arquivos de arquivos.zip

Antes de poder implantar TypeScript código emAWS Lambda, você precisa transpilá-lo em. JavaScript Esta página explica três maneiras de criar e implantar TypeScript código no Lambda com arquivos.zip:

- Usar o AWS Serverless Application Model (AWS SAM)
- Usar o AWS Cloud Development Kit (AWS CDK)
- Usar a AWS Command Line Interface (AWS CLI) e o esbuild

AWS SAMe AWS CDK simplifique a criação e a implantação de TypeScript funções. A especificação do modelo do AWS SAM fornece uma sintaxe simples e clara para descrever as funções, APIs, permissões, configurações e eventos do Lambda que compõem sua aplicação sem servidor. O AWS CDK permite criar aplicações confiáveis, escaláveis e com bom custo-benefício na nuvem com o considerável poder expressivo de uma linguagem de programação. O AWS CDK destina-se a usuários da AWS bastante experientes ou moderadamente experientes. Tanto o AWS CDK quanto o AWS SAM usam esbuild para transpilar TypeScript o código em. JavaScript

Usando AWS SAM para implantar TypeScript código no Lambda

Siga as etapas abaixo para baixar, criar e implantar uma amostra do TypeScript aplicativo Hello World usando AWS SAM o. Esta aplicação implementa um backend básico da API. Consiste em um endpoint do Amazon API Gateway e uma função Lambda. Ao enviar uma solicitação GET ao endpoint do API Gateway, você invoca a função Lambda. A função retorna uma mensagem `hello world`.

> **Note**
>
> AWS SAMusa o esbuild para criar funções Lambda do Node.js a TypeScript partir do código. o suporte do esbuild está atualmente em pré-visualização pública. Durante a pré-visualização pública, o suporte ao esbuild pode estar sujeito a alterações incompatíveis com versões anteriores.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- AWS CLI versão 2
- CLI do AWS SAM, versão 1.75 ou posterior
- Node.js 18.x

Implantar uma aplicação de exemplo do AWS SAM

1. Inicialize o aplicativo usando o TypeScript modelo Hello World.

```
sam init --app-template hello-world-typescript --name sam-app --package-type Zip --
runtime nodejs18.x
```

2. (Opcional) A aplicação de exemplo inclui configurações para ferramentas bastante usadas, como ESLint para linting de código e Jest para testes unitários. Para executar comandos lint e de teste:

```
cd sam-app/hello-world
npm install
npm run lint
npm run test
```

3. Crie a aplicação.

```
cd sam-app
sam build
```

4. Implante o aplicativo.

```
sam deploy --guided
```

5. Siga as instruções na tela. Para aceitar as opções padrão fornecidas na experiência interativa, responda com `Enter`.
6. A saída mostra o endpoint para a API REST. Abra o endpoint em um navegador para testar a função. Você deverá visualizar esta resposta:

```
{"message":"hello world"}
```

7. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
sam delete
```

Usando o AWS CDK para implantar TypeScript código no Lambda

Siga as etapas abaixo para criar e implantar um TypeScript aplicativo de amostra usando AWS CDK o. Esta aplicação implementa um backend básico da API. Consiste em um endpoint do API Gateway e uma função Lambda. Ao enviar uma solicitação GET ao endpoint do API Gateway, você invoca a função Lambda. A função retorna uma mensagem `hello world`.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- AWS CLI versão 2
- AWS CDK versão 2
- Node.js 18.x
- Docker ou esbuild

Implantar uma aplicação de exemplo do AWS CDK

1. Crie um diretório de projeto para sua aplicação.

```
mkdir hello-world
cd hello-world
```

2. Inicialize a aplicação.

```
cdk init app --language typescript
```

3. Adicione o pacote @types/aws-lambda como uma dependência de desenvolvimento. Esse pacote contém as definições de tipo para o Lambda.

```
npm install -D @types/aws-lambda
```

4. Abra o diretório lib. Você deve ver um arquivo chamado hello-world-stack.ts. Crie dois novos arquivos neste diretório: hello-world.function.ts e hello-world.ts.
5. Abra o hello-world.function.ts e adicione o código a seguir ao arquivo. Este é o código da função Lambda.

> **Note**
>
> A instrução `import` importa as definições de tipo de @types/aws-lambda. Ela não importa o pacote `aws-lambda` do NPM, que é uma ferramenta de terceiros não relacionada. Para obter mais informações, consulte aws-lambda no repositório. 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. Abra o hello-world.ts e adicione o código a seguir ao arquivo. Ele contém a NodejsFunction construção, que cria a função Lambda, e a LambdaRestApi construção, que cria a 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,
```

```
    });
  }
}
```

O constructo `NodejsFunction` assume o seguinte, por padrão:

- O manipulador de funções chama-se `handler`.
- O arquivo .ts que contém o código da função (hello-world.function.ts) está no mesmo diretório que o arquivo .ts que contém o constructo (hello-world.ts). O constructo usa o ID do constructo ("hello-world") e o nome de arquivo do manipulador do Lambda ("function") para encontrar o código da função. Por exemplo, se o código da função estiver em um arquivo chamado hello-world.my-function.ts, o arquivo hello-world.ts deverá referenciar o código da função deste modo:

```
const helloFunction = new NodejsFunction(this, 'my-function');
```

É possível alterar esse comportamento e configurar outros parâmetros do esbuild. Para mais informações, consulte Configuring esbuild (Configurar esbuild) na referência de API do AWS CDK.

7. Abra hello-world-stack.ts. Este é o código que define sua pilha do AWS CDK. Substitua o código pelo seguinte:

```
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. a partir do diretório do `hello-world` que contém seu arquivo `cdk.json`, implante sua aplicação.

```
cdk deploy
```

9. O AWS CDK cria e empacota a função Lambda usando esbuild e implanta a função no runtime do Lambda. A saída mostra o endpoint para a API REST. Abra o endpoint em um navegador para testar a função. Você deverá visualizar esta resposta:

   ```
   {"message":"hello world"}
   ```

   Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

Usando o AWS CLI e o esbuild para implantar TypeScript código no Lambda

O exemplo a seguir demonstra como transpilar e TypeScript implantar código no Lambda usando esbuild e o. esbuild produz um arquivo com todas as AWS CLI dependências. JavaScript Este é o único arquivo que você precisa adicionar ao arquivo.zip.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- AWS CLI versão 2
- Node.js 18.x
- Uma função de execução para a função Lambda
- Para usuários do Windows, um utilitário de arquivo zip, como o 7zip.

Implantar uma função de exemplo

1. Em sua máquina local, crie um diretório de projeto para sua nova função.
2. Crie um novo projeto Node.js com npm ou um gerenciador de pacotes de sua escolha.

   ```
   npm init
   ```

3. Adicione pacotes @types/aws-lambda e esbuild como dependência de desenvolvimento. O pacote `@types/aws-lambda` contém as definições de tipo para o Lambda.

   ```
   npm install -D @types/aws-lambda esbuild
   ```

4. Crie um novo arquivo chamado index.ts. Adicione o código a seguir ao novo arquivo. Este é o código da função Lambda. A função retorna uma mensagem `hello world`. A função não cria recursos do API Gateway.

> **Note**
>
> A instrução `import` importa as definições de tipo de @types/aws-lambda. Ela não importa o pacote `aws-lambda` do NPM, que é uma ferramenta de terceiros não relacionada. Para obter mais informações, consulte aws-lambda no repositório. 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. Adicione um script de compilação ao arquivo package.json. Isso configura o esbuild para criar automaticamente o pacote de implantação .zip. Para obter mais informações, consulte Build scripts (Scripts de compilação) na documentação do 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

Neste exemplo, o comando `"postbuild"` usa o utilitário 7zip para criar o arquivo .zip. Use seu próprio utilitário zip do Windows preferido e modifique o comando conforme necessário.

```
"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. Construa o pacote.

```
npm run build
```

7. Crie uma função Lambda usando o pacote de implantação .zip. Substitua o texto em destaque pelo nome do recurso da Amazon (ARN) da função de execução.

```
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. Execute um evento de teste para confirmar que a função retorna a resposta a seguir. Para invocar essa função usando o API Gateway, crie e configure uma API REST.

```
{
  "statusCode": 200,
  "body": "{\"message\":\"hello world\"}"
}
```

Implantar código TypeScript transcompilado no Lambda com imagens de contêiner

Você pode implantar seu código TypeScript em uma função do AWS Lambda como imagem de contêiner Node.js. A AWS fornece imagens base para Node.js para ajudar você a criar a imagem do contêiner. Essas imagens base são pré-carregadas com um runtime de linguagem e outros componentes necessários para executar a imagem no Lambda. A AWS fornece um Dockerfile para cada uma das imagens base para ajudar a criar sua imagem de contêiner.

Se você usa uma imagem base pertencente a uma comunidade ou empresa privada, é necessário adicionar um cliente de interface de runtime (RIC) Node.js à imagem base para torná-la compatível com o Lambda.

O Lambda oferece um emulador de interface de runtime para testes locais. As imagens base da AWS para o Node.js incluem o emulador de interface de runtime. Caso use uma imagem base alternativa, como uma imagem Alpine Linux ou Debian, você poderá criar o emulador na sua imagem ou instalá-lo na máquina local.

Usar uma imagem base Node.js para criar e empacotar código da função TypeScript

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- Docker
- Node.js 18.x

Criação de uma imagem a partir de uma imagem base

Para criar uma imagem a partir de uma imagem base da AWS para o Lambda

1. Em sua máquina local, crie um diretório de projeto para sua nova função.
2. Crie um novo projeto Node.js com `npm` ou um gerenciador de pacotes de sua escolha.

```
npm init
```

3. Adicione pacotes @types/aws-lambda e esbuild como dependência de desenvolvimento. O pacote `@types/aws-lambda` contém as definições de tipo para o Lambda.

```
npm install -D @types/aws-lambda esbuild
```

4. Adicione um script de construção ao arquivo `package.json`.

```
  "scripts": {
  "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --
target=es2020 --outfile=dist/index.js"
}
```

5. Crie um novo arquivo chamado `index.ts`. Adicione o código de exemplo a seguir ao novo arquivo. Este é o código da função do Lambda. A função retorna uma mensagem `hello world`.

> **Note**
>
> A instrução `import` importa as definições de tipo de @types/aws-lambda. Ela não importa o pacote `aws-lambda` do NPM, que é uma ferramenta de terceiros não relacionada. Para obter mais informações, consulte aws-lambda no repositório DefinitelyTyped do 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. Crie um novo Dockerfile com a seguinte configuração:

   - Defina a propriedade `FROM` como o URI da imagem base.

- Defina o argumento `CMD` para especificar o manipulador de funções do Lambda.

Example Dockerfile

O Dockerfile a seguir usa uma compilação em várias etapas. A primeira etapa transcompila o código TypeScript em JavaScript. A segunda etapa produz uma imagem de contêiner contendo somente arquivos JavaScript e dependências de produção.

```
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. Crie a imagem do Docker com o comando docker build. O exemplo a seguir nomeia a imagem como `docker-image` e atribui a ela a tag `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> O comando especifica a opção `--platform linux/amd64` para garantir que seu contêiner seja compatível com o ambiente de execução do Lambda, independentemente da arquitetura da sua máquina de compilação. Se você pretende criar uma função do Lambda usando a arquitetura do conjunto de instruções ARM64, certifique-se de alterar o comando para usar a opção `--platform linux/arm64` em vez disso.

(Opcional) Teste a imagem localmente

1. Inicie a imagem do Docker com o comando docker run. Neste exemplo, `docker-image` é o nome da imagem e `test` é a tag.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Esse comando executa a imagem como um contêiner e cria um endpoint local em `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se você criou a imagem do Docker para a arquitetura do conjunto de instruções ARM64, certifique-se de usar a opção `--platform linux/arm64`, em vez de `--platform linux/amd64`.

2. Em uma nova janela de terminal, publique um evento no endpoint local.

   Linux/macOS

   No Linux e no MacOS, execute o seguinte comando `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   No PowerShell, execute o seguinte comando `Invoke-WebRequest`:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Obtenha o ID do contêiner.

```
docker ps
```

4. Use o comando docker kill para parar o contêiner. Nesse comando, substitua `3766c4ab331c` pelo ID do contêiner da etapa anterior.

```
docker kill 3766c4ab331c
```

Implantação da imagem

Para enviar a imagem ao Amazon ECR e criar a função do Lambda

1. Execute o comando get-login-password para autenticar a CLI do Docker no seu registro do Amazon ECR.
   - Defina o valor `--region` para a Região da AWS onde você deseja criar o repositório do Amazon ECR.
   - Substituir `111122223333` por seu ID da Conta da 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. Crie um repositório no Amazon ECR usando o 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**
>
> O repositório do Amazon ECR deve estar na mesma Região da AWS que a função do Lambda.

Se tiver êxito, você verá uma resposta como esta:

```
{
    "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. Copie o `repositoryUri` da saída na etapa anterior.
4. Execute o comando docker tag para aplicar uma tag na sua imagem local em seu repositório do Amazon ECR como a versão mais recente. Neste comando:

   - Substitua `docker-image:test` pelo nome e tag da sua imagem do Docker.
   - Substitua `<ECRrepositoryUri>` pelo `repositoryUri` que você copiou. Certifique-se de incluir `:latest` no final do URI.

   ```
   docker tag docker-image:test <ECRrepositoryUri>:latest
   ```

   Exemplo:

   ```
   docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
   world:latest
   ```

5. Execute o comando docker push para implantar a imagem local no repositório do Amazon ECR. Certifique-se de incluir `:latest` no final do URI do repositório.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crie um perfil de execução para a função, caso ainda não tenha um. Você precisará do nome do recurso da Amazon (ARN) do perfil na próxima etapa.
7. Criar a função do Lambda. Em `ImageUri`, especifique o URI do repositório anterior. Certifique-se de incluir `:latest` no final do 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**
>
> É possível criar uma função usando uma imagem em uma conta da AWS diferente desde que a imagem esteja na mesma região da função do Lambda. Para ter mais informações, consulte Permissões entre contas do Amazon ECR.

8. Invoque a função.

```
aws lambda invoke --function-name hello-world response.json
```

Você obterá uma resposta parecida com esta:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Para ver a saída da função, verifique o arquivo `response.json`.

Para atualizar o código da função, você deve criar a imagem novamente, fazer upload da nova imagem no repositório do Amazon ECR e, em seguida, usar o comando update-function-code para implantar a imagem na função do Lambda.

O Lambda resolve a tag de imagem em um resumo de imagem específico. Isso significa que, se você apontar a tag de imagem que foi usada para implantar a função em uma nova imagem no

Amazon ECR, o Lambda não atualizará automaticamente a função para usar a nova imagem. Para implantar a nova imagem na mesma função do Lambda, você deverá usar o comando `update-function-code`, mesmo que a tag da imagem no Amazon ECR permaneça a mesma.

AWS Lambdaobjeto de contexto em TypeScript

Quando o Lambda executa a função, ele transmite um objeto de contexto para o handler. Esse objeto fornece métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de execução.

Métodos de contexto

- `getRemainingTimeInMillis()`: retorna o número de milissegundos restantes antes do tempo limite da execução.

Propriedades de contexto

- `functionName`: o nome da função do Lambda.
- `functionVersion`: a versão da função.
- `invokedFunctionArn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `memoryLimitInMB`: a quantidade de memória alocada para a função.
- `awsRequestId`: o identificador da solicitação de invocação.
- `logGroupName`: o grupo de logs da função.
- `logStreamName`: a transmissão de log para a instância da função.
- `identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
  - `cognitoIdentityId`: a identidade autenticada do Amazon Cognito.
  - `cognitoIdentityPoolId`: o grupo de identidades do Amazon Cognito que autorizou a invocação.
- `clientContext`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.
  - `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`: os valores personalizados que são definidos pela aplicação cliente.
- `callbackWaitsForEmptyEventLoop`: definido como falso para enviar a resposta imediatamente quando o retorno de chamada é executado, em vez de aguardar até que o loop de eventos do Node.js esteja vazio. Se for falso, todos os eventos pendentes continuarão a ser executados durante a próxima invocação.

Você pode usar o pacote npm @types/aws-lambda para trabalhar com o objeto de contexto.

Example arquivo index.ts

O exemplo a seguir registra informações de contexto da função e retorna a localização dos logs.

> **Note**
>
> Antes de usar esse código em uma função do Lambda, você deve adicionar o pacote @types/aws-lambda como uma dependência de desenvolvimento. Esse pacote contém as definições de tipo para o Lambda. Quando `@types/aws-lambda` está instalado, a instrução `import` (`import ... from 'aws-lambda'`) importa as definições de tipo. Ela não importa o pacote `aws-lambda` do NPM, que é uma ferramenta de terceiros não relacionada. Para obter mais informações, consulte aws-lambda no repositório. 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 registro de funções em TypeScript

O AWS Lambda monitora automaticamente as funções do Lambda e envia entradas de logs para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente de runtime do Lambda envia detalhes sobre cada invocação e outras saídas do código da função para o fluxo de logs. Para obter mais informações sobre o CloudWatch Logs, consulte Usar logs do Amazon CloudWatch com o AWS Lambda.

Para exibir logs do código de função, você pode usar métodos no objeto de console. Para um log mais detalhado, você pode usar qualquer biblioteca de registro em log que grava em `stdout` ou `stderr`.

Seções

Ferramentas e bibliotecas

O Powertools para AWS Lambda (TypeScript) é um kit de ferramentas para desenvolvedores para implementar as práticas recomendadas da tecnologia sem servidor e aumentar a velocidade do desenvolvedor. O utilitário Logger fornece um registrador otimizado para Lambda que inclui informações adicionais sobre o contexto da função em todas as suas funções com saída estruturada como JSON. Use esse utilitário para fazer o seguinte:

- Capturar campos-chave do contexto do Lambda, inicialização a frio e estruturas registrando em log a saída como JSON
- Registrar em log eventos de invocação do Lambda quando instruído (desativado por padrão)
- Imprimir todos os logs somente para uma porcentagem das invocações por meio da amostragem de logs (desativado por padrão)
- Anexar chaves adicionais ao log estruturado a qualquer momento

- Use um formatador de log personalizado (Bring Your Own Formatter) para gerar logs em uma estrutura compatível com o RFC de logs da sua organização

Uso do Powertools para AWS Lambda (TypeScript) e do AWS SAM para registro em log estruturado

Siga as etapas abaixo para baixar, criar e implantar um exemplo de aplicação Hello World TypeScript com os módulos integrados do Powertools para AWS Lambda (TypeScript) usando o AWS SAM. Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET ao endpoint do API Gateway, a função do Lambda invoca, envia logs e métricas usando o formato de métricas incorporadas ao CloudWatch e envia rastreamentos ao AWS X-Ray. A função retorna uma mensagem `hello world`.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Node.js 18.x ou posterior
- AWS CLI versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS SAM

1. Inicialize a aplicação usando o modelo Hello World TypeScript.

```
sam init --app-template hello-world-powertools-typescript --name sam-app --package-type Zip --runtime nodejs18.x
```

2. Crie a aplicação.

```
cd sam-app && sam build
```

3. Implante o aplicativo.

```
sam deploy --guided
```

4. Siga as instruções na tela. Para aceitar as opções padrão fornecidas na experiência interativa, pressione `Enter`.

   > **Note**
   >
   > Em HelloWorldFunction pode não ter autorização definida, tudo bem?, certifique-se de inserir `y`.

5. Obtenha o URL da aplicação implantada:

   ```
   aws cloudformation describe-stacks --stack-name sam-app --query
    'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
   ```

6. Invoque o endpoint da API:

   ```
   curl <URL_FROM_PREVIOUS_STEP>
   ```

   Se tiver êxito, você verá esta resposta:

   ```
   {"message":"hello world"}
   ```

7. Para obter os logs da função, execute sam logs. Para obter mais informações, consulte Trabalhar com logs no Guia do desenvolvedor do AWS Serverless Application Model.

   ```
   sam logs --stack-name sam-app
   ```

   O resultado de saída do log se parece com:

   ```
   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. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
sam delete
```

Gerenciar a retenção de logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs por tempo indeterminado, exclua o grupo de logs ou configure um período de retenção após o qual o CloudWatch excluirá os logs automaticamente. Para configurar a retenção de logs, adicione o seguinte ao seu modelo do 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 do Powertools para AWS Lambda (TypeScript) e do AWS CDK para registro em log estruturado

Siga as etapas abaixo para baixar, criar e implantar um exemplo de aplicação Hello World TypeScript com os módulos integrados do Powertools para AWS Lambda (TypeScript) usando o AWS CDK. Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET ao endpoint do API Gateway, a função do Lambda invoca, envia logs e métricas usando o formato de métricas incorporadas ao CloudWatch e envia rastreamentos ao AWS X-Ray. A função retorna uma mensagem `hello world`.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Node.js 18.x ou posterior
- AWS CLI versão 2
- AWS CDK versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS CDK

1. Crie um diretório de projeto para sua aplicação.

```
mkdir hello-world
cd hello-world
```

2. Inicialize a aplicação.

```
cdk init app --language typescript
```

3. Adicione o pacote @types/aws-lambda como uma dependência de desenvolvimento.

```
npm install -D @types/aws-lambda
```

4. Instale o Utilitário logger do Powertools.

```
npm install @aws-lambda-powertools/logger
```

5. Abra o diretório lib. Você deverá ver um arquivo chamado hello-world-stack.ts. Crie novos dois novos arquivos neste diretório: hello-world.function.ts e hello-world.ts.
6. Abra o hello-world.function.ts e adicione o código a seguir ao arquivo. Este é o código da função 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. Abra o hello-world.ts e adicione o código a seguir ao arquivo. Ele contém a estrutura NodeJSFunction, que cria a função do Lambda, configura variáveis de ambiente para o Powertools e define a retenção de logs para uma semana. Também inclui a estrutura LambdaRestAPI, que cria a 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. Abra o hello-world-stack.ts. Este é o código que define sua pilha do AWS CDK. Substitua o código pelo seguinte:

```
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. Volte ao diretório do projeto.

```
cd hello-world
```

10. Implante o aplicativo.

```
cdk deploy
```

11. Obtenha o URL da aplicação implantada:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?ExportName==`apiUrl`].OutputValue' --output text
```

12. Invoque o endpoint da API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Se tiver êxito, você verá esta resposta:

```
{"message":"hello world"}
```

13. Para obter os logs da função, execute sam logs. Para obter mais informações, consulte Trabalhar com logs no Guia do desenvolvedor do AWS Serverless Application Model.

```
sam logs --stack-name HelloWorldStack
```

O resultado de saída do log se parece com:

```
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. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
cdk destroy
```

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda.

Se seu código puder ser testado no editor de Código incorporado, você encontrará logs nos resultados de execução. Ao usar o recurso de teste do console para invocar uma função, você encontrará Saída de log na seção Detalhes.

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a página Log groups (Grupos de log).
2. Escolha o grupo de logs de sua função (/aws/lambda/***nome-de-sua-função***).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma instância da sua função. Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos intrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

TypeScript Código de rastreamento em AWS Lambda

O Lambda se integra ao AWS X-Ray para ajudar você a rastrear, depurar e otimizar aplicações do Lambda. É possível usar o X-Ray para rastrear uma solicitação enquanto ela atravessa recursos na aplicação, o que pode incluir funções Lambda e outros produtos da AWS.

Para enviar dados de rastreamento ao X-Ray, é possível usar uma das três bibliotecas SDK:

- AWSDistro for OpenTelemetry (ADOT) — Uma distribuição segura, pronta para produção e AWS suportada do SDK (OTel). OpenTelemetry
- SdK do AWS X-Ray para Node.js: um SDK para geração e envio de dados de rastreamento ao X-Ray.
- Powertools for AWS Lambda (TypeScript) — Um kit de ferramentas para desenvolvedores para implementar as melhores práticas sem servidor e aumentar a velocidade do desenvolvedor.

Cada um dos SDKs oferece maneiras de enviar dados de telemetria ao serviço do X-Ray. Em seguida, é possível usar o X-Ray para visualizar, filtrar e obter insights sobre as métricas de performance da aplicação para identificar problemas e oportunidades de otimização.

> **Important**
>
> Os SDKs do X-Ray e do Powertools para AWS Lambda fazem parte de uma solução de instrumentação totalmente integrada oferecida pela AWS. As camadas do Lambda para ADOT fazem parte de um padrão em todo o setor para instrumentação de rastreamento que coleta mais dados em geral, mas pode não ser adequado para todos os casos de uso. Você pode implementar o end-to-end rastreamento no X-Ray usando qualquer uma das soluções. Para saber mais sobre como escolher entre elas, consulte Como escolher entre os SDKs do AWS Distro para OpenTelemetry e do X-Ray.

Seções

Usando Powertools para AWS Lambda (TypeScript) e AWS SAM para rastreamento

Siga as etapas abaixo para baixar, criar e implantar um exemplo de TypeScript aplicativo Hello World com módulos Powertools for AWS Lambda (TypeScript) integrados usando o. AWS SAM Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET para o endpoint do API Gateway, a função Lambda invoca, envia registros e métricas usando o formato métrico incorporado e envia rastreamentos CloudWatch para. AWS X-Ray A função retorna uma mensagem `hello world`.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Node.js 18.x ou posterior
- AWS CLI versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS SAM

1. Inicialize o aplicativo usando o TypeScript modelo Hello World.

```
sam init --app-template hello-world-powertools-typescript --name sam-app --package-type Zip --runtime nodejs18.x --no-tracing
```

2. Crie a aplicação.

```
cd sam-app && sam build
```

3. Implante o aplicativo.

```
sam deploy --guided
```

4. Siga as instruções na tela. Para aceitar as opções padrão fornecidas na experiência interativa, pressione `Enter`.

> **Note**
>
> Pois HelloWorldFunction pode não ter autorização definida, está tudo bem? , certifique-se de entrary.

5. Obtenha o URL da aplicação implantada:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Invoque o endpoint da API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Se tiver êxito, você verá esta resposta:

```
{"message":"hello world"}
```

7. Para obter os rastreamentos da função, execute sam traces.

```
sam traces
```

A saída de rastreamento se parece com:

```
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. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
sam delete
```

O X-Ray não rastreia todas as solicitações para sua aplicação. O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento seja eficiente, enquanto ainda fornece uma amostra representativa das solicitações. A taxa de amostragem é uma solicitação por segundo e 5% de solicitações adicionais.

> **Note**
>
> Não é possível configurar a taxa de amostragem do X-Ray para suas funções.

Usando Powertools para AWS Lambda (TypeScript) e o AWS CDK para rastreamento

Siga as etapas abaixo para baixar, criar e implantar um exemplo de TypeScript aplicativo Hello World com módulos Powertools for AWS Lambda (TypeScript) integrados usando o. AWS CDK Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET para o endpoint do API Gateway, a função Lambda invoca, envia registros e métricas usando o formato métrico incorporado e envia rastreamentos CloudWatch para. AWS X-Ray A função retorna uma mensagem `hello world`.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Node.js 18.x ou posterior
- AWS CLI versão 2
- AWS CDK versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS Cloud Development Kit (AWS CDK)

1. Crie um diretório de projeto para sua aplicação.

```
mkdir hello-world
cd hello-world
```

2. Inicialize a aplicação.

```
cdk init app --language typescript
```

3. Adicione o pacote @types/aws-lambda como uma dependência de desenvolvimento.

```
npm install -D @types/aws-lambda
```

4. Instale o Utilitário Tracer do Powertools.

```
npm install @aws-lambda-powertools/tracer
```

5. Abra o diretório lib. Você deve ver um arquivo chamado hello-world-stack.ts. Crie novos dois novos arquivos neste diretório: hello-world.function.ts e hello-world.ts.
6. Abra o hello-world.function.ts e adicione o código a seguir ao arquivo. Este é o código da função 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. Abra o hello-world.ts e adicione o código a seguir ao arquivo. Ele contém a NodejsFunction construção, que cria a função Lambda, configura variáveis de ambiente para Powertools e define a retenção de registros em uma semana. Também inclui a LambdaRestApi construção, que cria a 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. Abra hello-world-stack.ts. Este é o código que define sua pilha do AWS CDK. Substitua o código pelo seguinte:

```
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. Implante o aplicativo.

```
cd ..
cdk deploy
```

10. Obtenha o URL da aplicação implantada:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?ExportName==`apiUrl`].OutputValue' --output text
```

11. Invoque o endpoint da API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Se tiver êxito, você verá esta resposta:

```
{"message":"hello world"}
```

12. Para obter os rastreamentos da função, execute sam traces.

```
sam traces
```

A saída de rastreamento se parece com:

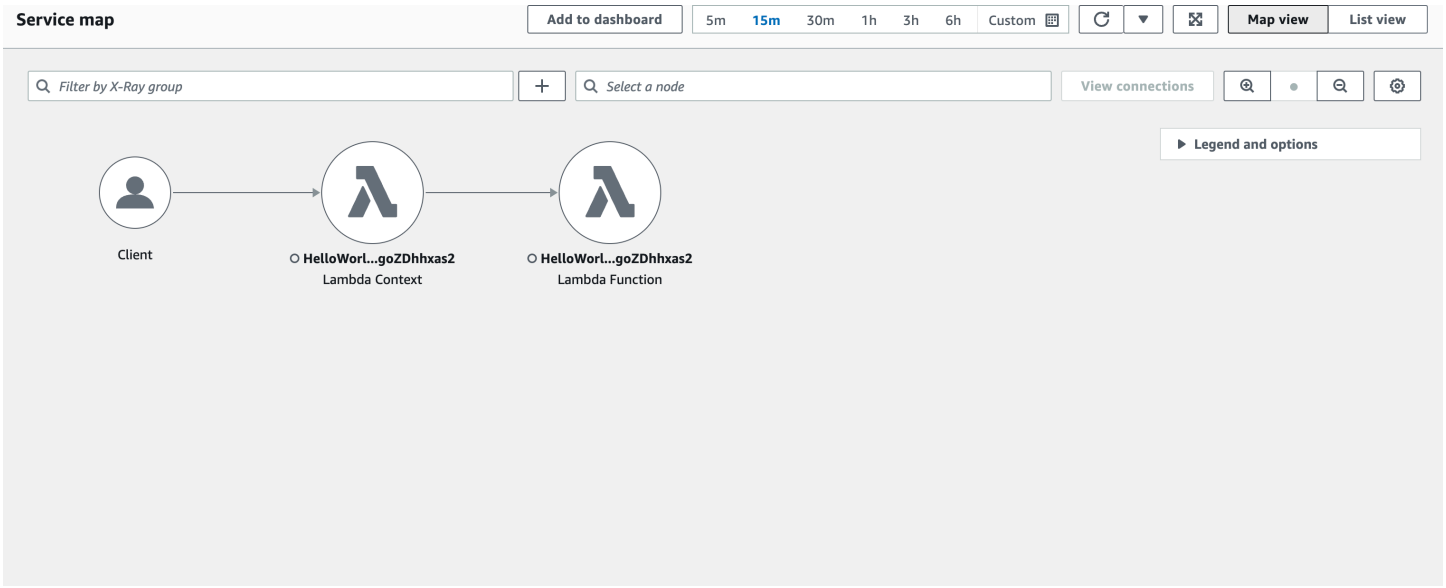
```
XRay Event [revision 1] at (2023-01-31T11:50:06.997000) with id
 (1-11a2222-111a222222cb33de3b95daf9) and duration (0.449s)
  - 0.350s - HelloWorldStack-helloworldfunction111A2BCD-Xyzv11a1bcde [HTTP: 200]
  - 0.157s - HelloWorldStack-helloworldfunction111A2BCD-Xyzv11a1bcde
    - 0.169s - Initialization
    - 0.058s - Invocation
      - 0.055s - ## index.handler
        - 0.000s - ### MySubSegment
    - 0.099s - Overhead
```

13. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
cdk destroy
```

Interpretar um rastreamento do X-Ray

Após configurar o rastreamento ativo, você pode observar solicitações específicas por meio da aplicação. O mapa de rastreamento do X-Ray fornece informações sobre a aplicação e todos os seus componentes. O seguinte exemplo mostra um rastreamento do exemplo de aplicação:



Criar funções do Lambda com Python

Você pode executar o código Python noAWS Lambda. O Lambda fornece runtimes para o Python que executa o seu código para processar eventos. Seu código é executado em um ambiente que inclui o SDK for Python (Boto 3), com as credenciais de uma função do AWS Identity and Access Management (IAM) que você gerencia. Para saber mais sobre as versões do SDK incluídas nos runtimes do Python, consulte the section called “Versões do SDK incluídas no runtime”.

O Lambda oferece suporte aos seguintes runtimes Python.

Python

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
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 de outubro de 2024	28 de fevereiro de 2025	31 de março de 2025

> **Note**
>
> As informações de runtime nessa tabela estão em constante atualização. Para obter mais informações sobre o uso dos AWS SDKs no Lambda, consulte Managing AWS SDKs in Lambda functions no Serverless Land.

Para criar uma função em Python

1. Abra o console do lambda.
2. Escolha a opção Criar função.
3. Configure as seguintes opções:
   - Nome da função: digite um nome para a função.
   - Runtime: escolha Python 3.12.
4. Escolha a opção Criar função.
5. Para configurar um evento de teste, escolha Test (Testar).
6. Em Nome do evento, insira **`test`**.
7. Escolha Salvar alterações.
8. Escolha Test (Testar) para invocar a função.

O console cria uma função do Lambda com um único arquivo de origem chamado `lambda_function`. Você pode editar esse arquivo e adicionar mais arquivos no editor de códigos integrado. Para salvar suas alterações, selecione Salvar. Em seguida, para executar seu código, escolhaTeste.

> **Note**
>
> O console do Lambda usa o AWS Cloud9 para fornecer um ambiente de desenvolvimento integrado no navegador. Você também pode usar o AWS Cloud9 para desenvolver funções do Lambda em seu próprio ambiente. Para obter mais informações, consulte Working with AWS Lambda functions using the AWS Toolkit no Guia do usuário do AWS Cloud9.

> **Note**
>
> Para começar com o desenvolvimento de aplicativos no ambiente local, implante um dos aplicativos de exemplo disponíveis no repositório do GitHub deste guia.
>
> Aplicativos Lambda de exemplo do em Python
>
> - blank-python: uma função Python que mostra o uso de registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e o AWS SDK.

Sua função do Lambda é fornecida com um grupo de logs do CloudWatch Logs. O runtime envia detalhes sobre cada invocação para o CloudWatch Logs. Ele retransmite quaisquer logs que sua função produz durante a invocação. Se a função retornar um erro, o Lambda formatará o erro e o retornará para o invocador.

Tópicos

Versões do SDK incluídas no runtime

A versão do AWS SDK incluída no runtime do Python depende da versão do runtime e da Região da AWS. Para encontrar a versão do SDK incluída no runtime que você está usando, crie uma função do Lambda com o código a seguir.

```
import boto3
import botocore

def lambda_handler(event, context):
   print(f'boto3 version: {boto3.__version__}')
   print(f'botocore version: {botocore.__version__}')
```

Formato de resposta

Nos runtimes do Python 3.12 e posteriores, as funções retornam caracteres Unicode como parte da resposta JSON. Os runtimes anteriores do Python retornam sequências de escape para caracteres

Unicode nas respostas. Por exemplo, no Python 3.11, se você retornar uma string Unicode, como “こんにちは”, ela escapará dos caracteres Unicode e retornará “\u3053\u3093\u306b\u3061\u306f”. O runtime do Python 3.12 retorna o “こんにちは” original.

O uso de respostas Unicode reduz o tamanho das respostas do Lambda, facilitando o ajuste de respostas maiores para o tamanho máximo de carga útil de 6 MB para funções síncronas. No exemplo anterior, a versão com escape é de 32 bytes, em comparação com 17 bytes com a string Unicode.

Ao atualizar para o Python 3.12, talvez seja necessário ajustar o código para levar em conta o novo formato de resposta. Se o chamador espera Unicode de escape, você deve adicionar código à função de retorno para escapar do Unicode manualmente ou ajustar o chamador para lidar com o retorno do Unicode.

Desligamento normal para extensões

Os runtimes do Python 3.12 e versões posteriores oferecem recursos aprimorados de desligamento normal para funções com extensões externas. Quando o Lambda desliga um ambiente de execução, ele envia um sinal `SIGTERM` para o runtime e, depois, um evento `SHUTDOWN` para cada extensão externa registrada. Você pode capturar o sinal `SIGTERM` na função do Lambda e apagar recursos, como conexões de banco de dados que foram criadas pela função.

Para saber mais sobre o ciclo de vida do ambiente de execução, consulte Ambiente de execução do Lambda. Para ver exemplos de como usar o desligamento normal com extensões, consulte o AWS Samples GitHub repository.

Definir o manipulador de função do Lambda em Python

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. A função é executada até que o manipulador retorne uma resposta, seja encerrado ou atinja o tempo limite.

Você pode usar a seguinte sintaxe geral ao criar um manipulador de funções no Python:

```
def handler_name(event, context):
    ...
    return some_value
```

Nomenclatura

O nome do manipulador da função do Lambda especificado no momento em que você cria uma função do Lambda é derivado:

- do nome do arquivo no qual a função de manipulador do Lambda está localizada.
- do nome da função do manipulador do Python.

Um manipulador de funções pode ter qualquer nome; no entanto, o padrão no console do Lambda é `lambda_function.lambda_handler`. Esse nome de manipulador da função reflete o nome da função (`lambda_handler`) e o arquivo em que o código do manipulador está armazenado (`lambda_function.py`).

Se você criar uma função no console usando um nome de arquivo ou nome de manipulador de funções diferente, deverá editar o nome do manipulador padrão.

Para alterar o nome do manipulador de funções (console)

1. Abra a página Funções do console do Lambda e escolha sua função.
2. Escolha a guia Código.
3. Role para baixo até o painel Configurações de runtime e escolha Editar.
4. Em Manipulador, insira o novo nome para seu manipulador de funções.
5. Escolha Salvar.

Como funciona

Quando o Lambda invoca seu handler de função, o runtime do Lambda transmite dois argumentos ao handler da função:

- O primeiro argumento é o objeto do evento. Um evento é um documento no formato JSON que contém dados para uma função do Lambda processar. O runtime do Lambda converte o evento em um objeto e o transmite para o código da função. Ele geralmente é do tipo Python `dict`. Ele também pode ser do tipo `list`, `str`, `int`, `float`, ou `NoneType`.

  O objeto do evento contém informações do serviço de chamada. Ao invocar uma função, você determina a estrutura e o conteúdo do evento. Quando um serviço da AWS invoca a função, ele define a estrutura do evento. Para obter mais informações sobre eventos de serviços da AWS, consulte Invocando o Lambda com eventos de outros serviços da AWS.
- O segundo argumento é o objeto de contexto. Um objeto de contexto é passado para sua função pelo Lambda no runtime. Esse objeto oferece métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de runtime.

Retornar um valor

Opcionalmente, um manipulador poderá retornar um valor. O que acontece com o valor retornado depende do tipo de invocação e do serviço que invocou a função. Por exemplo:

- Se você usar o tipo de invocação `RequestResponse`, como o Invocação síncrona, o AWS Lambda retornará o resultado da chamada de função Python para o cliente que invoca a função do Lambda (na resposta HTTP à solicitação de invocação, serializada em JSON). Por exemplo, o console do AWS Lambda usa o tipo de invocação `RequestResponse`, assim, quando você invocar a função no console, ele exibirá o valor retornado.
- Se o handler retornar objetos que não podem ser serializados por `json.dumps`, o runtime retornará um erro.
- Se o handler retornar `None`, assim como funções Python sem uma instrução `return` fazem implicitamente, o runtime retornará `null`.
- Se você usar o tipo de invocação `Event` (uma invocação assíncrona), o valor será descartado.

> **Note**
>
> No Python 3.9 e em versões posteriores, o Lambda inclui o requestId da invocação na resposta de erro.

Exemplos

A seção a seguir mostra exemplos de funções Python com as quais você pode usar o Lambda. Se você usar o console do Lambda para criar sua função, não será necessário associar um arquivo .zip para executar as funções nesta seção. Essas funções usam bibliotecas Python padrão que estão incluídas no runtime do Lambda selecionado. Para ter mais informações, consulte Pacotes de implantação do Lambda.

Retornar uma mensagem

O exemplo a seguir mostra uma função chamada `lambda_handler`. A função aceita a entrada do usuário de um nome e sobrenome, e retorna uma mensagem que contém dados do evento recebido como entrada.

```
def lambda_handler(event, context):
    message = 'Hello {} {}!'.format(event['first_name'], event['last_name'])
    return {
        'message' : message
    }
```

Você pode usar os seguintes dados de evento para chamar a função:

```
{
    "first_name": "John",
    "last_name": "Smith"
}
```

A resposta mostra os dados do evento passados como entrada:

```
{
    "message": "Hello John Smith!"
}
```

Analisar uma resposta

O exemplo a seguir mostra uma função chamada `lambda_handler`. A função usa dados de evento passados pelo Lambda no runtime. Ele analisa a variável de ambiente na `AWS_REGION` retornada na resposta 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
        })
    }
```

Você pode usar quaisquer dados de eventos para chamar a função:

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

Os runtimes do Lambda definem diversas variáveis de ambiente durante a inicialização. Para obter mais informações sobre as variáveis de ambiente retornadas na resposta no runtime, consulte Usar variáveis de ambiente no Lambda para configurar valores no código.

A função neste exemplo depende de uma resposta bem-sucedida (em `200`) da API Invoke. Para obter mais informações sobre o status da API de invocação, consulte a Sintaxe de resposta Invoke.

Retornar um cálculo

O exemplo a seguir mostra uma função chamada `lambda_handler`. A função aceita a entrada do usuário e retorna um cálculo para ele. Para obter mais informações sobre este exemplo, consulte o repositório do GitHub de aws-doc-sdk-examples.

```
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
```

Você pode usar os seguintes dados de evento para chamar a função:

```
{
    "action": "increment",
    "number": 3
}
```

Trabalhar com arquivos .zip para funções do Lambda em Python

O código da sua função do AWS Lambda compreende um arquivo .py contendo o código do manipulador da função e todos os pacotes e módulos adicionais dos quais o código depende. Para implantar o código dessa função no Lambda, você usa um pacote de implantação. Esse pacote pode ser um arquivo .zip ou uma imagem de contêiner. Para obter mais informações sobre o uso de imagens de contêiner com Python, consulte Implantar funções do Lambda em Python com imagens de contêiner.

Para criar um pacote de implantação como arquivo .zip, você pode usar um utilitário de arquivo .zip integrado da ferramenta da linha de comando ou qualquer outro utilitário de arquivo .zip, como o 7zip. Os exemplos mostrados nas seções a seguir pressupõem que você esteja usando uma ferramenta `zip` da linha de comando em um ambiente Linux ou MacOS. Para usar os mesmos comandos no Windows, você pode instalar o Subsistema Windows para Linux para obter uma versão do Ubuntu e do Bash integrada ao Windows.

Observe que o Lambda usa permissões de arquivo POSIX, então pode ser necessário definir permissões para a pasta do pacote de implantação antes da criação do arquivo .zip.

Tópicos

Dependências de runtime em Python

Para funções do Lambda que usam o runtime do Python, uma dependência pode ser qualquer pacote ou módulo do Python. Ao implantar a função usando um arquivo .zip, você pode adicionar essas dependências ao arquivo .zip com o código da função ou usar uma camada do Lambda. Uma camada é um arquivo .zip que pode conter código adicional e outro conteúdo. Para saber mais sobre como usar camadas do Lambda no Python, consulte the section called “Camadas”.

Os runtimes do Lambda em Python incluem o AWS SDK for Python (Boto3) e suas dependências. O Lambda fornece o SDK no runtime para cenários de implantação em que você não consegue adicionar suas próprias dependências. Esses cenários incluem a criação de funções no console usando o editor de código integrado ou funções embutidas em modelos do AWS Serverless Application Model (AWS SAM) ou do AWS CloudFormation.

O Lambda atualiza periodicamente as bibliotecas no runtime do Python para incluir as atualizações e os patches de segurança mais recentes. Se sua função usa a versão do SDK Boto3 incluída no runtime, mas o pacote de implantação inclui dependências do SDK, isso pode causar problemas de desalinhamento de versão. Por exemplo, o pacote de implantação pode incluir a dependência urllib3 do SDK. Quando o Lambda atualiza o SDK no runtime, problemas de compatibilidade entre a nova versão do runtime e a versão do urllib3 no pacote de implantação podem fazer com que a função apresente falha.

> **Important**
>
> Para manter o controle total sobre suas dependências e evitar possíveis problemas de desalinhamento de versão, recomendamos que você adicione todas as dependências da função ao pacote de implantação, mesmo que as versões delas estejam incluídas no runtime do Lambda. Isso inclui o SDK do Boto3.

Para descobrir qual versão do SDK para Python (Boto3) está incluída no runtime que você está usando, consulte the section called “Versões do SDK incluídas no runtime”.

No modelo de responsabilidade compartilhada da AWS, você é responsável pelo gerenciamento de todas as dependências dos pacotes de implantação das suas funções. Isso inclui a aplicação de atualizações e patches de segurança. Para atualizar as dependências no pacote de implantação da função, primeiro crie um novo arquivo .zip e depois carregue esse arquivo no Lambda. Consulte Criar um pacote de implantação .zip com dependências e Criar e atualizar funções do Lambda em Python usando arquivos .zip para obter mais informações.

Criar um pacote de implantação .zip sem dependências

Se o código da função não tiver dependências, o arquivo .zip conterá somente o arquivo .py com o código do manipulador da função. Use seu utilitário zip preferencial para criar um arquivo .zip com o arquivo .py na raiz. Se o arquivo .py não estiver na raiz do arquivo .zip, o Lambda não poderá executar o código.

Para saber como implantar o arquivo .zip para criar uma função do Lambda ou atualizar uma já existente, consulte Criar e atualizar funções do Lambda em Python usando arquivos .zip.

Criar um pacote de implantação .zip com dependências

Se o código da função depender de pacotes ou módulos adicionais, você poderá acrescentar essas dependências ao arquivo .zip com o código da função ou usar uma camada do Lambda. As instruções desta seção mostram como incluir as dependências no pacote de implantação .zip. Para que o Lambda execute seu código, o arquivo .py contendo o código do manipulador e todas as dependências da sua função devem ser instalados na raiz do arquivo .zip.

Suponha que o código da função esteja salvo em um arquivo denominado `lambda_function.py`. Os exemplos de comandos da CLI a seguir criam um arquivo .zip denominado `my_deployment_package.zip` que contém o código da função e suas dependências. Você pode instalar suas dependências diretamente em uma pasta do diretório do projeto ou usar um ambiente virtual em Python.

Para criar o pacote de implantação (diretório do projeto)

1. Navegue até o diretório do projeto que contém o arquivo `lambda_function.py` de código-fonte. Neste exemplo, o diretório se chama `my_function`.

   ```
   cd my_function
   ```

2. Crie um diretório com o nome de package, no qual você instalará as dependências.

   ```
   mkdir package
   ```

   Observe que, para um pacote de implantação .zip, o Lambda espera que o código-fonte e suas dependências estejam todos na raiz do arquivo .zip. No entanto, instalar dependências diretamente no diretório do projeto pode introduzir um grande número de novos arquivos e pastas e dificultar a navegação pelo IDE. Crie um diretório `package` separado aqui para manter suas dependências separadas do código-fonte.

3. Instale as dependências no diretório `package`. O exemplo abaixo instala o SDK do Boto3 do Python Package Index usando pip. Se o código da função usar pacotes Python que você mesmo criou, salve-os no diretório `package`.

   ```
   pip install --target ./package boto3
   ```

4. Crie um arquivo .zip com as bibliotecas instaladas na raiz.

```
cd package
zip -r ../my_deployment_package.zip .
```

   Isso gerará um arquivo `my_deployment_package.zip` no diretório do projeto.

5. Adicione o arquivo lambda_function.py à raiz do arquivo .zip

```
cd ..
zip my_deployment_package.zip lambda_function.py
```

   O arquivo .zip deve ter uma estrutura de diretórios simples, com o código do manipulador da função e todas as pastas de dependência instaladas na raiz, como a seguir.

```
my_deployment_package.zip
|- bin
|  |-jp.py
|- boto3
|  |-compat.py
|  |-data
|  |-docs
...
|- lambda_function.py
```

   Se o arquivo .py que contém o código do manipulador da função não estiver na raiz do arquivo .zip, o Lambda não poderá executar o código.

Para criar o pacote de implantação (ambiente virtual)

1. Crie e ative um ambiente virtual no diretório do projeto. Neste exemplo, o diretório do projeto é denominado `my_function`.

```
~$ cd my_function
~/my_function$ python3.12 -m venv my_virtual_env
~/my_function$ source ./my_virtual_env/bin/activate
```

2. Instale as bibliotecas necessárias usando pip. O exemplo a seguir instala o SDK do Boto3

```
(my_virtual_env) ~/my_function$ pip install boto3
```

3. Use `pip show` para encontrar o local no ambiente virtual em que o pip instalou as dependências.

```
(my_virtual_env) ~/my_function$ pip show <package_name>
```

   A pasta na qual o pip instala as bibliotecas pode ser denominada `site-packages` ou `dist-packages`. Essa pasta pode estar localizada no diretório `lib/python3.x` ou `lib64/python3.x` (em que python3.x representa a versão do Python que você está usando).

4. Desativar o ambiente virtual

```
(my_virtual_env) ~/my_function$ deactivate
```

5. Navegue até o diretório que contém as dependências que você instalou com o pip e crie um arquivo zip no diretório do projeto com as dependências instaladas na raiz. Neste exemplo, o pip instalou as dependências no diretório `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. Navegue até a raiz do diretório do projeto em que o arquivo .py que contém o código do manipulador está localizado e adicione esse arquivo à raiz do pacote .zip. Neste exemplo, o arquivo do código da função é denominado `lambda_function.py`.

```
~/my_function/my_virtual_env/lib/python3.12/site-packages$ cd ../../../../
~/my_function$ zip my_deployment_package.zip lambda_function.py
```

Caminho de pesquisa de dependências e bibliotecas incluídas no runtime

Quando você usa uma instrução `import` no código, o runtime do Python pesquisa os diretórios no caminho de pesquisa até encontrar o módulo ou pacote. Por padrão, o primeiro local que o runtime pesquisa é o diretório no qual o pacote de implantação .zip é descompactado e montado (`/var/task`). Se você incluir uma versão de uma biblioteca incluída no runtime do pacote de implantação, a sua versão terá precedência sobre a versão incluída no runtime. As dependências do pacote de implantação também têm precedência sobre as dependências das camadas.

Quando você adiciona uma dependência a uma camada, o Lambda a extrai para `/opt/python/lib/python3.x/site-packages`, em que `python3.x` representa a versão do runtime que você

está usando, ou para `/opt/python`. No caminho de pesquisa, esses diretórios têm precedência sobre os diretórios que contêm as bibliotecas incluídas no runtime e as bibliotecas instaladas no pip (`/var/runtime` e `/var/lang/lib/python3.x/site-packages`). As bibliotecas em camadas da função, portanto, têm precedência sobre as versões incluídas no runtime.

> **Note**
>
> No runtime gerenciado e na imagem base do Python 3.11, o AWS SDK e suas dependências estão instalados no diretório `/var/lang/lib/python3.11/site-packages`.

Você pode ver o caminho de pesquisa completo para a função do Lambda adicionando o trecho de código a seguir.

```
import sys

search_path = sys.path
print(search_path)
```

> **Note**
>
> Como as dependências do pacote de implantação ou das camadas têm precedência sobre as bibliotecas incluídas no runtime, isso pode causar problemas de desalinhamento de versão se você incluir uma dependência do SDK, como urllib3, no pacote sem também incluir o SDK. Se você implantar sua própria versão de uma dependência do Boto3, também deverá implantar o Boto3 como uma dependência no pacote de implantação. Recomendamos que você empacote todas as dependências da função, mesmo que as versões delas estejam incluídas no runtime.

Você também pode adicionar dependências em uma pasta separada dentro do pacote .zip. Por exemplo, você pode adicionar uma versão do SDK do Boto3 a uma pasta do pacote .zip denominada `common`. Quando o pacote .zip é descompactado e montado, essa pasta é colocada dentro do diretório `/var/task`. Para usar uma dependência proveniente de uma pasta do pacote de implantação .zip no código, use uma instrução `import from`. Por exemplo, para usar uma versão do Boto3 proveniente de uma pasta denominada `common` no pacote.zip, use a instrução a seguir.

```
from common import boto3
```

Usar pastas __pycache__

É recomendável não incluir pastas `__pycache__` no pacote de implantação da função. O bytecode do Python compilado em uma máquina de compilação com arquitetura ou sistema operacional diferente pode não ser compatível com o ambiente de execução do Lambda.

Criar pacotes de implantação .zip com bibliotecas nativas

Se a função usar somente pacotes e módulos Python puros, você poderá usar o comando `pip install` para instalar as dependências em qualquer máquina de compilação local e criar o arquivo .zip. Muitas bibliotecas populares de Python, incluindo NumPy e Pandas, não são Python puro e contêm código escrito em C ou C++. Ao adicionar bibliotecas que contêm código C/C++ ao pacote de implantação, você deve criar seu pacote corretamente para garantir que ele seja compatível com o ambiente de execução do Lambda.

A maioria dos pacotes disponíveis no Python Package Index (PyPI) está disponível como "wheels" (arquivos .whl). Um arquivo .whl é um tipo de arquivo ZIP que contém uma distribuição criada com binários pré-compilados para um sistema operacional específico e uma arquitetura de conjunto de instruções. Para tornar seu pacote de implantação compatível com o Lambda, instale o wheel para sistemas operacionais Linux e a arquitetura do conjunto de instruções da sua função.

Alguns pacotes podem estar disponíveis apenas como distribuições de origem. Para esses pacotes, você mesmo precisa compilar e construir os componentes C/C++.

Para ver quais distribuições estão disponíveis para o pacote necessário, faça o seguinte:

1. Pesquise o nome do pacote na página principal do Python Package Index.
2. Escolha a versão do pacote que você deseja usar.
3. Escolha Baixar arquivos.

Trabalhar com distribuições construídas (wheels)

Para baixar um wheels compatível com o Lambda, use a opção `--platform` do pip.

Se a função do Lambda usar a arquitetura do conjunto de instruções x86_64, execute o comando `pip install` a seguir para instalar um wheel compatível no diretório `package`. Substitua `--python 3.x` pela versão do runtime do Python que você está usando.

```
pip install \
```

```
--platform manylinux2014_x86_64 \
--target=package \
--implementation cp \
--python-version 3.x \
--only-binary=:all: --upgrade \
<package_name>
```

Se sua função usar a arquitetura do conjunto de instruções arm64, execute o comando a seguir. Substitua `--python 3.x` pela versão do runtime do Python que você está usando.

```
pip install \
--platform manylinux2014_aarch64 \
--target=package \
--implementation cp \
--python-version 3.x \
--only-binary=:all: --upgrade \
<package_name>
```

Trabalhar com distribuições de origem

Se seu pacote só estiver disponível como distribuição de origem, você mesmo precisará criar as bibliotecas C/C++. Para tornar seu pacote compatível com o ambiente de execução do Lambda, é necessário criá-lo em um ambiente que use o mesmo sistema operacional Amazon Linux 2. Você pode fazer isso criando o pacote em uma instância Linux do Amazon EC2.

Para saber como iniciar e se conectar a uma instância do Linux do Amazon EC2, consulte Tutorial: comece a usar instâncias do Linux do Amazon EC2 no Guia do usuário do Amazon EC2 para instâncias do Linux.

Criar e atualizar funções do Lambda em Python usando arquivos .zip

Depois de criar seu pacote de implantação .zip, você poderá usá-lo para criar uma nova função do Lambda ou atualizar uma existente. É possível implantar o pacote .zip usando o console do Lambda, a AWS Command Line Interface e a API do Lambda. Você também pode criar e atualizar funções do Lambda usando o AWS Serverless Application Model (AWS SAM) e o AWS CloudFormation.

O tamanho máximo de um pacote de implantação .zip para o Lambda é 250 MB (descompactado). Esse limite se aplica ao tamanho combinado de todos os arquivos que você carrega, inclusive qualquer camada do Lambda.

O runtime do Lambda precisa de permissão para ler os arquivos no pacote de implantação. Na notação octal de permissões do Linux, o Lambda precisa de 644 permissões para arquivos não executáveis (rw-r--r--) e 755 permissões (rwxr-xr-x) para diretórios e arquivos executáveis.

No Linux e no MacOS, use o comando `chmod` para alterar as permissões de arquivo em arquivos e diretórios do seu pacote de implantação. Por exemplo, para dar a um arquivo executável as permissões corretas, execute o comando a seguir.

```
chmod 755 <filepath>
```

Para alterar as permissões de arquivo no Windows, consulte Set, View, Change, or Remove Permissions on an Object na documentação do Microsoft Windows.

Criar e atualizar funções com arquivos .zip usando o console

Para criar uma nova função, você deve primeiro criar a função no console e depois carregar o arquivo .zip. Para atualizar uma função existente, abra a página da função e siga o mesmo procedimento para adicionar o arquivo .zip atualizado.

Se o arquivo .zip for menor que 50 MB, você poderá criar ou atualizar uma função carregando o arquivo diretamente da máquina local. Para arquivos .zip maiores que 50 MB, você deve primeiro carregar o pacote para um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando o AWS Management Console, consulte Conceitos básicos do Amazon S3. Para carregar arquivos usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

> **Note**
>
> Você não pode alterar o tipo de pacote de implantação (.zip ou imagem de contêiner) de uma função existente. Por exemplo, você não pode converter uma função de imagem de contêiner para usar um arquivo compactado .zip. É necessário criar uma nova função.

Para criar uma função (console)

1. Abra a página Funções do console do Lambda e escolha Criar função.
2. Escolha Author from scratch (Criar do zero).
3. Em Basic information (Informações básicas), faça o seguinte:

a. Em Nome da função, insira o nome da função.

b. Em Runtime, selecione o runtime que você deseja usar.

c. (Opcional) Em Arquitetura, escolha a arquitetura do conjunto de instruções para a função. O valor da arquitetura padrão é X86_64. Certifique-se de que o pacote de implantação .zip da função seja compatível com a arquitetura do conjunto de instruções que você escolheu.

4. (Opcional) Em Permissões, expanda Alterar função de execução padrão. Crie uma função de execução ou use uma existente.
5. Escolha a opção Criar função. O Lambda cria uma função básica “Hello world” usando o runtime escolhido.

Você pode carregar o arquivo .zip da máquina local (console)

1. Na página Funções do console Lambda, escolha a função para a qual você deseja carregar o arquivo .zip.
2. Selecione a guia Código.
3. No painel do Código-fonte, escolha Carregar de.
4. Escolha o arquivo .zip.
5. Para carregar o arquivo .zip, faça o seguinte:

   a. Selecione Carregar e, em seguida, selecione o arquivo .zip no seletor de arquivos.

   b. Escolha Open (Abrir).

   c. Escolha Salvar.

Para carregar um arquivo .zip de um bucket do Amazon S3 (console)

1. Na página Funções do console do Lambda, escolha a função para a qual você deseja carregar um novo arquivo .zip.
2. Selecione a guia Código.
3. No painel do Código-fonte, escolha Carregar de.
4. Escolha Local do Amazon S3.
5. Cole o URL do link do Amazon S3 do arquivo .zip e escolha Salvar.

Atualizar funções do arquivo .zip usando o editor de código do console

Para algumas funções com pacotes de implantação .zip, você pode usar o editor de código integrado do console do Lambda para atualizar o código da função diretamente. Para usar esse recurso, a função deve atender aos seguintes critérios:

- A função deve usar um dos runtimes da linguagem interpretada (Python, Node.js ou Ruby)
- O pacote de implantação da função deve ser menor que 3 MB.

O código das funções com pacotes de implantação de imagens de contêiner não pode ser editado diretamente no console.

Para atualizar o código da função usando o editor de código do console

1. Abra a página Funções do console do Lambda e selecione a função.
2. Selecione a guia Código.
3. No painel Código-fonte, selecione o arquivo de código-fonte e edite-o no editor de código integrado.
4. Quando terminar de editar o código, escolha Implantar para salvar as alterações e atualizar a função.

Criar e atualizar funções com arquivos .zip usando a AWS CLI

Você pode usar a AWS CLI para criar uma função ou atualizar uma existente usando um arquivo .zip. Use os comandos create-function e update-function-code para implantar o pacote .zip. Se o arquivo .zip for menor que 50 MB, você poderá carregar o pacote .zip de um local do arquivo na máquina de compilação local. Para arquivos .zip maiores, você deve carregar o pacote .zip de um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

> **Note**
>
> Se você carregar o arquivo .zip de um bucket do Amazon S3 usando a AWS CLI, o bucket deverá estar na mesma Região da AWS que sua função.

Para criar uma função usando um arquivo .zip com a AWS CLI, você deve especificar o seguinte:

- O nome da função (`--function-name`)
- O runtime da função (`--runtime`)
- O nome do recurso da Amazon (ARN) da função de execução da função (`--role`)
- O nome do método do manipulador no código da função (`--handler`)

Você também deve especificar a local do arquivo .zip. Se o arquivo .zip estiver localizado em uma pasta da máquina de compilação local, use a opção `--zip-file` para especificar o caminho do arquivo, conforme mostrado no comando do exemplo a seguir.

```
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
```

Para especificar o local do arquivo .zip em um bucket do Amazon S3, use a opção `--code` conforme mostrado no comando do exemplo a seguir. Você só precisa usar o parâmetro `S3ObjectVersion` para objetos com versionamento.

```
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
```

Para atualizar uma função existente usando a CLI, especifique o nome da função usando o parâmetro `--function-name`. Você também deve especificar o local do arquivo .zip que deseja usar para atualizar o código da função. Se o arquivo .zip estiver localizado em uma pasta da máquina de compilação local, use a opção `--zip-file` para especificar o caminho do arquivo, conforme mostrado no comando do exemplo a seguir.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Para especificar o local do arquivo .zip em um bucket do Amazon S3, use as opções `--s3-bucket` e `--s3-key` conforme mostrado no comando do exemplo a seguir. Você só precisa usar o parâmetro `--s3-object-version` para objetos com versionamento.

```
aws lambda update-function-code --function-name myFunction \
```

```
--s3-bucket DOC-EXAMPLE-BUCKET --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Criar e atualizar funções com arquivos .zip usando a API do Lamba

Para criar e atualizar funções usando um arquivo .zip, use as seguintes operações de API:

- CreateFunction
- UpdateFunctionCode

Criar e atualizar funções com arquivos .zip usando o AWS SAM

O AWS Serverless Application Model (AWS SAM) é um kit de ferramentas que ajuda a simplificar o processo de criação e execução de aplicações com tecnologia sem servidor na AWS. Você define os recursos para a aplicação em um modelo YAML ou JSON e usa a interface da linha de comando do AWS SAM (CLI do AWS SAM) para criar, empacotar e implantar aplicações. Quando você cria uma função do Lambda com base em um modelo do AWS SAM, o AWS SAM cria automaticamente um pacote de implantação .zip ou uma imagem de contêiner com o código da função e quaisquer dependências que você especificar. Para saber mais sobre como usar o AWS SAM para criar e implantar funções do Lambda, consulte Conceitos básicos do AWS SAM no Guia do desenvolvedor do AWS Serverless Application Model.

Você também pode usar o AWS SAM para criar uma função do Lambda usando um arquivo .zip existente. Para criar uma função do Lambda usando o AWS SAM, salve o arquivo .zip em um bucket do Amazon S3 ou em uma pasta local na máquina de compilação. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

No modelo do AWS SAM, o recurso `AWS::Serverless::Function` especifica a função do Lambda. Nesse recurso, defina as seguintes propriedades para criar uma função usando um arquivo .zip:

- `PackageType`: definir como `Zip`
- `CodeUri`: definir como o URI do Amazon S3 do código da função, o caminho para a pasta local ou o objeto FunctionCode
- `Runtime`: definir como o runtime escolhido

Com o AWS SAM, se o arquivo .zip for maior que 50 MB, você não precisará carregá-lo primeiro em um bucket do Amazon S3. O AWS SAM poderá carregar pacotes .zip com o tamanho máximo permitido de 250 MB (descompactados) de um local da máquina de compilação local.

Para saber mais sobre a implantação de funções usando o arquivo .zip no AWS SAM, consulte AWS::Serverless::Function no Guia do desenvolvedor do AWS SAM.

Criar e atualizar funções com arquivos .zip usando o AWS CloudFormation

Você pode usar o AWS CloudFormation para criar uma função do Lambda usando um arquivo .zip. Para criar uma função do Lambda de um arquivo.zip, primeiro carregue o arquivo em um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

Para os runtimes do Node.js e do Python, você também pode fornecer código-fonte embutido no modelo AWS CloudFormation. Em seguida, o AWS CloudFormation cria um arquivo .zip que contém o código quando você cria a função.

Usar um arquivo .zip existente

No modelo do AWS CloudFormation, o recurso `AWS::Lambda::Function` especifica a função do Lambda. Nesse recurso, defina as seguintes propriedades para criar uma função usando um arquivo .zip:

- `PackageType`: definir como `Zip`
- `Code`: inserir o nome do bucket do Amazon S3 e o nome do arquivo .zip nos campos `S3Bucket` e `S3Key`
- `Runtime`: definir como o runtime escolhido

Criar um arquivo .zip com base no código embutido

Você pode declarar funções simples escritas em Python ou Node.js embutidas em um modelo do AWS CloudFormation. Como o código está incorporado em YAML ou JSON, você não pode adicionar nenhuma dependência externa ao pacote de implantação. Isso significa que a função precisa usar a versão do AWS SDK que está incluída no runtime. Os requisitos do modelo, como a necessidade de evitar certos caracteres, também dificultam o uso dos recursos de verificação de sintaxe e preenchimento de código do IDE. Isso significa que seu modelo pode exigir testes adicionais. Em função dessas limitações, declarar funções embutidas é mais adequado para códigos muito simples que não mudam com frequência.

Para criar um arquivo .zip com base no código embutido para runtimes do Node.js e do Python, defina as seguintes propriedades no recurso `AWS::Lambda::Function` do modelo:

- `PackageType`: definir como `Zip`
- `Code`: digitar o código da função no campo `ZipFile`
- `Runtime`: definir como o runtime escolhido

O arquivo .zip que o AWS CloudFormation gera não pode exceder 4 MB. Para saber mais sobre a implantação de funções usando o arquivo .zip no AWS CloudFormation, consulte AWS::Lambda::Function no Guia do desenvolvedor do AWS CloudFormation.

Implante funções do Lambda em Python com imagens de contêiner

Existem três maneiras de criar uma imagem de contêiner para uma função do Lambda em Python:

- Usar uma imagem base da AWS para Python

  As imagens base da AWS são pré-carregadas com um runtime de linguagem, um cliente de interface de runtime para gerenciar a interação entre o Lambda e o código da sua função e um emulador de interface de runtime para testes locais.

- Usar uma imagem base somente para sistema operacional da AWS

  As imagens base somente para sistema operacional da AWS contêm uma distribuição do Amazon Linux e o emulador de interface de runtime. Essas imagens são comumente usadas para criar imagens de contêiner para linguagens compiladas, como Go e Rust e para uma linguagem ou versão de linguagem para a qual o Lambda não fornece uma imagem base, como Node.js 19. Você também pode usar imagens base somente para sistema operacional para implementar um runtime personalizado. Para tornar a imagem compatível com o Lambda, você deve incluir o cliente de interface de runtime do Python na imagem.

- Usar uma imagem base que não é da AWS

  Você também pode usar uma imagem base alternativa de outro registro de contêiner, como Alpine Linux ou Debian. Você também pode usar uma imagem personalizada criada por sua organização. Para tornar a imagem compatível com o Lambda, você deve incluir o cliente de interface de runtime do Python na imagem.

> **Tip**
>
> Para reduzir o tempo necessário para que as funções do contêiner do Lambda se tornem ativas, consulte Use multi-stage builds na documentação do Docker. Para criar imagens de contêiner eficientes, siga as Melhores práticas para gravar Dockerfiles.

Esta página explica como criar, testar e implantar imagens de contêiner para o Lambda.

Tópicos

- Usar uma imagem base alternativa com o cliente da interface de runtime

Imagens base da AWS para Python

A AWS fornece as seguintes imagens base para Python:

Tags	Runtime	Sistema operacional	Dockerfile	Desaprovação
3.12	Python 3.12	Amazon Linux 2023	Dockerfile para Python 3.12 no GitHub	
3.11	Python 3.11	Amazon Linux 2	Dockerfile para Python 3.11 no GitHub	
3.10	Python 3.10	Amazon Linux 2	Dockerfile para Python 3.10 no GitHub	
3.9	Python 3.9	Amazon Linux 2	Dockerfile para Python 3.9 no GitHub	
3.8	Python 3.8	Amazon Linux 2	Dockerfile para Python 3.8 no GitHub	14 de outubro de 2024

Repositório do Amazon ECR: gallery.ecr.aws/lambda/python

As imagens base do Python 3.12 e posteriores são baseadas na imagem de contêiner mínimo do Amazon Linux 2023. As imagens base do Python 3.8-3.11 são baseadas na imagem do Amazon Linux 2. As imagens baseadas no AL2023 oferecem várias vantagens em relação ao Amazon Linux 2, incluindo uma área de implantação menor e versões atualizadas de bibliotecas, como `glibc`.

As imagens baseadas no AL2023 usam o `microdnf` (com link simbólico `dnf`) como o gerenciador de pacotes, em vez do `yum`, que é o gerenciador de pacotes padrão no Amazon Linux 2. O `microdnf` é uma implementação autônoma do `dnf`. Para obter uma lista dos pacotes incluídos nas imagens baseadas no AL2023, consulte as colunas Contêiner mínimo em Comparar pacotes instalados em imagens de contêiner do Amazon Linux 2023. Para obter mais informações sobre as diferenças entre o AL2023 e o Amazon Linux 2, consulte Introdução ao runtime do Amazon Linux 2023 para AWS Lambda no blog AWS Compute.

> **Note**
>
> Para executar imagens baseadas no AL2023 localmente, inclusive com o AWS Serverless Application Model (AWS SAM), você deve usar o Docker versão 20.10.10 ou posterior.

Caminho de pesquisa de dependência nas imagens base

Quando você usa uma instrução `import` no código, o runtime do Python pesquisa os diretórios no caminho de pesquisa até encontrar o módulo ou pacote. Por padrão, o runtime pesquisa primeiro o diretório `{LAMBDA_TASK_ROOT}`. Se você incluir uma versão de uma biblioteca incluída no runtime em sua imagem, sua versão terá precedência sobre a versão incluída no runtime.

Outras etapas no caminho de pesquisa dependem da versão da imagem base do Lambda para Python que você está usando:

- Python 3.11 e posteriores: as bibliotecas incluídas no runtime e as bibliotecas instaladas no pip são instaladas no diretório `/var/lang/lib/python3.11/site-packages`. Esse diretório tem precedência sobre `/var/runtime` no caminho de pesquisa. Você pode substituir o SDK usando o pip para instalar uma versão mais recente. Você pode usar o pip para verificar se o SDK incluído no runtime e suas dependências são compatíveis com qualquer pacote que você instalar.
- Python 3.8-3.10: as bibliotecas incluídas no runtime são instaladas no diretório `/var/runtime`. As bibliotecas instaladas pelo pip são instaladas no diretório `/var/lang/lib/python3.x/site-packages`. O diretório `/var/runtime` tem precedência sobre `/var/lang/lib/python3.x/site-packages` no caminho de pesquisa.

Você pode ver o caminho de pesquisa completo para a função do Lambda adicionando o trecho de código a seguir.

```
import sys

search_path = sys.path
print(search_path)
```

Usar uma imagem base da AWS para Python

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- Docker (versão mínima 20.10.10 para Python 3.12 e imagens base posteriores)
- Python

Criação de uma imagem a partir de uma imagem base

Para criar uma imagem de contêiner com base em uma imagem base da AWS para Python

1. Crie um diretório para o projeto e depois mude para esse diretório.

   ```
   mkdir example
   cd example
   ```

2. Crie um novo arquivo chamado `lambda_function.py`. É possível adicionar o exemplo de código de função a seguir ao arquivo para testes ou usar o seu próprio código.

   Example Função do Python

   ```
   import sys
   def handler(event, context):
       return 'Hello from AWS Lambda using Python' + sys.version + '!'
   ```

3. Crie um novo arquivo chamado `requirements.txt`. Caso esteja usando o exemplo de código de função da etapa anterior, você poderá deixar o arquivo em branco porque não há dependências. Caso contrário, liste cada biblioteca necessária. Por exemplo, veja como `requirements.txt` deverá ficar se sua função usar oAWS SDK for Python (Boto3):

   Example requirements.txt

   ```
   boto3
   ```

4. Crie um novo Dockerfile com a seguinte configuração:

   - Defina a propriedade `FROM` como o URI da imagem base.

- Use o comando COPY para copiar o código da função e as dependências do runtime para `{LAMBDA_TASK_ROOT}`, uma variável de ambiente definida pelo Lambda.
- Defina o argumento `CMD` como o manipulador de funções do 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. Crie a imagem do Docker com o comando docker build. O exemplo a seguir nomeia a imagem como `docker-image` e atribui a ela a tag `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> O comando especifica a opção `--platform linux/amd64` para garantir que seu contêiner seja compatível com o ambiente de execução do Lambda, independentemente da arquitetura da sua máquina de compilação. Se você pretende criar uma função do Lambda usando a arquitetura do conjunto de instruções ARM64, certifique-se de alterar o comando para usar a opção `--platform linux/arm64` em vez disso.

(Opcional) Teste a imagem localmente

1. Inicie a imagem do Docker com o comando docker run. Neste exemplo, `docker-image` é o nome da imagem e `test` é a tag.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Esse comando executa a imagem como um contêiner e cria um endpoint local em `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se você criou a imagem do Docker para a arquitetura do conjunto de instruções ARM64, certifique-se de usar a opção `--platform linux/arm64`, em vez de `--platform linux/amd64`.

2. Em uma nova janela de terminal, publique um evento no endpoint local.

   Linux/macOS

   No Linux e no MacOS, execute o seguinte comando `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   No PowerShell, execute o seguinte comando `Invoke-WebRequest`:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Obtenha o ID do contêiner.

```
docker ps
```

4. Use o comando docker kill para parar o contêiner. Nesse comando, substitua `3766c4ab331c` pelo ID do contêiner da etapa anterior.

```
docker kill 3766c4ab331c
```

Implantação da imagem

Para enviar a imagem ao Amazon ECR e criar a função do Lambda

1. Execute o comando get-login-password para autenticar a CLI do Docker no seu registro do Amazon ECR.
   - Defina o valor `--region` para a Região da AWS onde você deseja criar o repositório do Amazon ECR.
   - Substituir `111122223333` por seu ID da Conta da 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. Crie um repositório no Amazon ECR usando o 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**
>
> O repositório do Amazon ECR deve estar na mesma Região da AWS que a função do Lambda.

Se tiver êxito, você verá uma resposta como esta:

```
{
    "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. Copie o `repositoryUri` da saída na etapa anterior.
4. Execute o comando docker tag para aplicar uma tag na sua imagem local em seu repositório do Amazon ECR como a versão mais recente. Neste comando:

   - Substitua `docker-image:test` pelo nome e tag da sua imagem do Docker.
   - Substitua `<ECRrepositoryUri>` pelo `repositoryUri` que você copiou. Certifique-se de incluir `:latest` no final do URI.

   ```
   docker tag docker-image:test <ECRrepositoryUri>:latest
   ```

   Exemplo:

   ```
   docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
   world:latest
   ```

5. Execute o comando docker push para implantar a imagem local no repositório do Amazon ECR. Certifique-se de incluir `:latest` no final do URI do repositório.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crie um perfil de execução para a função, caso ainda não tenha um. Você precisará do nome do recurso da Amazon (ARN) do perfil na próxima etapa.
7. Criar a função do Lambda. Em `ImageUri`, especifique o URI do repositório anterior. Certifique-se de incluir `:latest` no final do 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**
>
> É possível criar uma função usando uma imagem em uma conta da AWS diferente desde que a imagem esteja na mesma região da função do Lambda. Para ter mais informações, consulte Permissões entre contas do Amazon ECR.

8. Invoque a função.

```
aws lambda invoke --function-name hello-world response.json
```

Você obterá uma resposta parecida com esta:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Para ver a saída da função, verifique o arquivo `response.json`.

Para atualizar o código da função, você deve criar a imagem novamente, fazer upload da nova imagem no repositório do Amazon ECR e, em seguida, usar o comando update-function-code para implantar a imagem na função do Lambda.

O Lambda resolve a tag de imagem em um resumo de imagem específico. Isso significa que, se você apontar a tag de imagem que foi usada para implantar a função em uma nova imagem no

Amazon ECR, o Lambda não atualizará automaticamente a função para usar a nova imagem. Para implantar a nova imagem na mesma função do Lambda, você deverá usar o comando `update-function-code`, mesmo que a tag da imagem no Amazon ECR permaneça a mesma.

Usar uma imagem base alternativa com o cliente da interface de runtime

Se você usar uma imagem base somente para sistema operacional ou uma imagem base alternativa, deverá incluir o cliente de interface de runtime na imagem. O cliente de interface de runtime estende API de tempo de execução do Lambda, que gerencia a interação entre o Lambda e o código da sua função.

Instale o cliente de interface de runtime para Python usando o gerenciador de pacotes pip:

```
pip install awslambdaric
```

Você também pode baixar o cliente de interface de runtime Python no GitHub.

O exemplo a seguir demonstra como criar uma imagem de contêiner para Python usando uma imagem base que não é da AWS. O exemplo de Dockerfile usa uma imagem base oficial do Python. O Dockerfile inclui o cliente de interface de runtime para Python.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- Docker
- Python

Criar uma imagem de uma imagem base alternativa

Para criar uma imagem de contêiner de uma imagem base que não é da AWS

1. Crie um diretório para o projeto e depois mude para esse diretório.

   ```
   mkdir example
   cd example
   ```

2. Crie um novo arquivo chamado `lambda_function.py`. É possível adicionar o exemplo de código de função a seguir ao arquivo para testes ou usar o seu próprio código.

Example Função do Python

```
import sys
def handler(event, context):
    return 'Hello from AWS Lambda using Python' + sys.version + '!'
```

3. Crie um novo arquivo chamado `requirements.txt`. Caso esteja usando o exemplo de código de função da etapa anterior, você poderá deixar o arquivo em branco porque não há dependências. Caso contrário, liste cada biblioteca necessária. Por exemplo, veja como `requirements.txt` deverá ficar se sua função usar oAWS SDK for Python (Boto3):

   Example requirements.txt

   ```
   boto3
   ```

4. Crie um novo Dockerfile. O Dockerfile a seguir usa uma imagem base oficial do Python em vez de uma imagem base da AWS. O Dockerfile inclui o cliente de interface de runtime, o que torna a imagem compatível com o Lambda. O exemplo de Dockerfile a seguir usa uma compilação em várias etapas.

   - Defina a propriedade `FROM` como a imagem básica.
   - Defina o `ENTRYPOINT` como o módulo em que você deseja que o contêiner do Docker seja executado quando for iniciado. Nesse caso, o módulo é o cliente de interface de runtime.
   - Defina o `CMD` como o manipulador de funções do 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. Crie a imagem do Docker com o comando docker build. O exemplo a seguir nomeia a imagem como `docker-image` e atribui a ela a tag `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> O comando especifica a opção `--platform linux/amd64` para garantir que seu contêiner seja compatível com o ambiente de execução do Lambda, independentemente da arquitetura da sua máquina de compilação. Se você pretende criar uma função do Lambda usando a arquitetura do conjunto de instruções ARM64, certifique-se de alterar o comando para usar a opção `--platform linux/arm64` em vez disso.

(Opcional) Teste a imagem localmente

Use o emulador de interface de runtime para testar a imagem localmente. Você pode compilar o emulador em sua imagem ou usar o procedimento a seguir instalá-lo na sua máquina local.

Para instalar o emulador de interface de runtime na sua máquina local

1. No diretório do projeto, execute o comando a seguir para baixar o emulador de interface de runtime (arquitetura x86-64) do GitHub e instalá-lo na sua máquina local.

   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
   ```

   Para instalar o emulador arm64, substitua o URL do repositório do GitHub no comando anterior pelo seguinte:

   ```
   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
   ```

   Para instalar o emulador de arm64, substitua `$downloadLink` pelo seguinte:

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
   ```

2. Inicie a imagem do Docker com o comando docker run. Observe o seguinte:

   - `docker-image` é o nome da imagem e `test` é a tag.
   - `/usr/local/bin/python -m awslambdaric lambda_function.handler` é o `ENTRYPOINT` seguido pelo `CMD` do 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
```

Esse comando executa a imagem como um contêiner e cria um endpoint local em `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se você criou a imagem do Docker para a arquitetura do conjunto de instruções ARM64, certifique-se de usar a opção `--platform linux/arm64`, em vez de `--platform linux/amd64`.

3. Publique um evento no endpoint local.

Linux/macOS

No Linux e no MacOS, execute o seguinte comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

No PowerShell, execute o seguinte comando `Invoke-WebRequest`:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Obtenha o ID do contêiner.

```
docker ps
```

5. Use o comando docker kill para parar o contêiner. Nesse comando, substitua `3766c4ab331c` pelo ID do contêiner da etapa anterior.

```
docker kill 3766c4ab331c
```

Implantação da imagem

Para enviar a imagem ao Amazon ECR e criar a função do Lambda

1. Execute o comando get-login-password para autenticar a CLI do Docker no seu registro do Amazon ECR.
   - Defina o valor `--region` para a Região da AWS onde você deseja criar o repositório do Amazon ECR.
   - Substituir `111122223333` por seu ID da Conta da 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. Crie um repositório no Amazon ECR usando o 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**
>
> O repositório do Amazon ECR deve estar na mesma Região da AWS que a função do Lambda.

Se tiver êxito, você verá uma resposta como esta:

```
{
    "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. Copie o `repositoryUri` da saída na etapa anterior.
4. Execute o comando docker tag para aplicar uma tag na sua imagem local em seu repositório do Amazon ECR como a versão mais recente. Neste comando:

- Substitua `docker-image:test` pelo nome e tag da sua imagem do Docker.
- Substitua `<ECRrepositoryUri>` pelo `repositoryUri` que você copiou. Certifique-se de incluir `:latest` no final do URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Exemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Execute o comando docker push para implantar a imagem local no repositório do Amazon ECR. Certifique-se de incluir `:latest` no final do URI do repositório.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crie um perfil de execução para a função, caso ainda não tenha um. Você precisará do nome do recurso da Amazon (ARN) do perfil na próxima etapa.
7. Criar a função do Lambda. Em `ImageUri`, especifique o URI do repositório anterior. Certifique-se de incluir `:latest` no final do 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**
>
> É possível criar uma função usando uma imagem em uma conta da AWS diferente desde que a imagem esteja na mesma região da função do Lambda. Para ter mais informações, consulte Permissões entre contas do Amazon ECR.

8. Invoque a função.

```
aws lambda invoke --function-name hello-world response.json
```

Você obterá uma resposta parecida com esta:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Para ver a saída da função, verifique o arquivo `response.json`.

Para atualizar o código da função, você deve criar a imagem novamente, fazer upload da nova imagem no repositório do Amazon ECR e, em seguida, usar o comando update-function-code para implantar a imagem na função do Lambda.

O Lambda resolve a tag de imagem em um resumo de imagem específico. Isso significa que, se você apontar a tag de imagem que foi usada para implantar a função em uma nova imagem no Amazon ECR, o Lambda não atualizará automaticamente a função para usar a nova imagem. Para implantar a nova imagem na mesma função do Lambda, você deverá usar o comando `update-function-code`, mesmo que a tag da imagem no Amazon ECR permaneça a mesma.

Para obter um exemplo de como criar uma imagem Python a partir de uma imagem base Alpine, consulte Container image support for Lambda no AWS Blog.

Como trabalhar com camadas para funções do Lambda em Python

Uma camada do Lambda é um arquivo .zip que contém código ou dados complementares. As camadas geralmente contêm dependências de biblioteca, um runtime personalizado ou arquivos de configuração. A criação de uma camada envolve três etapas gerais:

1. Empacotar o conteúdo da sua camada. Isso significa criar um arquivo .zip contendo as dependências que você deseja usar em suas funções.
2. Criar a camada no Lambda.
3. Adicionar a camada às suas funções.

Este tópico contém etapas e orientações sobre como empacotar e criar adequadamente uma camada do Lambda em Python com dependências externas de bibliotecas.

Tópicos

Pré-requisitos

Para seguir as etapas desta seção, você deve ter o seguinte:

- Python 3.11 e o instalador de pacotes pip
- AWS Command Line Interface (AWS CLI) versão 2

Neste tópico, mencionamos a aplicação de amostra `layer-python` no repositório awsdocs no GitHub. Essa aplicação contém scripts que baixam as dependências e geram as camadas. A aplicação também contém funções correspondentes que usam dependências das camadas. Após criar uma camada, você pode implantar e invocar a função correspondente para verificar se tudo

funciona corretamente. Como você usa o runtime do Python 3.11 para as funções, as camadas também devem ser compatíveis com o Python 3.11.

Na aplicação de amostra `layer-python`, há dois exemplos:

- O primeiro exemplo envolve empacotar a biblioteca `requests` em uma camada do Lambda. O diretório `layer/` contém os scripts para gerar a camada. O diretório `function/` contém uma amostra de função para ajudar a testar se a camada funciona. A maior parte deste tutorial explica como criar e empacotar essa camada.
- O segundo exemplo envolve empacotar a biblioteca `numpy` em uma camada do Lambda. O diretório `layer-numpy/` contém os scripts para gerar a camada. O diretório `function-numpy/` contém uma amostra de função para ajudar a testar se a camada funciona. Para obter um exemplo de como criar e empacotar essa camada, consulte the section called “Como trabalhar com distribuições wheel `manylinux`”.

Compatibilidade da camada em Python com o Amazon Linux

A primeira etapa para criar uma camada é agrupar todo o conteúdo da camada em um arquivo .zip. Devido às funções do Lambda serem executadas no Amazon Linux, seu conteúdo da camada deve ser capaz de compilar e criar em um ambiente Linux.

Em Python, a maioria dos pacotes está disponível como wheels (arquivos `.whl`), além da distribuição de origem. Cada wheel é um tipo de distribuição compilada compatível com uma combinação específica de versões do Python, sistemas operacionais e conjuntos de instruções de máquina.

As wheels são úteis para garantir que sua camada seja compatível com o Amazon Linux. Ao baixar suas dependências, baixe a wheel universal, se possível. (Por padrão, o `pip` instala a wheel universal, se houver uma disponível.) A wheel universal contém `any` como uma etiqueta de plataforma, indicando que é compatível com todas as plataformas, inclusive Amazon Linux.

No exemplo a seguir, você empacota a biblioteca `requests` em uma camada do Lambda. A biblioteca `requests` é um exemplo de pacote que está disponível como uma wheel universal.

Nem todos os pacotes do Python são distribuídos como wheels universais. Por exemplo, `numpy` tem várias distribuições de wheels, cada uma compatível com um conjunto diferente de plataformas. Para esses pacotes, baixe a distribuição `manylinux` a fim de garantir a compatibilidade com o Amazon Linux. Para obter instruções detalhadas sobre como empacotar essas camadas, consulte the section called “Como trabalhar com distribuições wheel `manylinux`”.

Em casos raros, talvez um pacote do Python não esteja disponível como wheel. Se existir apenas a distribuição de origem (sdist), recomendamos instalar e empacotar suas dependências em um ambiente Docker com base na imagem de contêiner básica do Amazon Linux 2023. Também recomendamos essa abordagem se você quiser incluir suas próprias bibliotecas personalizadas escritas em outras linguagens, como C/C++. Essa abordagem imita o ambiente de execução do Lambda no Docker e garante a compatibilidade do Amazon Linux para suas dependências de pacotes que não sejam do Python.

Caminhos de camada para runtimes em Python

Quando você adiciona uma camada a uma função, o Lambda carrega o conteúdo da camada no diretório `/opt` desse ambiente de execução. Para cada runtime do Lambda, a variável `PATH` já inclui caminhos de pasta específica no diretório `/opt`. Para garantir que a variável `PATH` colete o conteúdo da camada, o arquivo .zip da camada deve ter suas dependências nos seguintes caminhos de pasta:

- `python`
- `python/lib/python3.x/site-packages`

Por exemplo, o arquivo .zip de camada resultante que você cria neste tutorial tem a seguinte estrutura de diretórios:

```
layer_content.zip
# python
    # lib
        # python3.11
            # site-packages
                # requests
                # <other_dependencies> (i.e. dependencies of the requests package)
                # ...
```

A biblioteca `requests` está localizada corretamente no diretório `python/lib/python3.11/site-packages`. Isso garante que o Lambda possa localizar a biblioteca durante as invocações da função.

Empacotar o conteúdo de camada

Neste exemplo, você empacota a biblioteca `requests` Python em um arquivo .zip de camada. Conclua as etapas a seguir para instalar e empacotar o conteúdo de camada.

Para instalar e empacotar seu conteúdo de camada

1. Clone o repositório `aws-lambda-developer-guide` do GitHub, que contém a amostra de código de que você precisa no diretório `sample-apps/layer-python`.

   ```
   git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
   ```

2. Acesse o diretório `layer` do exemplo de aplicação `layer-python`. Esse diretório contém os scripts que você usa para criar e empacotar a camada corretamente.

   ```
   cd aws-lambda-developer-guide/sample-apps/layer-python/layer
   ```

3. Examine os arquivos `requirements.txt` Esse arquivo define as dependências que você deseja incluir na camada, ou seja, a biblioteca `requests`. Você pode atualizar esse arquivo para incluir quaisquer dependências que quiser incluir em sua própria camada.

   Example requirements.txt

   ```
   requests==2.31.0
   ```

4. Verifique se você tem permissões para executar os dois scripts.

   ```
   chmod 744 1-install.sh && chmod 744 2-package.sh
   ```

5. Execute o script `1-install.sh` usando o comando a seguir:

   ```
   ./1-install.sh
   ```

   Esse script usa `venv` para criar um ambiente virtual em Python chamado `create_layer`. Em seguida, ele instala todas as dependências necessárias no diretório `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. Execute o script `2-package.sh` usando o comando a seguir:

```
./2-package.sh
```

Esse script copia o conteúdo do diretório `create_layer/lib` em um novo diretório chamado `python`. Em seguida, ele compacta o conteúdo do diretório `python` em um arquivo chamado `layer_content.zip`. Esse é o arquivo .zip da sua camada. É possível descompactar o arquivo e verificar se ele contém a estrutura de arquivo correta, conforme apresentado na seção the section called “Caminhos de camada para runtimes em Python”.

Example 2-package.sh

```
mkdir python
cp -r create_layer/lib python/
zip -r layer_content.zip python
```

Como criar a camada

Nesta seção, você pega o arquivo `layer_content.zip` gerado na seção anterior e o carrega como uma camada do Lambda. É possível fazer upload de uma camada usando o AWS Management Console ou a API do Lambda por meio da AWS Command Line Interface (AWS CLI). Ao carregar seu arquivo .zip de camada, no seguinte comando PublishLayerVersion da AWS CLI, especifique `python3.11` como o runtime compatível e `arm64` como a arquitetura compatível.

```
aws lambda publish-layer-version --layer-name python-requests-layer \
    --zip-file fileb://layer_content.zip \
    --compatible-runtimes python3.11 \
    --compatible-architectures "arm64"
```

Na resposta, anote o `LayerVersionArn`, que será algo como `arn:aws:lambda:us-east-1:`*`123456789012`*`:layer:python-requests-layer:1`. Você precisará desse nome do recurso da Amazon (ARN) na próxima etapa deste tutorial, ao adicionar a camada à função.

Como adicionar a camada à sua função

Nesta seção, você implanta uma amostra de função do Lambda que usa a biblioteca `requests` em seu código de função e, em seguida, você anexa a camada. Para implantar a função, você precisará de um the section called “Perfil de execução (permissões para funções acessarem outros recursos)”.

Se ainda não tiver um perfil de execução, siga as etapas na seção recolhível. Caso contrário, pule para a próxima seção para implantar a função.

(Opcional) Criar um perfil de execução

Para criar uma função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione Create role (Criar função).
3. Crie uma função com as propriedades a seguir.

   - Entidade confiável–Lambda.
   - Permissions (Permissões): AWSLambdaBasicExecutionRole.
   - Role name (Nome da função): **`lambda-role`**.

   A política AWSLambdaBasicExecutionRole tem as permissões necessárias para a função gravar logs no CloudWatch Logs.

Para implantar a função do Lambda

1. Navegue até o diretório `function/`. Se você estiver no diretório `layer/`, execute o seguinte comando:

   ```
   cd ../function
   ```

2. Revise o código de função. A função importa a biblioteca `requests`, faz uma solicitação HTTP GET simples e retorna o código e o corpo do status.

   ```
   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. Crie um arquivo .zip de pacote de implantação usando o seguinte comando:

```
zip my_deployment_package.zip lambda_function.py
```

4. Implantar a função. No seguinte comando da AWS CLI, substitua o parâmetro `--role` pelo ARN do seu perfil de execução:

```
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
```

(Opcional) Invocar sua função sem anexar uma camada

Nessa fase, é possível, opcionalmente, tentar invocar sua função antes de anexar a camada. Se tentar fazer isso, você deverá receber um erro de importação porque sua função não pode fazer referência ao pacote `requests`. Para invocar sua função, use o seguinte comando da AWS CLI:

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

Você deve ver uma saída semelhante a:

```
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

Para ver o erro específico, abra o arquivo `response.json` de saída. Você deverá ver um `ImportModuleError` com a seguinte mensagem de erro:

```
"errorMessage": "Unable to import module 'lambda_function': No module named 'requests'"
```

Em seguida, anexe a camada à sua função. No seguinte comando da AWS CLI, substitua o parâmetro `--layers` pelo ARN da versão de camada que você anotou anteriormente:

```
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"
```

Por fim, tente invocar sua função usando o seguinte comando da AWS CLI:

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

Você deve ver uma saída semelhante a:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

O arquivo `response.json` de saída contém detalhes sobre a resposta.

(Opcional) Limpar os recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Para excluir a camada do Lambda

1. Abra a página Camadas do console do Lambda.
2. Selecione a camada que você criou.
3. Escolha Excluir, depois escolha Excluir novamente.

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Como trabalhar com distribuições wheel **manylinux**

Às vezes, um pacote que você deseja incluir como dependência não terá uma wheel universal (mais especificamente, não terá `any` como tag de plataforma). Nesse caso, em vez disso baixe a wheel compatível com `manylinux`. Isso garantirá que suas bibliotecas de camadas sejam compatíveis com o Amazon Linux.

O `numpy` é um pacote que não tem uma wheel universal. Se quiser incluir o pacote `numpy` em sua camada, você poderá executar as etapas do exemplo a seguir para instalar e empacotar sua camada corretamente.

Para instalar e empacotar seu conteúdo de camada

1. Clone o repositório `aws-lambda-developer-guide` do GitHub, que contém a amostra de código de que você precisa no diretório `sample-apps/layer-python`.

   ```
   git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
   ```

2. Acesse o diretório `layer-numpy` do exemplo de aplicação `layer-python`. Esse diretório contém os scripts que você usa para criar e empacotar a camada corretamente.

   ```
   cd aws-lambda-developer-guide/sample-apps/layer-python/layer-numpy
   ```

3. Examine os arquivos `requirements.txt` Esse arquivo define as dependências que você deseja incluir na sua camada, ou seja, a biblioteca `numpy`. Aqui, você especifica o URL da distribuição de wheel `manylinux` compatível com Python 3.11, Amazon Linux e o conjunto de instruções `x86_64`:

   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. Verifique se você tem permissões para executar os dois scripts.

   ```
   chmod 744 1-install.sh && chmod 744 2-package.sh
   ```

5. Execute o script `1-install.sh` usando o comando a seguir:

```
./1-install.sh
```

Esse script usa `venv` para criar um ambiente virtual em Python chamado `create_layer`. Em seguida, ele instala todas as dependências necessárias no diretório `create_layer/lib/python3.11/site-packages`. Nesse caso, o comando `pip` é diferente, pois você precisa especificar a tag `--platform` como `manylinux2014_x86_64`. Isso instrui o `pip` a instalar a wheel `manylinux` correta, mesmo que sua máquina local use macOS ou 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. Execute o script `2-package.sh` usando o comando a seguir:

```
./2-package.sh
```

Esse script copia o conteúdo do diretório `create_layer/lib` em um novo diretório chamado `python`. Em seguida, ele compacta o conteúdo do diretório `python` em um arquivo chamado `layer_content.zip`. Esse é o arquivo .zip da sua camada. É possível descompactar o arquivo e verificar se ele contém a estrutura de arquivo correta, conforme apresentado na seção the section called “Caminhos de camada para runtimes em Python”.

Example 2-package.sh

```
mkdir python
cp -r create_layer/lib python/
zip -r layer_content.zip python
```

Para carregar essa camada no Lambda, use o seguinte comando PublishLayerVersion da 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"
```

Na resposta, anote o `LayerVersionArn`, que será algo como `arn:aws:lambda:us-east-1:123456789012:layer:python-numpy-layer:1`. Para verificar se sua camada funciona conforme o esperado, implante a função do Lambda no diretório `function-numpy`.

Para implantar a função do Lambda

1. Navegue até o diretório `function-numpy/`. Se você estiver no diretório `layer-numpy/`, execute o seguinte comando:

```
cd ../function-numpy
```

2. Revise o código de função. A função importa a biblioteca numpy, cria uma matriz numpy simples e retorna o código e o corpo de um status fictício.

```
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. Crie um arquivo .zip de pacote de implantação usando o seguinte comando:

```
zip my_deployment_package.zip lambda_function.py
```

4. Implantar a função. No seguinte comando da AWS CLI, substitua o parâmetro `--role` pelo ARN do seu perfil de execução:

```
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
```

(Opcional) Invocar sua função sem anexar uma camada

Como opção, é possível tentar invocar sua função antes de anexar a camada. Se tentar fazer isso, você deverá receber um erro de importação porque sua função não pode fazer referência ao pacote `numpy`. Para invocar sua função, use o seguinte comando da AWS CLI:

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

Você deve ver uma saída semelhante a:

```
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

Para ver o erro específico, abra o arquivo `response.json` de saída. Você deverá ver um `ImportModuleError` com a seguinte mensagem de erro:

```
"errorMessage": "Unable to import module 'lambda_function': No module named 'numpy'"
```

Em seguida, anexe a camada à sua função. No seguinte comando da AWS CLI, substitua o parâmetro `--layers` pelo ARN da versão da sua camada:

```
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"
```

Por fim, tente invocar sua função usando o seguinte comando da AWS CLI:

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

Você deve ver uma saída semelhante a:

```
{
```

```
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Você pode examinar os logs da função para verificar se o código imprime a matriz numpy na saída padrão.

AWS LambdaObjeto de contexto em Python

Quando o Lambda executa a função, ele transmite um objeto de contexto para o handler. Esse objeto fornece métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de execução. Para obter mais informações sobre como o objeto de contexto é passado para o manipulador de funções, consulte Definir o manipulador de função do Lambda em Python.

Métodos de contexto

- `get_remaining_time_in_millis`: retorna o número de milissegundos restantes antes do tempo limite da execução.

Propriedades de contexto

- `function_name`: o nome da função do Lambda.
- `function_version`: a versão da função.
- `invoked_function_arn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `memory_limit_in_mb`: a quantidade de memória alocada para a função.
- `aws_request_id`: o identificador da solicitação de invocação.
- `log_group_name`: o grupo de logs da função.
- `log_stream_name`: a transmissão de log para a instância da função.
- `identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
  - `cognito_identity_id`: a identidade autenticada do Amazon Cognito.
  - `cognito_identity_pool_id`: o grupo de identidades do Amazon Cognito que autorizou a invocação.
- `client_context`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`
  - `client.app_package_name`

- `custom`: um `dict` de valores personalizados definidos pela aplicação cliente para dispositivos móveis.
- `env`: um `dict` de informações do ambiente fornecidas pelo AWS SDK.

O exemplo a seguir mostra uma função do handler que registra informações de contexto.

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())
```

Além das opções listadas acima, você também pode usar o AWS X-Ray SDK para Instrumentação do código Python no AWS Lambda para identificar caminhos de código críticos, rastrear a performance e capturar os dados para análise.

Registro em log da função do AWS Lambda em Python

O AWS Lambda monitora automaticamente as funções do Lambda e envia entradas de logs para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente de runtime do Lambda envia detalhes sobre cada invocação e outras saídas do código da função para o fluxo de logs. Para obter mais informações sobre o CloudWatch Logs, consulte Usar logs do Amazon CloudWatch com o AWS Lambda.

Para gerar logs do seu código de função, use o módulo integrado `logging`. Para entradas mais detalhadas, você pode usar qualquer biblioteca de registro em log que grava em `stdout` ou `stderr`.

Imprimir para o log

Para enviar uma saída básica para os logs, use um método `print` em sua função. O exemplo a seguir registra em log os valores do fluxo e do grupo de logs do CloudWatch Logs e o objeto do evento.

Se a sua função gera logs usando instruções Python `print`, o Lambda só poderá enviar saídas de log para o CloudWatch Logs em formato de texto simples. Para capturar logs em JSON estruturado, você precisa usar uma biblioteca de logs compatível. Consulte the section called “Usar controles avançados de registro em log do Lambda com Python” Para mais informações.

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 saída do 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
```

O runtime do Python registra as linhas `START`, `END` e `REPORT` para cada invocação. A linha `REPORT` inclui os seguintes dados:

RELATAR campos de dados de linha

- RequestId: o ID de solicitação exclusivo para a invocação.
- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.
- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o runtime levou para carregar a função e executar o código fora do método do handler.
- XRAY TraceId: para solicitações rastreadas, o ID de rastreamento do AWS X-Ray.
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Usar uma biblioteca de registro em log

Para logs mais detalhados, use o módulo registro em log na biblioteca padrão ou em qualquer biblioteca de registro em log de terceiros que grava em `stdout` ou `stderr`.

Para runtimes do Python compatíveis, você pode escolher se os registros criados usando o módulo `logging` padrão são capturados em texto simples ou JSON. Para saber mais, consulte the section called “Usar controles avançados de registro em log do Lambda com Python”.

Atualmente, o formato padrão de log para todos os runtimes do Python é texto simples. O exemplo a seguir mostra como as saídas de log criadas usando o módulo `logging` padrão são capturadas em texto simples no CloudWatch Logs.

```
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)
```

A saída de `logger` inclui o nível do log, o timestamp e o ID da solicitação.

```
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 o formato de log da função é definido como texto sem formatação, a configuração padrão de nível de log para runtimes do Python é WARN. Isso significa que o Lambda só envia saídas de log de nível WARN e inferior para o CloudWatch Logs. Para alterar o nível de log padrão, use o método `logging setLevel()` do Python conforme mostrado neste código de exemplo. Se você definir o formato do log da função como JSON, recomendamos que você configure o nível de log da função usando os controles de registro em log avançados do Lambda e não definindo o nível de log no código. Para saber mais, consulte the section called “Usar a filtragem em nível de log com Python”

Usar controles avançados de registro em log do Lambda com Python

Para dar mais controle sobre como os logs das funções são capturados, processados e consumidos, você pode configurar as seguintes opções de registro em log para runtimes do Python compatíveis com o Lambda:

- Formato do log: selecione entre texto simples e formato JSON estruturado para os logs da sua função.
- Nível de log: para logs no formato JSON, escolha o nível de detalhe dos logs enviados pelo Lambda para o Amazon CloudWatch, como ERROR, DEBUG ou INFO.
- Grupo de logs: escolha o grupo de logs do CloudWatch para o qual sua função envia logs.

Para obter mais informações sobre essas opções de registro em log e instruções sobre como configurar a função para usá-las, consulte the section called “Configurar controles avançados de registro em log para a função do Lambda”.

Para saber mais sobre como usar as opções de formato de log e nível de log com as funções do Lambda para Python, consulte as orientações nas seções a seguir.

Usar logs JSON estruturados com Python

Se você selecionar JSON para o formato de log da sua função, o Lambda enviará a saída de logs da biblioteca de logs padrão do Python para o CloudWatch como JSON estruturado. Cada objeto de log JSON contém pelo menos quatro pares de valores-chave com as seguintes chaves:

- `"timestamp"`: o horário em que a mensagem de log foi gerada.
- `"level"`: o nível de log atribuído à mensagem.
- `"message"`: o conteúdo da mensagem de log.
- `"requestId"`: o ID de solicitação exclusivo para invocar a função.

A biblioteca `logging` do Python também pode adicionar pares de valores de chave, como `"logger"`, para esse objeto JSON.

Os exemplos nas seções a seguir mostram como as saídas de log geradas usando a biblioteca `logging` do Python são capturadas no CloudWatch Logs quando você configura o formato de log da função como JSON.

Se você usar o método `print` para produzir saídas básicas de log, conforme descrito em the section called “Imprimir para o log”, o Lambda capturará essas saídas como texto simples, mesmo se o formato de log da função for configurado como JSON.

Saídas de log JSON padrão usando a biblioteca de logs do Python

O exemplo de trecho de código e saída de log a seguir mostra como as saídas de log padrão geradas usando a biblioteca `logging` do Python são capturadas no CloudWatch Logs quando o formato de log da função é configurado como JSON.

Example Código de log do Python

```
import logging
logger = logging.getLogger()

def lambda_handler(event, context):
    logger.info("Inside the handler function")
```

Example Registro em log JSON

```
{
    "timestamp":"2023-10-27T19:17:45.586Z",
    "level":"INFO",
    "message":"Inside the handler function",
    "logger": "root",
    "requestId":"79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Registrar parâmetros extras em log JSON

Quando o formato de log da sua função está definido como JSON, você também pode registrar parâmetros adicionais em log com a biblioteca `logging` do Python padrão usando a palavra-chave `extra` para passar um dicionário do Python para a saída do log.

Example Código de log do Python

```
import logging

def lambda_handler(event, context):
    logging.info(
        "extra parameters example",
        extra={"a":"b", "b": [3]},
```

```
    )
```

Example Registro em 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
  ]
}
```

Exceções de log em JSON

O trecho de código a seguir mostra como as exceções do Python são capturadas na saída de log da função quando o formato de log é configurado como JSON. Observe que as saídas de log geradas usando `logging.exception` são atribuídas ao nível de log ERROR.

Example Código de log do Python

```
import logging

def lambda_handler(event, context):
    try:
        raise Exception("exception")
    except:
        logging.exception("msg")
```

Example Registro em 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"
}
```

Logs JSON estruturados com outras ferramentas de registro em log

Se o código já usa outra biblioteca de logs, como o Powertools para AWS Lambda, para produzir logs JSON estruturados, você não precisará fazer nenhuma alteração. O AWS Lambda não codifica duas vezes nenhum log que já esteja codificado em JSON. Mesmo que a função seja configurada para usar logs em formato JSON, suas saídas de log serão exibidas no CloudWatch com a estrutura JSON que você definiu.

O exemplo a seguir mostra como as saídas de log geradas usando o pacote Powertools para AWS Lambda são capturadas no CloudWatch Logs. O formato da saída de log é o mesmo, independentemente se a configuração de log da função for definida como JSON ou TEXT. Para obter mais informações sobre como usar o Powertools para AWS Lambda, consulte the section called “Uso do Powertools para AWS Lambda (Python) e do AWS SAM para registro em log estruturado” e the section called “Uso do Powertools para AWS Lambda (Python) e do AWS CDK para registro em log estruturado”.

Example Trecho de código de log em Python (usando o Powertools para AWS Lambda)

```
from aws_lambda_powertools import Logger

logger = Logger()

def lambda_handler(event, context):
    logger.info("Inside the handler function")
```

Example Registro em log JSON (usando o Powertools para 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"
}
```

Usar a filtragem em nível de log com Python

Ao configurar a filtragem em nível de log, você pode optar por enviar somente logs de um determinado nível, ou inferior, para o CloudWatch Logs. Para saber como configurar a filtragem em nível de log para a função, consulte the section called “Filtragem em nível de log”.

Para o AWS Lambda filtrar os logs de aplicação de acordo com o nível de log, a função precisa usar logs em formato JSON. Isso pode ser feito de duas maneiras:

- Crie saídas de log usando a biblioteca `logging` padrão do Python e configure a função para usar logs em formato JSON. Dessa forma, o AWS Lambda filtra as saídas de log usando o par de valores-chave “nível” no objeto JSON descrito em the section called “Usar logs JSON estruturados com Python”. Para saber como configurar o formato de log da função, consulte the section called “Configurar controles avançados de registro em log para a função do Lambda”.
- Use outro método ou biblioteca de logs para criar logs JSON estruturados no código, de modo que incluam um par de valores-chave de “nível” para definir o nível da saída de log. Por exemplo, você pode usar o Powertools para AWS Lambda para gerar saídas de log JSON estruturado do seu código.

  Também é possível usar uma instrução de impressão para gerar um objeto JSON que contenha um identificador de nível de log. A instrução de impressão a seguir produz uma saída em formato JSON em que o nível do leg é definido como INFO. O AWS Lambda enviará o objeto JSON para o CloudWatch Logs se o nível de log da função estiver definido como INFO, DEBUG ou TRACE.

  ```
  print('{"msg":"My log message", "level":"info"}')
  ```

Para que o Lambda filtre os logs da função, você também precisa incluir um par de valores-chave `"timestamp"` na saída do log JSON. A hora deve ser especificada em um formato RFC 3339 válido de carimbo de data/hora. Se você não fornecer um carimbo de data/hora válido, o Lambda atribuirá ao log o nível INFO e adicionará um carimbo de data/hora.

Visualização de logs no console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda.

Se seu código puder ser testado no editor de Código incorporado, você encontrará logs nos resultados de execução. Ao usar o recurso de teste do console para invocar uma função, você encontrará Saída de log na seção Detalhes.

Visualização de logs no console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a página Log groups (Grupos de log).
2. Escolha o grupo de logs de sua função (/aws/lambda/***nome-de-sua-função***).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma instância da sua função. Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos intrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Visualização de logs com a AWS CLI

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- AWS CLI – Configuração rápida com `aws configure`

Você pode usar a AWS CLI para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

A seguinte saída deverá ser mostrada:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `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
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

A seguinte saída deverá ser mostrada:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e Ubuntu no Windows. Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa `sed` para remover as aspas do arquivo de saída e fica inativo por 15 segundos

para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 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 macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod -R 755 get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
}
```


Excluir logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ouconfigurar um período de retençãoapós o qual os logs são excluídos automaticamente.

Ferramentas e bibliotecas

O Powertools para AWS Lambda (Python) é um kit de ferramentas para desenvolvedores para implementar práticas recomendadas da tecnologia sem servidor e aumentar a velocidade do desenvolvedor. O utilitário Logger fornece um registrador otimizado para Lambda que inclui informações adicionais sobre o contexto da função em todas as suas funções com saída estruturada como JSON. Use esse utilitário para fazer o seguinte:

- Capturar campos-chave do contexto do Lambda, inicialização a frio e estruturas registrando em log a saída como JSON
- Registrar em log eventos de invocação do Lambda quando instruído (desativado por padrão)
- Imprimir todos os logs somente para uma porcentagem das invocações por meio da amostragem de logs (desativado por padrão)
- Anexar chaves adicionais ao log estruturado a qualquer momento
- Use um formatador de log personalizado (Bring Your Own Formatter) para gerar logs em uma estrutura compatível com o RFC de logs da sua organização

Uso do Powertools para AWS Lambda (Python) e do AWS SAM para registro em log estruturado

Siga as etapas abaixo para baixar, construir e implantar um exemplo de aplicação Python do Hello World com os módulos integrados do Powertools for Python com o AWS SAM. Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET ao endpoint do API Gateway, a função do Lambda invoca, envia logs e métricas usando o formato de métricas incorporadas ao CloudWatch e envia rastreamentos ao AWS X-Ray. A função retorna uma mensagem `hello world`.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Python 3.9
- AWS CLI versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS SAM

1. Inicialize a aplicação usando o modelo Python do Hello World.

```
sam init --app-template hello-world-powertools-python --name sam-app --package-type
 Zip --runtime python3.9 --no-tracing
```

2. Crie a aplicação.

```
cd sam-app && sam build
```

3. Implante o aplicativo.

```
sam deploy --guided
```

4. Siga as instruções na tela. Para aceitar as opções padrão fornecidas na experiência interativa, pressione `Enter`.

> **Note**
>
> Em HelloWorldFunction pode não ter autorização definida, tudo bem?, certifique-se de inserir y.

5. Obtenha o URL da aplicação implantada:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Invoque o endpoint da API:

```
curl GET <URL_FROM_PREVIOUS_STEP>
```

Se tiver êxito, você verá esta resposta:

```
{"message":"hello world"}
```

7. Para obter os logs da função, execute sam logs. Para obter mais informações, consulte Trabalhar com logs no Guia do desenvolvedor do AWS Serverless Application Model.

```
sam logs --stack-name sam-app
```

O resultado de saída do log se parece com:

```
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. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
sam delete
```

Gerenciar a retenção de logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs por tempo indeterminado, exclua o grupo de logs ou configure um período de retenção após o qual o CloudWatch excluirá os logs automaticamente. Para configurar a retenção de logs, adicione o seguinte ao seu modelo do 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 do Powertools para AWS Lambda (Python) e do AWS CDK para registro em log estruturado

Siga as etapas abaixo para baixar, criar e implantar um exemplo de aplicação Python do Hello World com os módulos integrados do Powertools para AWS Lambda Python usando o AWS CDK. Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET ao endpoint do API Gateway, a função do Lambda invoca, envia logs e métricas usando o formato de métricas incorporadas ao CloudWatch e envia rastreamentos ao AWS X-Ray. A função retorna uma mensagem de hello world.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Python 3.9
- AWS CLI versão 2

- AWS CDK versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS CDK

1. Crie um diretório de projeto para sua aplicação.

```
mkdir hello-world
cd hello-world
```

2. Inicialize a aplicação.

```
cdk init app --language python
```

3. Instale as dependências do Python.

```
pip install -r requirements.txt
```

4. Crie um diretório lambda_function na pasta raiz.

```
mkdir lambda_function
cd lambda_function
```

5. Crie um arquivo app.py e adicione o código a seguir ao arquivo. Este é o código da função do 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. Abra o diretório hello_world. Você verá um arquivo chamado hello-world-stack.py.

```
cd ..
cd hello_world
```

7. Abra o hello_world_stack.py e adicione o código a seguir ao arquivo. Ele contém o construtor do Lambda, que cria a função do Lambda, configura variáveis de ambiente para o Powertools e define a retenção de logs para uma semana, e o construtor do ApiGatewayv1, que cria a 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. Implante o aplicativo.

```
cd ..
cdk deploy
```

9. Obtenha o URL da aplicação implantada:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?OutputKey==`apiUrl`].OutputValue' --output text
```

10. Invoque o endpoint da API:

```
curl GET <URL_FROM_PREVIOUS_STEP>
```

Se tiver êxito, você verá esta resposta:

```
{"message":"hello world"}
```

11. Para obter os logs da função, execute sam logs. Para obter mais informações, consulte Trabalhar com logs no Guia do desenvolvedor do AWS Serverless Application Model.

```
sam logs --stack-name HelloWorldStack
```

O resultado de saída do log se parece com:

```
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. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
cdk destroy
```

Testes da função do AWS Lambda em Python

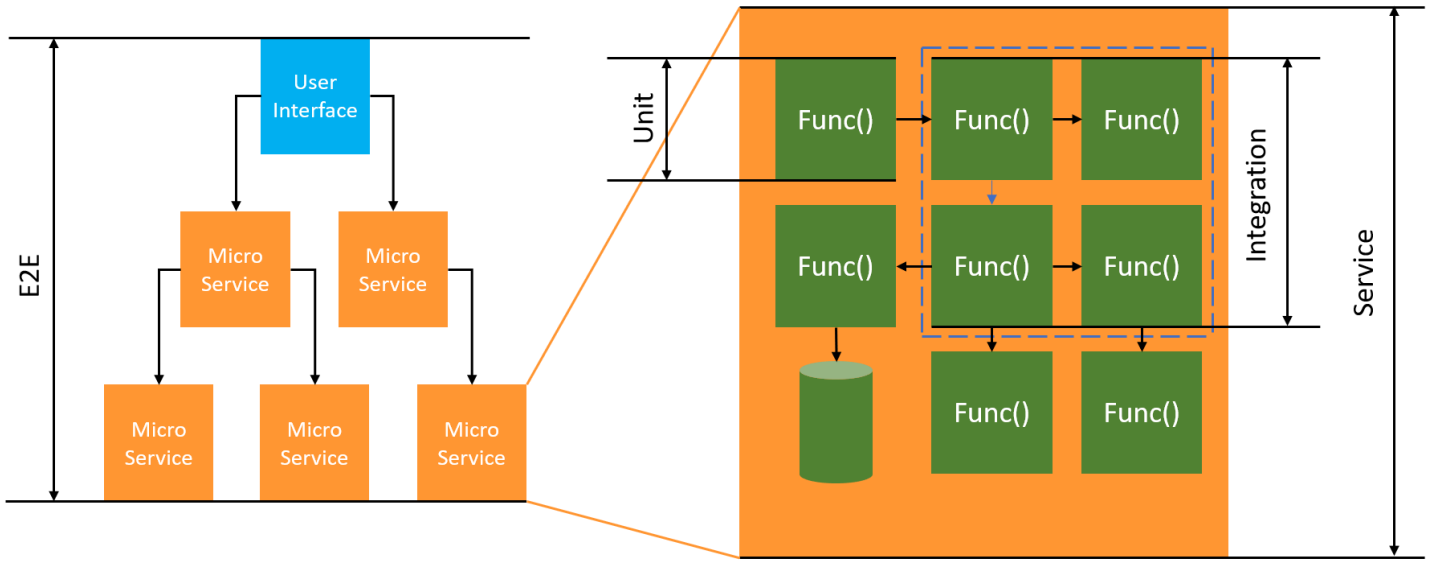
> **Note**
>
> Consulte o capítulo Testar funções para obter uma introdução completa às técnicas e melhores práticas para testar soluções com tecnologia sem servidor.

Os testes de funções com tecnologia sem servidor usam tipos e técnicas de teste tradicionais, mas você também deve considerar testar aplicações com tecnologia sem servidor como um todo. Os testes baseados em nuvem fornecerão a medida mais precisa da qualidade das suas funções e aplicações com tecnologia sem servidor.

Uma arquitetura de aplicações com tecnologia sem servidor inclui serviços gerenciados que fornecem funcionalidade de aplicações críticas por meio de chamadas de API. Por esse motivo, seu ciclo de desenvolvimento deve incluir testes automatizados que verifiquem a funcionalidade quando sua função e seus serviços interagem.

Se você não criar testes baseados em nuvem, poderá encontrar problemas devido às diferenças entre o ambiente local e o ambiente implantado. Seu processo de integração contínua deve executar testes em comparação com um conjunto de recursos provisionados na nuvem antes de promover seu código para o próximo ambiente de implantação, como GQ, Preparação ou Produção.

Continue lendo este breve guia para aprender sobre estratégias de testes para aplicações com tecnologia sem servidor ou acesse o repositório de exemplos de testes com tecnologia sem servidor para conhecer exemplos práticos, específicos da linguagem e do runtime escolhidos.



Para testes sem servidor, você ainda escreverá testes unitários, de integração e end-to-endde testes.

- Testes unitários: testes executados em comparação com um bloco de código isolado. Por exemplo, verificar a lógica de negócios para calcular a taxa de entrega de acordo com um item e um destino determinado.
- Testes de integração: testes envolvendo dois ou mais componentes ou serviços que interagem, normalmente em um ambiente de nuvem. Por exemplo, a verificação de eventos de processos de uma função em uma fila.
- nd-to-end Testes E - Testes que verificam o comportamento em todo o aplicativo. Por exemplo, garantir que a infraestrutura seja configurada corretamente e que os eventos fluam entre os serviços conforme o esperado para registrar o pedido de um cliente.

Testar suas aplicações com tecnologia sem servidor

Geralmente, você usará uma combinação de abordagens para testar o código da aplicação com tecnologia sem servidor, incluindo testes na nuvem, testes com simulações e, ocasionalmente, testes com emuladores.

Testes na nuvem

O teste na nuvem é valioso para todas as fases do teste, incluindo testes unitários, testes de integração e end-to-end testes. Você executa testes com o código implantado na nuvem e interagindo com serviços baseados na nuvem. Essa abordagem fornece a medida mais precisa da qualidade do código.

Uma maneira conveniente de depurar a função do Lambda na nuvem é com o uso de um evento de teste no console. Um evento de teste é uma entrada JSON para sua função. Se a função não necessitar de entrada, o evento poderá ser um documento JSON vazio (`{}`). O console fornece eventos de exemplo para uma variedade de integrações de serviços. Depois de criar um evento no console, você pode compartilhá-lo com sua equipe para tornar os testes mais fáceis e consistentes.

> **Note**
>
> Testar uma função no console é uma maneira rápida de começar a usar, mas automatizar os ciclos de teste garante a qualidade da aplicação e a velocidade de desenvolvimento.

Ferramentas de teste

Existem ferramentas e técnicas para acelerar os loops de feedback de desenvolvimento. Por exemplo, o AWS SAM Accelerate e o modo de observação do AWS CDK diminuem o tempo necessário para atualizar os ambientes de nuvem.

Moto é uma biblioteca Python para simular serviços e recursos da AWS para que você possa testar suas funções com pouca ou nenhuma modificação usando decoradores para interceptar e simular respostas.

O recurso de validação do Powertools para AWS Lambda (Python) fornece decoradores para que você possa validar eventos de entrada e respostas de saída das suas funções do Python.

Para obter mais informações, leia a publicação do blog Testes de unidade do Lambda com Python e simulação dos serviços da AWS.

Para reduzir a latência envolvida com as iterações de implantação na nuvem, consulte AWS Serverless Application Model (AWS SAM) Accelerate e o modo de exibição do AWS Cloud Development Kit (AWS CDK). Essas ferramentas monitoram sua infraestrutura e codificam alterações. Elas reagem a essas alterações criando e implantando atualizações incrementais automaticamente em seu ambiente de nuvem.

Exemplos que usam essas ferramentas estão disponíveis no repositório de código de exemplos de testes Python.

Instrumentação do código Python no AWS Lambda

O Lambda se integra ao AWS X-Ray para ajudar você a rastrear, depurar e otimizar aplicações do Lambda. É possível usar o X-Ray para rastrear uma solicitação enquanto ela atravessa recursos na aplicação, o que pode incluir funções Lambda e outros produtos da AWS.

Para enviar dados de rastreamento ao X-Ray, é possível usar uma das três bibliotecas SDK:

- AWS Distro for OpenTelemetry (ADOT): uma distribuição segura, pronta para produção e com suporte na AWS do SDK OpenTelemetry (OTel).
- AWS X-Ray SDK for Python: um SDK para geração e envio de dados de rastreamento ao X-Ray.
- Powertools para AWS Lambda (Python): um kit de ferramentas para desenvolvedores para implementar práticas recomendadas da tecnologia sem servidor e aumentar a velocidade do desenvolvedor.

Cada um dos SDKs oferece maneiras de enviar dados de telemetria ao serviço do X-Ray. Em seguida, é possível usar o X-Ray para visualizar, filtrar e obter insights sobre as métricas de performance da aplicação para identificar problemas e oportunidades de otimização.

> **Important**
>
> Os SDKs do X-Ray e do Powertools para AWS Lambda fazem parte de uma solução de instrumentação totalmente integrada oferecida pela AWS. As camadas do Lambda para ADOT fazem parte de um padrão em todo o setor para instrumentação de rastreamento que coleta mais dados em geral, mas pode não ser adequado para todos os casos de uso. É possível implementar o rastreamento de ponta a ponta no X-Ray usando ambas as soluções. Para saber mais sobre como escolher entre elas, consulte Como escolher entre os SDKs do AWS Distro para OpenTelemetry e do X-Ray.

Seções

- Ativar o rastreamento com a API do Lambda
- Ativar o rastreamento com o AWS CloudFormation
- Interpretar um rastreamento do X-Ray
- Armazenar dependências de runtime em uma camada (SDK do X-Ray)

Uso do Powertools para AWS Lambda (Python) e do AWS SAM para rastreamento

Siga as etapas abaixo para baixar, criar e implantar um exemplo de aplicação Python do Hello World com os módulos integrados do Powertools para AWS Lambda Python usando o AWS SAM. Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET ao endpoint do API Gateway, a função do Lambda invoca, envia logs e métricas usando o formato de métricas incorporadas ao CloudWatch e envia rastreamentos ao AWS X-Ray. A função retorna uma mensagem de hello world.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Python 3.9
- AWS CLI versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS SAM

1. Inicialize a aplicação usando o modelo Python do Hello World.

```
sam init --app-template hello-world-powertools-python --name sam-app --package-type
 Zip --runtime python3.9 --no-tracing
```

2. Crie a aplicação.

```
cd sam-app && sam build
```

3. Implante o aplicativo.

```
sam deploy --guided
```

4. Siga as instruções na tela. Para aceitar as opções padrão fornecidas na experiência interativa, pressione `Enter`.

> **Note**
>
> Em HelloWorldFunction pode não ter autorização definida, tudo bem?, certifique-se de inserir `y`.

5. Obtenha o URL da aplicação implantada:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Invoque o endpoint da API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Se tiver êxito, você verá esta resposta:

```
{"message":"hello world"}
```

7. Para obter os rastreamentos da função, execute sam traces.

```
sam traces
```

A saída de rastreamento se parece com:

```
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. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
sam delete
```

O X-Ray não rastreia todas as solicitações para sua aplicação. O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento seja eficiente, enquanto ainda fornece uma amostra representativa das solicitações. A taxa de amostragem é uma solicitação por segundo e 5% de solicitações adicionais.

> **Note**
>
> Não é possível configurar a taxa de amostragem do X-Ray para suas funções.

Uso do Powertools para AWS Lambda (Python) e do AWS CDK para rastreamento

Siga as etapas abaixo para baixar, criar e implantar um exemplo de aplicação Python do Hello World com os módulos integrados do Powertools para AWS Lambda Python usando o AWS CDK. Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET ao endpoint do API Gateway, a função do Lambda invoca, envia logs e métricas usando o formato de métricas incorporadas ao CloudWatch e envia rastreamentos ao AWS X-Ray. A função retorna uma mensagem de hello world.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Python 3.9
- AWS CLI versão 2
- AWS CDK versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS CDK

1. Crie um diretório de projeto para sua aplicação.

```
mkdir hello-world
cd hello-world
```

2. Inicialize a aplicação.

```
cdk init app --language python
```

3. Instale as dependências do Python.

```
pip install -r requirements.txt
```

4. Crie um diretório lambda_function na pasta raiz.

```
mkdir lambda_function
```

```
cd lambda_function
```

5. Crie um arquivo app.py e adicione o código a seguir ao arquivo. Este é o código da função do 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. Abra o diretório hello_world. Você verá um arquivo chamado hello-world-stack.py.

```
cd ..
cd hello_world
```

7. Abra o hello_world_stack.py e adicione o código a seguir ao arquivo. Ele contém o construtor do Lambda, que cria a função do Lambda, configura variáveis de ambiente para o Powertools e define a retenção de logs para uma semana, e o construtor do ApiGatewayv1, que cria a 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. Implante o aplicativo.

```
cd ..
cdk deploy
```

9. Obtenha o URL da aplicação implantada:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?OutputKey==`apiUrl`].OutputValue' --output text
```

10. Invoque o endpoint da API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

    Se tiver êxito, você verá esta resposta:

```
{"message":"hello world"}
```

11. Para obter os rastreamentos da função, execute sam traces.

```
sam traces
```

    A saída dos rastreamentos se parece com:

```
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. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
cdk destroy
```

Usar o ADOT para instrumentar funções Python

O ADOT fornece camadas do Lambda totalmente gerenciadas que empacotam tudo o que você precisa para coletar dados de telemetria usando o SDK do OTel. Ao consumir essa camada, é possível instrumentar suas funções Lambda sem precisar modificar nenhum código de função. Você também pode configurar sua camada para fazer a inicialização personalizada do OTel. Para obter mais informações, consulte Custom configuration for the ADOT Collector on Lambda (Configuração personalizada para o ADOT Collector no Lambda) na documentação do ADOT.

Para runtimes Python, você pode adicionar a camada do Lambda gerenciada pela AWS para ADOT Python a fim de instrumentar suas funções automaticamente. Essa camada funciona nas arquiteturas arm64 e x86_64. Para obter instruções detalhadas sobre como adicionar essa camada, consulte Suporte do AWS Distro for OpenTelemetry Lambda para Python.

Usar o SDK do X-Ray para instrumentar suas funções Python

Para registrar detalhes sobre as chamadas feitas pela sua função do Lambda para outros recursos na sua aplicação, você também pode usar o AWS X-Ray SDK for Python. Para obter o SDK, adicione o pacote `aws-xray-sdk` às dependências do aplicativo.

Example requirements.txt

```
jsonpickle==1.3
aws-xray-sdk==2.4.3
```

No seu código de função, você pode instrumentar clientes AWS SDK aplicando patch à biblioteca `boto3` com o módulo `aws_xray_sdk.core`.

Example função: rastrear um cliente AWS SDK

```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

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

client = boto3.client('lambda')
client.get_account_settings()
```

```
def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES\r' + jsonpickle.encode(dict(**os.environ)))
  ...
```

Depois de adicionar as dependências corretas e fazer as devidas mudanças de código, ative o rastreamento na configuração da sua função usando o console do Lambda ou a API.

Ativar o rastreamento com o console do Lambda

Para alternar o rastreamento ativo na sua função do Lambda usando o console, siga as etapas abaixo:

Para ativar o rastreamento ativo

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e depois Monitoring and operations tools (Ferramentas de monitoramento e operações).
4. Selecione a opção Editar.
5. Em X-Ray, ative a opção Active tracing (Rastreamento ativo).
6. Escolha Salvar.

Ativar o rastreamento com a API do Lambda

Configure o rastreamento na sua função do Lambda com aAWS CLI ou o AWS SDK, usando as seguintes operações de API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration \
--function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Ativar o rastreamento com o AWS CloudFormation

Para ativar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example function-inline.yml: configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example template.yml: configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretar um rastreamento do X-Ray

Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você ativa o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à função de execução da função. Caso contrário, adicione a política AWSXRayDaemonWriteAccess à função de execução.

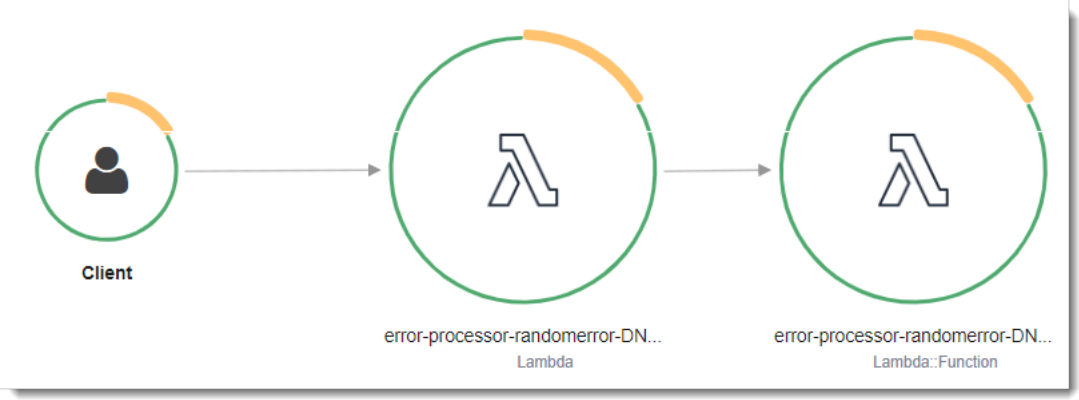
Após configurar o rastreamento ativo, você pode observar solicitações específicas por meio da aplicação. O grafo de serviço do X-Ray exibe informações sobre sua aplicação e todos os componentes. A imagem a seguir demonstra uma aplicação com duas funções. A função principal processa eventos e, às vezes, retorna erros. A segunda função de cima para baixo processa erros

que aparecem no primeiro grupo de logs e usa o AWS SDK para chamar o X-Ray, o Amazon Simple Storage Service (Amazon S3) e o Amazon CloudWatch Logs.

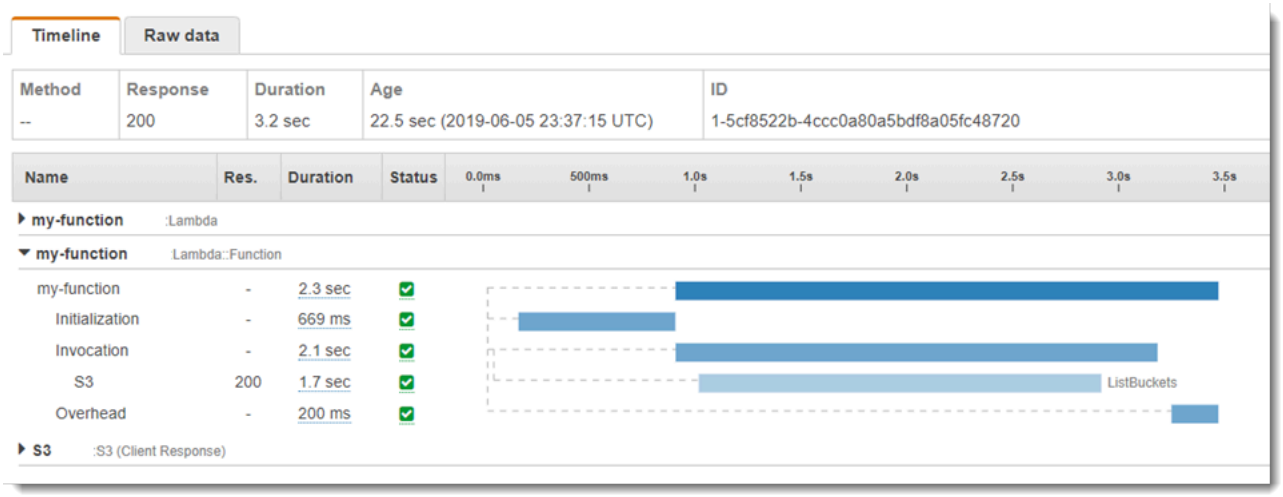
O X-Ray não rastreia todas as solicitações para sua aplicação. O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento seja eficiente, enquanto ainda fornece uma amostra representativa das solicitações. A taxa de amostragem é uma solicitação por segundo e 5% de solicitações adicionais.

> **Note**
>
> Não é possível configurar a taxa de amostragem do X-Ray para suas funções.

No X-Ray, um rastreamento registra informações sobre uma solicitação que é processada por um ou mais serviços. O Lambda registra dois segmentos por rastreamento, o que cria dois nós no gráfico de serviços. A imagem a seguir destaca esses dois nós:



O primeiro nó à esquerda representa o serviço do Lambda, que recebe a solicitação de invocação. O segundo nó representa a sua função do Lambda específica. O exemplo a seguir mostra um rastreamento com esses dois segmentos. Ambos têm o nome my-function, mas um tem a origem `AWS::Lambda` e o outro, a origem `AWS::Lambda::Function`. Se o segmento `AWS::Lambda` mostrar um erro, o serviço Lambda teve um problema. Se o segmento `AWS::Lambda::Function` mostrar um erro, sua função teve um problema.



Este exemplo expande o segmento `AWS::Lambda::Function` para mostrar seus três subsegmentos:

- Inicialização: representa o tempo gasto carregando a função e executando o código de inicialização. Esse subsegmento aparece somente para o primeiro evento que cada instância da função processa.
- Invocação: representa o tempo gasto na execução do código do manipulador.
- Sobrecarga: representa o tempo gasto pelo runtime do Lambda preparando-se para lidar com o próximo evento.

Você também pode instrumentar clientes HTTP, registrar consultas SQL e criar subsegmentos personalizados com anotações e metadados. Para obter mais informações, consulte AWS X-Ray SDK for Python no Guia do desenvolvedor do AWS X-Ray.

> **Definição de preço**
>
> Você pode usar o rastreamento do X-Ray gratuitamente todos os meses até determinado limite como parte do nível gratuito da AWS. Além do limite, o X-Ray cobra por

armazenamento e recuperação de rastreamento. Para obter mais informações, consulte Preços do AWS X-Ray.

Armazenar dependências de runtime em uma camada (SDK do X-Ray)

Se você usar o X-Ray SDK para instrumentar os clientes do AWS SDK com seu código de função, seu pacote de implantação poderá se tornar bastante grande. Para evitar o upload de dependências de runtime todas as vezes que você atualizar seu código de função, empacote o SDK do X-Ray em uma camada do Lambda.

O exemplo a seguir mostra um recurso `AWS::Serverless::LayerVersion` que armazena o AWS X-Ray SDK for Python.

Example template.yml: camada de dependências

```
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
```

Com essa configuração, você atualizará a camada de biblioteca somente se alterar as dependências de runtime. Já que o pacote de implantação de função inclui apenas o seu código, isso pode ajudar a reduzir o tempo de upload.

A criação de uma camada de dependências requer alterações de compilação para gerar o arquivo da camada antes da implantação. Para obter um exemplo funcional, consulte o aplicativo de exemplo blank-python.

Construir funções do Lambda com Ruby

Você pode executar o código Ruby noAWS Lambda. O Lambda fornece runtimes para Ruby que executam seu código para processar eventos. Seu código é executado em um ambiente que inclui o AWS SDK for Ruby, com as credenciais de uma função do AWS Identity and Access Management (IAM) que você gerencia. Para saber mais sobre as versões do SDK incluídas nos runtimes do Ruby, consulte the section called “Versões do SDK incluídas no runtime”.

O Lambda oferece suporte aos seguintes runtimes do Ruby:

Ruby

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
Ruby 3.3	`ruby3.3`	Amazon Linux 2023			
Ruby 3.2	`ruby3.2`	Amazon Linux 2			

Para criar uma função do Ruby

1. Abra o console do lambda.
2. Escolha a opção Criar função.
3. Configure as seguintes opções:
   - Nome da função: digite um nome para a função.
   - Runtime: escolha Ruby 3.2.
4. Escolha a opção Criar função.
5. Para configurar um evento de teste, escolha Test (Testar).
6. Em Nome do evento, insira **`test`**.
7. Escolha Salvar alterações.
8. Escolha Test (Testar) para invocar a função.

O console cria uma função do Lambda com um único arquivo de origem chamado `lambda_function.rb`. Você pode editar esse arquivo e adicionar mais arquivos no editor de códigos integrado. Para salvar suas alterações, selecione Salvar. Em seguida, para executar seu código, escolhaTeste.

> **Note**
>
> O console do Lambda usa o AWS Cloud9 para fornecer um ambiente de desenvolvimento integrado no navegador. Você também pode usar o AWS Cloud9 para desenvolver funções do Lambda em seu próprio ambiente. Para obter mais informações, consulte Working with AWS Lambda functions using the AWS Toolkit no Guia do usuário do AWS Cloud9.

O arquivo `lambda_function.rb` exporta uma função chamada `lambda_handler` que utiliza um objeto de evento e um objeto de contexto. Esta é a função de manipulador que o Lambda chama quando a função é invocada. O runtime da função do Ruby obtém eventos de invocação do Lambda e os transmite ao handler. Na configuração da função, o valor do manipulador é `lambda_function.lambda_handler`.

Quando você salva seu código de função, o console do Lambda cria um pacote de implantação de arquivos .zip. Quando desenvolver o código de função fora do console (usando um IDE), você precisará criar um pacote de implantação para carregar o código na função do Lambda.

> **Note**
>
> Para começar com o desenvolvimento de aplicativos no ambiente local, implante um dos aplicativos de exemplo disponíveis no repositório do GitHub deste guia.
>
> Aplicações de exemplo do Lambda em Ruby
>
> - blank-ruby: uma função do Ruby que mostra o uso de registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.
> - Ruby Code Samples for AWS Lambda: exemplos de código escritos em Ruby que demonstram como interagir com o AWS Lambda.

O runtime transmite um objeto de contexto para o manipulador, além do evento de invocação. O objeto de contexto contém informações adicionais sobre a invocação, a função e o ambiente de execução. Outras informações estão disponíveis de variáveis de ambiente.

Sua função do Lambda é fornecida com um grupo de logs do CloudWatch Logs. O runtime envia detalhes sobre cada invocação para o CloudWatch Logs. Ele retransmite quaisquer logs que sua função produz durante a invocação. Se a função retornar um erro, o Lambda formatará o erro e o retornará para o invocador.

Tópicos

Versões do SDK incluídas no runtime

A versão do AWS SDK incluída no runtime do Ruby depende da versão do runtime e da Região da AWS. O AWS SDK para Ruby foi projetado para ser modular e é separado pelo AWS service (Serviço da AWS). Para encontrar o número da versão de um determinado gem de serviço incluído no runtime que você está usando, crie uma função do Lambda com o código no formato a seguir. Substitua `aws-sdk-s3` e `Aws::S3` pelo nome dos gems de serviço que o código usa.

```
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
```

Como habilitar Yet Another Ruby JIT (YJIT)

O runtime Ruby 3.2 é compatível com YJIT, um compilador Ruby JIT leve e minimalista. O YJIT fornece performance significativamente maior, mas também usa mais memória do que o interpretador Ruby. O YJIT é recomendado para workloads Ruby on Rails.

Por padrão, o YJIT não está habilitado. Para habilitar o YJIT para uma função Ruby 3.2, defina a variável de ambiente `RUBY_YJIT_ENABLE` como `1`. Para confirmar que o YJIT está habilitado, imprima o resultado do método `RubyVM::YJIT.enabled?`.

Example : confirmação de que o YJIT está habilitado

```
puts(RubyVM::YJIT.enabled?())
# => true
```

Definir o manipulador de função do Lambda em Ruby

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. A função é executada até que o manipulador retorne uma resposta, seja encerrado ou atinja o tempo limite.

No exemplo a seguir, o arquivo `function.rb` define um método de handler chamado `handler`. A função do handler usa dois objetos como entrada e retorna um documento JSON.

Example function.rb

```
require 'json'

def handler(event:, context:)
    { event: JSON.generate(event), context: JSON.generate(context.inspect) }
end
```

Em sua configuração de função, a configuração `handler` informa o Lambda onde encontrar o manipulador. Para o exemplo anterior, o valor correto para essa configuração é **`function.handler`**. Ele inclui dois nomes separados por um ponto: o nome do arquivo e o nome do método do handler.

Você também pode definir o método de handler em uma classe. O exemplo a seguir define um método de handler chamado `process` em uma classe denominada `Handler` em um módulo chamado `LambdaFunctions`.

Example source.rb

```
module LambdaFunctions
  class Handler
    def self.process(event:,context:)
      "Hello!"
    end
  end
end
```

Nesse caso, a configuração do handler é **`source.LambdaFunctions::Handler.process`**.

Os dois objetos que o handler aceita são os eventos de invocação e contexto. O evento é um objeto Ruby que contém a carga que é fornecida pelo chamador. Se a carga útil for um documento JSON,

o objeto de evento é um hash do Ruby. Caso contrário, é uma string. O objeto de contexto tem métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de execução.

O manipulador da função é executado sempre que sua função do Lambda é invocada. O código estático fora do handler é executado uma vez por instância da função. Se o handler usar recursos como clientes SDK e conexões de banco de dados, você poderá criá-los fora do método de handler para reutilizá-los para várias invocações.

Cada instância de sua função pode processar vários eventos de invocação, mas ela processa apenas um evento por vez. O número de instâncias que processam um evento em um momento é a simultaneidade da sua função. Para obter mais informações sobre o ambiente de execução do Lambda, consulte Ambiente de execução do Lambda.

Como trabalhar com arquivos .zip para funções do Lambda em Ruby

O código de sua função do AWS Lambda compreende um arquivo .rb contendo o código do manipulador da função, em conjunto com quaisquer dependências adicionais (gems) das quais o código depende. Para implantar o código dessa função no Lambda, você usa um pacote de implantação. Esse pacote pode ser um arquivo .zip ou uma imagem de contêiner. Para obter mais informações sobre como usar imagens de contêiner com o Ruby, consulte Implantar funções do Lambda em Ruby com imagens de contêiner.

Para criar um pacote de implantação como arquivo .zip, você pode usar um utilitário de arquivo .zip integrado da ferramenta da linha de comando ou qualquer outro utilitário de arquivo .zip, como o 7zip. Os exemplos mostrados nas seções a seguir pressupõem que você esteja usando uma ferramenta `zip` da linha de comando em um ambiente Linux ou MacOS. Para usar os mesmos comandos no Windows, você pode instalar o Subsistema Windows para Linux para obter uma versão do Ubuntu e do Bash integrada ao Windows.

Observe que o Lambda usa permissões de arquivo POSIX, então pode ser necessário definir permissões para a pasta do pacote de implantação antes da criação do arquivo .zip.

Os comandos de exemplo nas seções a seguir usam o utilitário Bundler para adicionar dependências ao pacote de implantação. Para instalar o Bundler, execute o comando a seguir.

```
gem install bundler
```

Seções

- Dependências em Ruby
- Criar um pacote de implantação .zip sem dependências
- Criar uma implantação .zip empacotada com dependências
- Criar uma camada Ruby para suas dependências
- Criar pacotes de implantação .zip com bibliotecas nativas
- Criação e atualização de funções do Lambda em Ruby usando arquivos .zip

Dependências em Ruby

Para as funções do Lambda que usam o runtime do Ruby, uma dependência pode ser qualquer gem do Ruby. Ao implantar a função usando um arquivo .zip, você pode adicionar essas dependências ao arquivo .zip com o código da função ou usar uma camada do Lambda. Uma camada é um arquivo .zip que pode conter código adicional e outro conteúdo. Para saber mais sobre como usar as camadas do Lambda, consulte Camadas do Lambda.

O runtime do Ruby inclui o AWS SDK for Ruby. Caso a função use o SDK, você não precisa empacotá-lo com o código. No entanto, para manter o controle total das dependências ou para usar uma versão específica do SDK, você pode adicioná-lo ao pacote de implantação da função. É possível incluir o SDK em seu arquivo .zip ou adicioná-lo usando uma camada do Lambda. As dependências em seu arquivo .zip ou em camadas do Lambda têm precedência sobre as versões inclusas no runtime. Para descobrir qual versão do SDK para Ruby está incluída em sua versão de runtime, consulte the section called “Versões do SDK incluídas no runtime”.

No modelo de responsabilidade compartilhada da AWS, você é responsável pelo gerenciamento de todas as dependências dos pacotes de implantação das suas funções. Isso inclui a aplicação de atualizações e patches de segurança. Para atualizar as dependências no pacote de implantação da função, primeiro crie um novo arquivo .zip e depois carregue esse arquivo no Lambda. Consulte Criar uma implantação .zip empacotada com dependências e Criação e atualização de funções do Lambda em Ruby usando arquivos .zip para obter mais informações.

Criar um pacote de implantação .zip sem dependências

Se o código de função não tiver dependências, o arquivo .zip conterá somente o arquivo .rb com o código do manipulador da função. Use seu utilitário zip preferencial para criar um arquivo .zip com o arquivo .rb na raiz. Se o arquivo .rb não estiver na raiz do arquivo .zip, o Lambda não poderá executar seu código.

Para saber como implantar o arquivo .zip para criar uma função do Lambda ou atualizar uma já existente, consulte Criação e atualização de funções do Lambda em Ruby usando arquivos .zip.

Criar uma implantação .zip empacotada com dependências

Caso o código de função dependa de gems do Ruby adicionais, você pode adicionar essas dependências ao arquivo .zip com o código de função ou usar uma camada do Lambda. As instruções nesta seção mostram como incluir as dependências no pacote de implantação .zip. Para

obter instruções sobre como incluir suas dependências em uma camada, consulte the section called "Criar uma camada Ruby para suas dependências".

Suponha que o código de função esteja salvo em um arquivo denominado `lambda_function.rb` no diretório do seu projeto. Os exemplos de comandos da CLI a seguir criam um arquivo .zip denominado `my_deployment_package.zip` que contém o código da função e suas dependências.

Para criar o pacote de implantação

1. No diretório do seu projeto, crie um `Gemfile` para especificar as dependências.

   ```
   bundle init
   ```

2. Usando o editor de texto preferencial, edite o `Gemfile` para especificar as dependências da função. Por exemplo, para usar a gem TZInfo, edite o `Gemfile` para que se assemelhe ao apresentado a seguir.

   ```
   source "https://rubygems.org"
   gem "tzinfo"
   ```

3. Execute o comando a seguir para instalar as gems especificadas no `Gemfile` no diretório do seu projeto. Este comando define `vendor/bundle` como o caminho padrão para instalações de gem.

   ```
   bundle config set --local path 'vendor/bundle' && bundle install
   ```

   Você deve ver saída semelhante ao seguinte:

   ```
   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**
   >
   > Para instalar gems de forma global novamente em outro momento, execute o comando a seguir.

```
bundle config set --local system 'true'
```

4. Crie um arquivo .zip compactado contendo o arquivo `lambda_function.rb` com o código do manipulador da função e as dependências instaladas na etapa anterior.

```
zip -r my_deployment_package.zip lambda_function.rb vendor
```

Você deve ver saída semelhante ao seguinte:

```
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%)
...
```

Criar uma camada Ruby para suas dependências

As instruções nesta seção mostram como incluir suas dependências em uma camada. Para obter instruções sobre como incluir suas dependências em seu pacote de implantação, consulte the section called “Criar uma implantação .zip empacotada com dependências”.

Quando você adiciona uma camada a uma função, o Lambda carrega o conteúdo da camada no diretório `/opt` desse ambiente de execução. Para cada runtime do Lambda, a variável `PATH` já inclui caminhos de pasta específica no diretório `/opt`. Para garantir que a variável `PATH` colete o conteúdo da camada, o arquivo .zip da camada deve ter suas dependências nos seguintes caminhos de pasta:

- `ruby/gems/2.7.0` (`GEM_PATH`)
- `ruby/lib` (`RUBYLIB`)

Por exemplo, sua estrutura de arquivo .zip da sua camada pode ser assim:

```
json.zip
# ruby/gems/2.7.0/
```

```
                | build_info
                | cache
                | doc
                | extensions
                | gems
                | # json-2.1.0
                # specifications
                  # json-2.1.0.gemspec
```

Além disso, o Lambda detecta automaticamente todas as bibliotecas no diretório `/opt/lib` e quaisquer binários no diretório `/opt/bin`. Para garantir que o Lambda encontre corretamente o conteúdo da sua camada, você também pode criar uma camada com a seguinte estrutura:

```
custom-layer.zip
# lib
    | lib_1
    | lib_2
# bin
    | bin_1
    | bin_2
```

Depois de empacotar sua camada, consulte the section called “Criar e excluir camadas” e the section called “Adicionar camadas” para concluir sua configuração de camada.

Criar pacotes de implantação .zip com bibliotecas nativas

Muitas gems conhecidas do Ruby, como `nokogiri`, `nio4r` e `mysql`, contêm extensões nativas gravadas em C. Ao adicionar bibliotecas contendo código C ao pacote de implantação, é necessário compilar o pacote corretamente para garantir que seja compatível com o ambiente de execução do Lambda.

Para aplicações de produção, recomendamos desenvolver e implantar o código usando o AWS Serverless Application Model (AWS SAM). No AWS SAM, use a opção `sam build --use-container` para desenvolver a função dentro de um contêiner do Docker semelhante ao Lambda. Para saber mais sobre como usar o AWS SAM para implantar o código de função, consulte Building applications no Guia do desenvolvedor do AWS SAM.

Para criar um pacote de implantação .zip contendo gems com extensões nativas sem usar o AWS SAM, como alternativa, você pode usar um contêiner para empacotar suas dependências em um ambiente igual ao ambiente de runtime do Ruby no Lambda. Para concluir essas etapas, você

deve ter o Docker instalado em sua máquina de compilação. Para saber mais sobre a instalação do Docker, consulte Install Docker Engine.

Criar um pacote de implantação .zip em um contêiner do Docker

1. Crie uma pasta em sua máquina de compilação local para salvar o contêiner. Dentro dessa pasta, crie um arquivo chamado `dockerfile` e cole o código a seguir nele.

```
FROM public.ecr.aws/sam/build-ruby3.2:latest-x86_64
RUN gem update bundler
CMD "/bin/bash"
```

2. Dentro da pasta em que você criou o `dockerfile`, execute o comando a seguir para criar o contêiner do Docker.

```
docker build -t awsruby32 .
```

3. Navegue até o diretório do projeto que contém o arquivo `.rb` com o código do manipulador da função e o `Gemfile` especificando as dependências da função. De dentro desse diretório, execute o comando a seguir para iniciar o contêiner do Ruby no Lambda.

   Linux/MacOS

```
docker run --rm -it -v $PWD:/var/task -w /var/task awsruby32
```

   > **Note**
   >
   > No MacOS, você pode visualizar um aviso informando que a plataforma da imagem solicitada não corresponde à plataforma de host detectada. Ignore esse aviso.

   Windows PowerShell

```
docker run --rm -it -v ${pwd}:var/task -w /var/task awsruby32
```

   Quando o contêiner for iniciado, você deverá ver um aviso de bash.

```
bash-4.2#
```

4. Configure o utilitário empacotado para instalar as gems especificadas no `Gemfile` em um diretório local de `vendor/bundle` e instale as dependências.

```
bash-4.2# bundle config set --local path 'vendor/bundle' && bundle install
```

5. Crie o pacote de implantação .zip com o código de função e as dependências. Neste exemplo, o arquivo que contém o código do manipulador da função é denominado `lambda_function.rb`.

```
bash-4.2# zip -r my_deployment_package.zip lambda_function.rb vendor
```

6. Saia do contêiner e retorne ao diretório local do projeto.

```
bash-4.2# exit
```

   Agora é possível usar o pacote de implantação de arquivo .zip para criar ou atualizar sua função do Lambda. Consulte Criação e atualização de funções do Lambda em Ruby usando arquivos .zip

Criação e atualização de funções do Lambda em Ruby usando arquivos .zip

Depois de criar seu pacote de implantação .zip, você poderá usá-lo para criar uma nova função do Lambda ou atualizar uma existente. É possível implantar o pacote .zip usando o console do Lambda, a AWS Command Line Interface e a API do Lambda. Você também pode criar e atualizar funções do Lambda usando o AWS Serverless Application Model (AWS SAM) e o AWS CloudFormation.

O tamanho máximo de um pacote de implantação .zip para o Lambda é 250 MB (descompactado). Esse limite se aplica ao tamanho combinado de todos os arquivos que você carrega, inclusive qualquer camada do Lambda.

O runtime do Lambda precisa de permissão para ler os arquivos no pacote de implantação. Na notação octal de permissões do Linux, o Lambda precisa de 644 permissões para arquivos não executáveis (rw-r--r--) e 755 permissões (rwxr-xr-x) para diretórios e arquivos executáveis.

No Linux e no MacOS, use o comando `chmod` para alterar as permissões de arquivo em arquivos e diretórios do seu pacote de implantação. Por exemplo, para dar a um arquivo executável as permissões corretas, execute o comando a seguir.

```
chmod 755 <filepath>
```

Para alterar as permissões de arquivo no Windows, consulte Set, View, Change, or Remove Permissions on an Object na documentação do Microsoft Windows.

Criar e atualizar funções com arquivos .zip usando o console

Para criar uma nova função, você deve primeiro criar a função no console e depois carregar o arquivo .zip. Para atualizar uma função existente, abra a página da função e siga o mesmo procedimento para adicionar o arquivo .zip atualizado.

Se o arquivo .zip for menor que 50 MB, você poderá criar ou atualizar uma função carregando o arquivo diretamente da máquina local. Para arquivos .zip maiores que 50 MB, você deve primeiro carregar o pacote para um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando o AWS Management Console, consulte Conceitos básicos do Amazon S3. Para carregar arquivos usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

> **Note**
>
> Você não pode alterar o tipo de pacote de implantação (.zip ou imagem de contêiner) de uma função existente. Por exemplo, você não pode converter uma função de imagem de contêiner para usar um arquivo compactado .zip. É necessário criar uma nova função.

Para criar uma função (console)

1. Abra a página Funções do console do Lambda e escolha Criar função.
2. Escolha Author from scratch (Criar do zero).
3. Em Basic information (Informações básicas), faça o seguinte:
   a. Em Nome da função, insira o nome da função.
   b. Em Runtime, selecione o runtime que você deseja usar.
   c. (Opcional) Em Arquitetura, escolha a arquitetura do conjunto de instruções para a função. O valor da arquitetura padrão é X86_64. Certifique-se de que o pacote de implantação .zip da função seja compatível com a arquitetura do conjunto de instruções que você escolheu.
4. (Opcional) Em Permissões, expanda Alterar função de execução padrão. Crie uma função de execução ou use uma existente.

5. Escolha a opção Criar função. O Lambda cria uma função básica “Hello world” usando o runtime escolhido.

Você pode carregar o arquivo .zip da máquina local (console)

1. Na página Funções do console Lambda, escolha a função para a qual você deseja carregar o arquivo .zip.
2. Selecione a guia Código.
3. No painel do Código-fonte, escolha Carregar de.
4. Escolha o arquivo .zip.
5. Para carregar o arquivo .zip, faça o seguinte:
   a. Selecione Carregar e, em seguida, selecione o arquivo .zip no seletor de arquivos.
   b. Escolha Open (Abrir).
   c. Escolha Salvar.

Para carregar um arquivo .zip de um bucket do Amazon S3 (console)

1. Na página Funções do console do Lambda, escolha a função para a qual você deseja carregar um novo arquivo .zip.
2. Selecione a guia Código.
3. No painel do Código-fonte, escolha Carregar de.
4. Escolha Local do Amazon S3.
5. Cole o URL do link do Amazon S3 do arquivo .zip e escolha Salvar.

Atualizar funções do arquivo .zip usando o editor de código do console

Para algumas funções com pacotes de implantação .zip, você pode usar o editor de código integrado do console do Lambda para atualizar o código da função diretamente. Para usar esse recurso, a função deve atender aos seguintes critérios:

- A função deve usar um dos runtimes da linguagem interpretada (Python, Node.js ou Ruby)
- O pacote de implantação da função deve ser menor que 3 MB.

O código das funções com pacotes de implantação de imagens de contêiner não pode ser editado diretamente no console.

Para atualizar o código da função usando o editor de código do console

1. Abra a página Funções do console do Lambda e selecione a função.
2. Selecione a guia Código.
3. No painel Código-fonte, selecione o arquivo de código-fonte e edite-o no editor de código integrado.
4. Quando terminar de editar o código, escolha Implantar para salvar as alterações e atualizar a função.

Criar e atualizar funções com arquivos .zip usando a AWS CLI

Você pode usar a AWS CLI para criar uma função ou atualizar uma existente usando um arquivo .zip. Use os comandos create-function e update-function-code para implantar o pacote .zip. Se o arquivo .zip for menor que 50 MB, você poderá carregar o pacote .zip de um local do arquivo na máquina de compilação local. Para arquivos .zip maiores, você deve carregar o pacote .zip de um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

> **Note**
>
> Se você carregar o arquivo .zip de um bucket do Amazon S3 usando a AWS CLI, o bucket deverá estar na mesma Região da AWS que sua função.

Para criar uma função usando um arquivo .zip com a AWS CLI, você deve especificar o seguinte:

- O nome da função (`--function-name`)
- O runtime da função (`--runtime`)
- O nome do recurso da Amazon (ARN) da função de execução da função (`--role`)
- O nome do método do manipulador no código da função (`--handler`)

Você também deve especificar a local do arquivo .zip. Se o arquivo .zip estiver localizado em uma pasta da máquina de compilação local, use a opção `--zip-file` para especificar o caminho do arquivo, conforme mostrado no comando do exemplo a seguir.

```
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
```

Para especificar o local do arquivo .zip em um bucket do Amazon S3, use a opção `--code` conforme mostrado no comando do exemplo a seguir. Você só precisa usar o parâmetro `S3ObjectVersion` para objetos com versionamento.

```
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
```

Para atualizar uma função existente usando a CLI, especifique o nome da função usando o parâmetro `--function-name`. Você também deve especificar o local do arquivo .zip que deseja usar para atualizar o código da função. Se o arquivo .zip estiver localizado em uma pasta da máquina de compilação local, use a opção `--zip-file` para especificar o caminho do arquivo, conforme mostrado no comando do exemplo a seguir.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Para especificar o local do arquivo .zip em um bucket do Amazon S3, use as opções `--s3-bucket` e `--s3-key` conforme mostrado no comando do exemplo a seguir. Você só precisa usar o parâmetro `--s3-object-version` para objetos com versionamento.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket DOC-EXAMPLE-BUCKET --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Criar e atualizar funções com arquivos .zip usando a API do Lamba

Para criar e atualizar funções usando um arquivo .zip, use as seguintes operações de API:

- CreateFunction
- UpdateFunctionCode

Criar e atualizar funções com arquivos .zip usando o AWS SAM

O AWS Serverless Application Model (AWS SAM) é um kit de ferramentas que ajuda a simplificar o processo de criação e execução de aplicações com tecnologia sem servidor na AWS. Você define os recursos para a aplicação em um modelo YAML ou JSON e usa a interface da linha de comando do AWS SAM (CLI do AWS SAM) para criar, empacotar e implantar aplicações. Quando você cria uma função do Lambda com base em um modelo do AWS SAM, o AWS SAM cria automaticamente um pacote de implantação .zip ou uma imagem de contêiner com o código da função e quaisquer dependências que você especificar. Para saber mais sobre como usar o AWS SAM para criar e implantar funções do Lambda, consulte Conceitos básicos do AWS SAM no Guia do desenvolvedor do AWS Serverless Application Model.

Você também pode usar o AWS SAM para criar uma função do Lambda usando um arquivo .zip existente. Para criar uma função do Lambda usando o AWS SAM, salve o arquivo .zip em um bucket do Amazon S3 ou em uma pasta local na máquina de compilação. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

No modelo do AWS SAM, o recurso `AWS::Serverless::Function` especifica a função do Lambda. Nesse recurso, defina as seguintes propriedades para criar uma função usando um arquivo .zip:

- `PackageType`: definir como `Zip`
- `CodeUri`: definir como o URI do Amazon S3 do código da função, o caminho para a pasta local ou o objeto FunctionCode
- `Runtime`: definir como o runtime escolhido

Com o AWS SAM, se o arquivo .zip for maior que 50 MB, você não precisará carregá-lo primeiro em um bucket do Amazon S3. O AWS SAM poderá carregar pacotes .zip com o tamanho máximo permitido de 250 MB (descompactados) de um local da máquina de compilação local.

Para saber mais sobre a implantação de funções usando o arquivo .zip no AWS SAM, consulte AWS::Serverless::Function no Guia do desenvolvedor do AWS SAM.

Criar e atualizar funções com arquivos .zip usando o AWS CloudFormation

Você pode usar o AWS CloudFormation para criar uma função do Lambda usando um arquivo .zip. Para criar uma função do Lambda de um arquivo.zip, primeiro carregue o arquivo em um bucket do

Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

No modelo do AWS CloudFormation, o recurso `AWS::Lambda::Function` especifica a função do Lambda. Nesse recurso, defina as seguintes propriedades para criar uma função usando um arquivo .zip:

- `PackageType`: definir como `Zip`
- `Code`: inserir o nome do bucket do Amazon S3 e o nome do arquivo .zip nos campos `S3Bucket` e `S3Key`
- `Runtime`: definir como o runtime escolhido

O arquivo .zip que o AWS CloudFormation gera não pode exceder 4 MB. Para saber mais sobre a implantação de funções usando o arquivo .zip no AWS CloudFormation, consulte AWS::Lambda::Function no Guia do desenvolvedor do AWS CloudFormation.

Implantar funções do Lambda em Ruby com imagens de contêiner

Existem três maneiras de criar uma imagem de contêiner para uma função do Lambda em Ruby:

- Usar uma imagem base da AWS para Ruby

  As imagens base da AWS são pré-carregadas com um runtime de linguagem, um cliente de interface de runtime para gerenciar a interação entre o Lambda e o código da sua função e um emulador de interface de runtime para testes locais.

- Usar uma imagem base somente para sistema operacional da AWS

  As imagens base somente para sistema operacional da AWS contêm uma distribuição do Amazon Linux e o emulador de interface de runtime. Essas imagens são comumente usadas para criar imagens de contêiner para linguagens compiladas, como Go e Rust e para uma linguagem ou versão de linguagem para a qual o Lambda não fornece uma imagem base, como Node.js 19. Você também pode usar imagens base somente para sistema operacional para implementar um runtime personalizado. Para tornar a imagem compatível com o Lambda, você deve incluir o cliente de interface de runtime do Ruby na imagem.

- Usar uma imagem base que não é da AWS

  Você também pode usar uma imagem base alternativa de outro registro de contêiner, como Alpine Linux ou Debian. Você também pode usar uma imagem personalizada criada por sua organização. Para tornar a imagem compatível com o Lambda, você deve incluir o cliente de interface de runtime do Ruby na imagem.

> **Tip**
>
> Para reduzir o tempo necessário para que as funções do contêiner do Lambda se tornem ativas, consulte Use multi-stage builds na documentação do Docker. Para criar imagens de contêiner eficientes, siga as Melhores práticas para gravar Dockerfiles.

Esta página explica como criar, testar e implantar imagens de contêiner para o Lambda.

Tópicos

- Usar uma imagem base alternativa com o cliente da interface de runtime

Imagens base da AWS para Ruby

A AWS oferece as seguintes imagens base para Ruby:

Tags	Runtime	Sistema operacional	Dockerfile	Desaprovação
3.3	Ruby 3.3	Amazon Linux 2023	Dockerfile para Ruby 3.3 no GitHub	
3.2	Ruby 3.2	Amazon Linux 2	Dockerfile para Ruby 3.2 no GitHub	

Repositório do Amazon ECR: gallery.ecr.aws/lambda/ruby

Usar uma imagem base da AWS para Ruby

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- Docker
- Ruby

Criação de uma imagem a partir de uma imagem base

Para criar uma imagem de contêiner para Ruby

1. Crie um diretório para o projeto e depois mude para esse diretório.

```
mkdir example
cd example
```

2. Crie um novo arquivo chamado `Gemfile`. É aqui que você lista os pacotes do RubyGems necessários da sua aplicação. O AWS SDK for Ruby está disponível no RubyGems. Você deve

escolher gems de serviço da AWS específicos para instalar. Por exemplo, para usar o gem do Ruby para Lambda, seu Gemfile deve ser:

```
source 'https://rubygems.org'

gem 'aws-sdk-lambda'
```

Como alternativa, o gem aws-sdk contém todos os gems de serviço da AWS disponíveis. Esse gem é muito grande. Recomendamos que ele só seja usado se você depender de muitos serviços da AWS.

3. Instale as dependências especificadas no Gemfile usando instalação de pacote.

```
bundle install
```

4. Crie um novo arquivo chamado `lambda_function.rb`. É possível adicionar o exemplo de código de função a seguir ao arquivo para testes ou usar o seu próprio código.

   Example Função Ruby

```
module LambdaFunction
  class Handler
    def self.process(event:,context:)
      "Hello from Lambda!"
    end
  end
end
```

5. Crie um novo Dockerfile. É mostrado a seguir um exemplo de Dockerfile que usa uma imagem base da AWS. Esse Dockerfile usa a seguinte configuração:

   - Defina a propriedade `FROM` como o URI da imagem base.
   - Use o comando COPY para copiar o código da função e as dependências do runtime para `{LAMBDA_TASK_ROOT}`, uma variável de ambiente definida pelo Lambda.
   - Defina o argumento `CMD` como o manipulador de funções do 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. Crie a imagem do Docker com o comando docker build. O exemplo a seguir nomeia a imagem como `docker-image` e atribui a ela a tag `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> O comando especifica a opção `--platform linux/amd64` para garantir que seu contêiner seja compatível com o ambiente de execução do Lambda, independentemente da arquitetura da sua máquina de compilação. Se você pretende criar uma função do Lambda usando a arquitetura do conjunto de instruções ARM64, certifique-se de alterar o comando para usar a opção `--platform linux/arm64` em vez disso.

(Opcional) Teste a imagem localmente

1. Inicie a imagem do Docker com o comando docker run. Neste exemplo, `docker-image` é o nome da imagem e `test` é a tag.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Esse comando executa a imagem como um contêiner e cria um endpoint local em `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se você criou a imagem do Docker para a arquitetura do conjunto de instruções ARM64, certifique-se de usar a opção `--platform linux/arm64`, em vez de `--platform linux/amd64`.

2. Em uma nova janela de terminal, publique um evento no endpoint local.

   Linux/macOS

   No Linux e no MacOS, execute o seguinte comando `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   No PowerShell, execute o seguinte comando `Invoke-WebRequest`:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
    "application/json"
   ```

3. Obtenha o ID do contêiner.

```
docker ps
```

4. Use o comando docker kill para parar o contêiner. Nesse comando, substitua `3766c4ab331c` pelo ID do contêiner da etapa anterior.

```
docker kill 3766c4ab331c
```

Implantação da imagem

Para enviar a imagem ao Amazon ECR e criar a função do Lambda

1. Execute o comando get-login-password para autenticar a CLI do Docker no seu registro do Amazon ECR.

   - Defina o valor `--region` para a Região da AWS onde você deseja criar o repositório do Amazon ECR.
   - Substituir `111122223333` por seu ID da Conta da 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. Crie um repositório no Amazon ECR usando o 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**
>
> O repositório do Amazon ECR deve estar na mesma Região da AWS que a função do Lambda.

Se tiver êxito, você verá uma resposta como esta:

```
{
    "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. Copie o `repositoryUri` da saída na etapa anterior.
4. Execute o comando docker tag para aplicar uma tag na sua imagem local em seu repositório do Amazon ECR como a versão mais recente. Neste comando:

   - Substitua `docker-image:test` pelo nome e tag da sua imagem do Docker.
   - Substitua `<ECRrepositoryUri>` pelo `repositoryUri` que você copiou. Certifique-se de incluir `:latest` no final do URI.

   ```
   docker tag docker-image:test <ECRrepositoryUri>:latest
   ```

   Exemplo:

   ```
   docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
   world:latest
   ```

5. Execute o comando docker push para implantar a imagem local no repositório do Amazon ECR. Certifique-se de incluir `:latest` no final do URI do repositório.

   ```
   docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
   ```

6. Crie um perfil de execução para a função, caso ainda não tenha um. Você precisará do nome do recurso da Amazon (ARN) do perfil na próxima etapa.

7. Criar a função do Lambda. Em `ImageUri`, especifique o URI do repositório anterior. Certifique-se de incluir `:latest` no final do 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**
>
> É possível criar uma função usando uma imagem em uma conta da AWS diferente desde que a imagem esteja na mesma região da função do Lambda. Para ter mais informações, consulte Permissões entre contas do Amazon ECR.

8. Invoque a função.

```
aws lambda invoke --function-name hello-world response.json
```

Você obterá uma resposta parecida com esta:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Para ver a saída da função, verifique o arquivo `response.json`.

Para atualizar o código da função, você deve criar a imagem novamente, fazer upload da nova imagem no repositório do Amazon ECR e, em seguida, usar o comando update-function-code para implantar a imagem na função do Lambda.

O Lambda resolve a tag de imagem em um resumo de imagem específico. Isso significa que, se você apontar a tag de imagem que foi usada para implantar a função em uma nova imagem no Amazon ECR, o Lambda não atualizará automaticamente a função para usar a nova imagem. Para implantar a nova imagem na mesma função do Lambda, você deverá usar o comando `update-function-code`, mesmo que a tag da imagem no Amazon ECR permaneça a mesma.

Usar uma imagem base alternativa com o cliente da interface de runtime

Se você usar uma imagem base somente para sistema operacional ou uma imagem base alternativa, deverá incluir o cliente de interface de runtime na imagem. O cliente de interface de runtime estende API de tempo de execução do Lambda, que gerencia a interação entre o Lambda e o código da sua função.

Instale o cliente de interface de runtime para Ruby usando o gerenciador de pacotes RubyGems.org:

```
gem install aws_lambda_ric
```

Você também pode fazer download do cliente de interface de runtime do Ruby no GitHub. O cliente da interface de runtime é compatível com as versões 2.5.x a 2.7.x do Ruby.

O exemplo a seguir demonstra como criar uma imagem de contêiner para Ruby usando uma imagem base que não é da AWS. O exemplo de Dockerfile usa uma imagem base oficial do Ruby. O Dockerfile inclui o cliente de interface de runtime.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- Docker
- Ruby

Criar uma imagem de uma imagem base alternativa

Para criar uma imagem de contêiner usando uma imagem base alternativa do Ruby

1. Crie um diretório para o projeto e depois mude para esse diretório.

   ```
   mkdir example
   cd example
   ```

2. Crie um novo arquivo chamado `Gemfile`. É aqui que você lista os pacotes do RubyGems necessários da sua aplicação. O AWS SDK for Ruby está disponível no RubyGems. Você deve escolher gems de serviço da AWS específicos para instalar. Por exemplo, para usar o gem do Ruby para Lambda, seu Gemfile deve ser:

```
source 'https://rubygems.org'

gem 'aws-sdk-lambda'
```

Como alternativa, o gem aws-sdk contém todos os gems de serviço da AWS disponíveis. Esse gem é muito grande. Recomendamos que ele só seja usado se você depender de muitos serviços da AWS.

3. Instale as dependências especificadas no Gemfile usando instalação de pacote.

```
bundle install
```

4. Crie um novo arquivo chamado `lambda_function.rb`. É possível adicionar o exemplo de código de função a seguir ao arquivo para testes ou usar o seu próprio código.

   Example Função Ruby

```
module LambdaFunction
  class Handler
    def self.process(event:,context:)
      "Hello from Lambda!"
    end
  end
end
```

5. Crie um novo Dockerfile. O Dockerfile a seguir usa uma imagem base do Ruby em vez de uma imagem base da AWS. O Dockerfile inclui o cliente de interface de runtime do Ruby, o que torna a imagem compatível com o Lambda. Como alternativa, você pode adicionar o cliente da interface de runtime ao Gemfile da aplicação.

   - Defina a propriedade `FROM` como a imagem básica do Ruby.
   - Crie um diretório para o código da função e uma variável de ambiente que aponte para esse diretório. Neste exemplo, o diretório é `/var/task`, que espelha o ambiente de execução do Lambda. No entanto, você pode escolher qualquer diretório para o código da função, pois o Dockerfile não usa uma imagem básica da AWS.
   - Defina o `ENTRYPOINT` como o módulo em que você deseja que o contêiner do Docker seja executado quando for iniciado. Nesse caso, o módulo é o cliente de interface de runtime.
   - Defina o argumento `CMD` como o manipulador de funções do 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. Crie a imagem do Docker com o comando docker build. O exemplo a seguir nomeia a imagem como `docker-image` e atribui a ela a tag `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> O comando especifica a opção `--platform linux/amd64` para garantir que seu contêiner seja compatível com o ambiente de execução do Lambda, independentemente da arquitetura da sua máquina de compilação. Se você pretende criar uma função do Lambda usando a arquitetura do conjunto de instruções ARM64, certifique-se de alterar o comando para usar a opção `--platform linux/arm64` em vez disso.

(Opcional) Teste a imagem localmente

Use o emulador de interface de runtime para testar a imagem localmente. Você pode compilar o emulador em sua imagem ou usar o procedimento a seguir instalá-lo na sua máquina local.

Para instalar o emulador de interface de runtime na sua máquina local

1. No diretório do projeto, execute o comando a seguir para baixar o emulador de interface de runtime (arquitetura x86-64) do GitHub e instalá-lo na sua máquina local.

   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
   ```

   Para instalar o emulador arm64, substitua o URL do repositório do GitHub no comando anterior pelo seguinte:

   ```
   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
```

Para instalar o emulador de arm64, substitua `$downloadLink` pelo seguinte:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Inicie a imagem do Docker com o comando docker run. Observe o seguinte:

   - `docker-image` é o nome da imagem e `test` é a tag.
   - `aws_lambda_ric lambda_function.LambdaFunction::Handler.process` é o `ENTRYPOINT` seguido pelo `CMD` do 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
   ```

   Esse comando executa a imagem como um contêiner e cria um endpoint local em `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se você criou a imagem do Docker para a arquitetura do conjunto de instruções ARM64, certifique-se de usar a opção `--platform linux/arm64`, em vez de `--platform linux/amd64`.

3. Publique um evento no endpoint local.

   Linux/macOS

   No Linux e no MacOS, execute o seguinte comando `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   No PowerShell, execute o seguinte comando `Invoke-WebRequest`:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
    "application/json"
   ```

4. Obtenha o ID do contêiner.

```
docker ps
```

5. Use o comando docker kill para parar o contêiner. Nesse comando, substitua `3766c4ab331c` pelo ID do contêiner da etapa anterior.

```
docker kill 3766c4ab331c
```

Implantação da imagem

Para enviar a imagem ao Amazon ECR e criar a função do Lambda

1. Execute o comando get-login-password para autenticar a CLI do Docker no seu registro do Amazon ECR.

   - Defina o valor `--region` para a Região da AWS onde você deseja criar o repositório do Amazon ECR.
   - Substituir `111122223333` por seu ID da Conta da 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. Crie um repositório no Amazon ECR usando o 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**
>
> O repositório do Amazon ECR deve estar na mesma Região da AWS que a função do Lambda.

Se tiver êxito, você verá uma resposta como esta:

```
{
    "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. Copie o `repositoryUri` da saída na etapa anterior.
4. Execute o comando docker tag para aplicar uma tag na sua imagem local em seu repositório do Amazon ECR como a versão mais recente. Neste comando:

   - Substitua `docker-image:test` pelo nome e tag da sua imagem do Docker.
   - Substitua `<ECRrepositoryUri>` pelo `repositoryUri` que você copiou. Certifique-se de incluir `:latest` no final do URI.

   ```
   docker tag docker-image:test <ECRrepositoryUri>:latest
   ```

   Exemplo:

   ```
   docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
   world:latest
   ```

5. Execute o comando docker push para implantar a imagem local no repositório do Amazon ECR. Certifique-se de incluir `:latest` no final do URI do repositório.

   ```
   docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
   ```

6. Crie um perfil de execução para a função, caso ainda não tenha um. Você precisará do nome do recurso da Amazon (ARN) do perfil na próxima etapa.

7. Criar a função do Lambda. Em `ImageUri`, especifique o URI do repositório anterior. Certifique-se de incluir `:latest` no final do 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**
>
> É possível criar uma função usando uma imagem em uma conta da AWS diferente desde que a imagem esteja na mesma região da função do Lambda. Para ter mais informações, consulte Permissões entre contas do Amazon ECR.

8. Invoque a função.

```
aws lambda invoke --function-name hello-world response.json
```

Você obterá uma resposta parecida com esta:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Para ver a saída da função, verifique o arquivo `response.json`.

Para atualizar o código da função, você deve criar a imagem novamente, fazer upload da nova imagem no repositório do Amazon ECR e, em seguida, usar o comando update-function-code para implantar a imagem na função do Lambda.

O Lambda resolve a tag de imagem em um resumo de imagem específico. Isso significa que, se você apontar a tag de imagem que foi usada para implantar a função em uma nova imagem no Amazon ECR, o Lambda não atualizará automaticamente a função para usar a nova imagem. Para implantar a nova imagem na mesma função do Lambda, você deverá usar o comando `update-function-code`, mesmo que a tag da imagem no Amazon ECR permaneça a mesma.

AWS LambdaObjeto de contexto em Ruby

Quando o Lambda executa a função, ele transmite um objeto de contexto para o handler. Esse objeto fornece métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de execução.

Métodos de contexto

- `get_remaining_time_in_millis`: retorna o número de milissegundos restantes antes do tempo limite da execução.

Propriedades de contexto

- `function_name`: o nome da função do Lambda.
- `function_version`: a versão da função.
- `invoked_function_arn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `memory_limit_in_mb`: a quantidade de memória alocada para a função.
- `aws_request_id`: o identificador da solicitação de invocação.
- `log_group_name`: o grupo de logs da função.
- `log_stream_name`: a transmissão de log para a instância da função.
- `deadline_ms`: a data em que a função expira em milissegundos de tempo do Unix.
- `identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
- `client_context`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.

Registro em log da função do AWS Lambda em Ruby

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome e envia logs para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente do runtime do Lambda envia detalhes sobre cada invocação à transmissão de logs e transmite os logs e outras saídas do código de sua função. Para ter mais informações, consulte Usar logs do Amazon CloudWatch com o AWS Lambda.

Esta página descreve como produzir a saída de logs usando o código de sua função do Lambda ou acessar os logs usando a AWS Command Line Interface, o console do Lambda ou o console do CloudWatch.

Seções

Criar uma função que retorna logs

Para gerar os logs do seu código de função, você pode usar instruções `puts` ou qualquer biblioteca de logs que grave no `stdout` ou no `stderr`. O exemplo a seguir registra em log os valores das variáveis de ambiente e o objeto do 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 do 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
```

O runtime do Ruby registra em log as linhas `START`, `END` e `REPORT` para cada invocação. A linha do relatório fornece os detalhes a seguir.

RELATAR campos de dados de linha

- RequestId: o ID de solicitação exclusivo para a invocação.
- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.
- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o runtime levou para carregar a função e executar o código fora do método do handler.
- XRAY TraceId: para solicitações rastreadas, o ID de rastreamento do AWS X-Ray.
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Para obter logs mais detalhados, use a the section called “Biblioteca do Logger”.

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda.

Se seu código puder ser testado no editor de Código incorporado, você encontrará logs nos resultados de execução. Ao usar o recurso de teste do console para invocar uma função, você encontrará Saída de log na seção Detalhes.

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a página Log groups (Grupos de log).
2. Escolha o grupo de logs de sua função (/aws/lambda/***nome-de-sua-função***).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma instância da sua função. Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos intrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- AWS CLI – Configuração rápida com `aws configure`

Você pode usar a AWS CLI para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

A seguinte saída deverá ser mostrada:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `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
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

A seguinte saída deverá ser mostrada:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e Ubuntu no Windows. Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa `sed` para remover as aspas do arquivo de saída e fica inativo por 15 segundos

para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 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 macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod -R 755 get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
}
```


Excluir logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ou[configurar um período de retenção](#)após o qual os logs são excluídos automaticamente.

Biblioteca do Logger

A biblioteca do logger do Ruby retorna logs simplificados que são facilmente lidos. Use o utilitário logger para gerar informações detalhadas, mensagens e códigos de erro relacionados à sua função.

```
# 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
```

A saída de `logger` inclui o nível do log, o timestamp e o ID da solicitação.

```
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
```

Instrumentar o código Ruby no AWS Lambda

O Lambda se integra ao AWS X-Ray para permitir que você rastreie, depure e otimize aplicativos do Lambda. É possível usar o X-Ray para rastrear uma solicitação à medida que ela atravessa recursos na aplicação, da API de frontend ao armazenamento e aos banco de dados no backend. Ao simplesmente adicionar a biblioteca do X-Ray SDK à configuração de compilação, é possível registrar erros e latência para qualquer chamada que a função faça para um serviço da AWS.

Após configurar o rastreamento ativo, você pode observar solicitações específicas por meio da aplicação. O grafo de serviço do X-Ray exibe informações sobre sua aplicação e todos os componentes. A imagem a seguir demonstra uma aplicação com duas funções. A função principal processa eventos e, às vezes, retorna erros. A segunda função de cima para baixo processa erros que aparecem no primeiro grupo de logs e usa o AWS SDK para chamar o X-Ray, o Amazon Simple Storage Service (Amazon S3) e o Amazon CloudWatch Logs.

Para alternar o rastreamento ativo na sua função do Lambda usando o console, siga as etapas abaixo:

Para ativar o rastreamento ativo

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e depois Monitoring and operations tools (Ferramentas de monitoramento e operações).
4. Selecione a opção Editar.

5. Em X-Ray, ative a opção Active tracing (Rastreamento ativo).
6. Escolha Salvar.

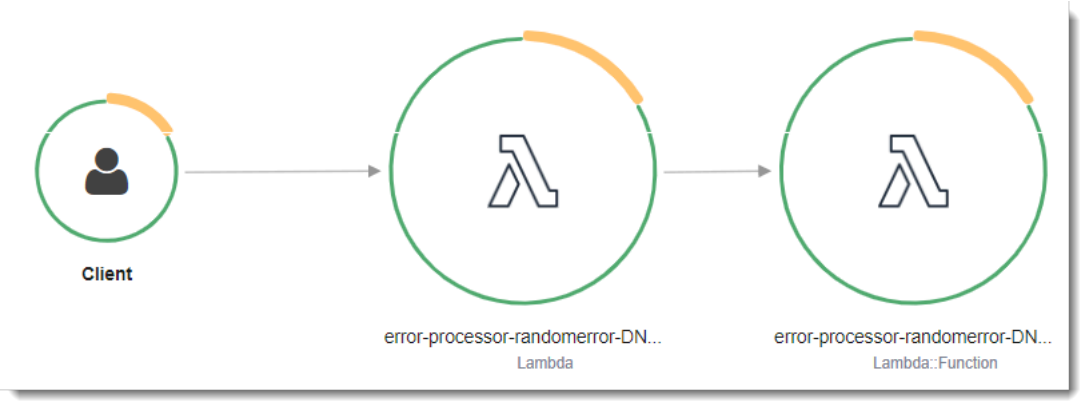
> **Definição de preço**
>
> Você pode usar o rastreamento do X-Ray gratuitamente todos os meses até determinado limite como parte do nível gratuito da AWS. Além do limite, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter mais informações, consulte Preços do AWS X-Ray.

Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você ativa o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à função de execução da função. Caso contrário, adicione a política AWSXRayDaemonWriteAccess à função de execução.

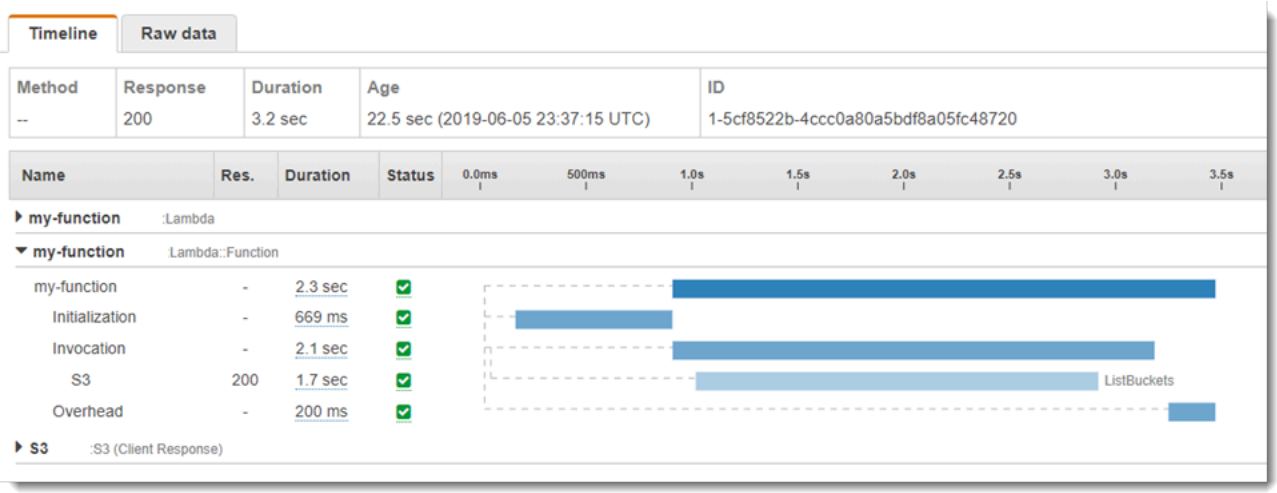
O X-Ray não rastreia todas as solicitações para sua aplicação. O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento seja eficiente, enquanto ainda fornece uma amostra representativa das solicitações. A taxa de amostragem é uma solicitação por segundo e 5% de solicitações adicionais.

> **Note**
>
> Não é possível configurar a taxa de amostragem do X-Ray para suas funções.

No X-Ray, um rastreamento registra informações sobre uma solicitação que é processada por um ou mais serviços. O Lambda registra dois segmentos por rastreamento, o que cria dois nós no gráfico de serviços. A imagem a seguir destaca esses dois nós:



O primeiro nó à esquerda representa o serviço do Lambda, que recebe a solicitação de invocação. O segundo nó representa a sua função do Lambda específica. O exemplo a seguir mostra um rastreamento com esses dois segmentos. Ambos têm o nome my-function, mas um tem a origem `AWS::Lambda` e o outro, a origem `AWS::Lambda::Function`. Se o segmento `AWS::Lambda` mostrar um erro, o serviço Lambda teve um problema. Se o segmento `AWS::Lambda::Function` mostrar um erro, sua função teve um problema.



Este exemplo expande o segmento `AWS::Lambda::Function` para mostrar seus três subsegmentos:

- Inicialização: representa o tempo gasto carregando a função e executando o código de inicialização. Esse subsegmento aparece somente para o primeiro evento que cada instância da função processa.
- Invocação: representa o tempo gasto na execução do código do manipulador.
- Sobrecarga: representa o tempo gasto pelo runtime do Lambda preparando-se para lidar com o próximo evento.

É possível instrumentar o código do manipulador para gravar metadados e rastrear chamadas downstream. Para registrar detalhes sobre chamadas feitas pelo manipulador para outros recursos e serviços, use o X-Ray SDK for Ruby. Para obter o SDK, adicione o pacote `aws-xray-sdk` às dependências do aplicativo.

Example blank-ruby/function/Gemfile

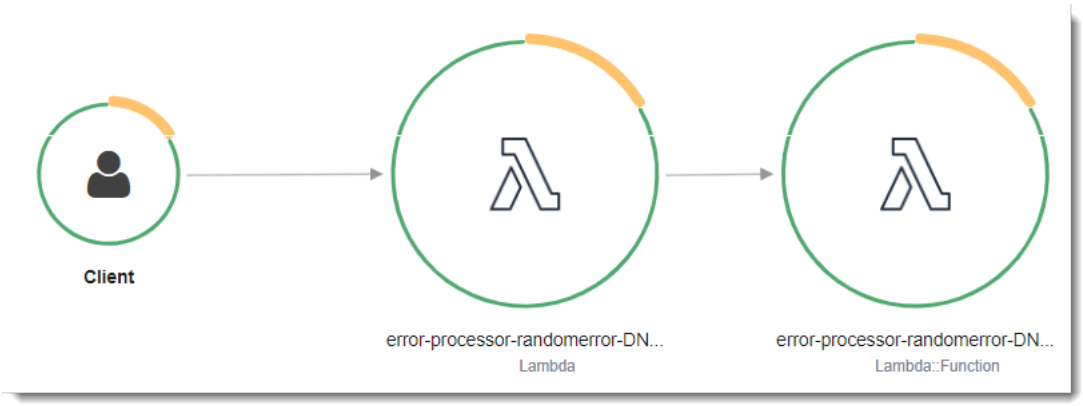
```
# Gemfile
source 'https://rubygems.org'

gem 'aws-xray-sdk', '0.11.4'
gem 'aws-sdk-lambda', '1.39.0'
gem 'test-unit', '3.3.5'
```

Para instrumentar clientes do AWS SDK, solicite o módulo `aws-xray-sdk/lambda` depois de criar um cliente no código de inicialização.

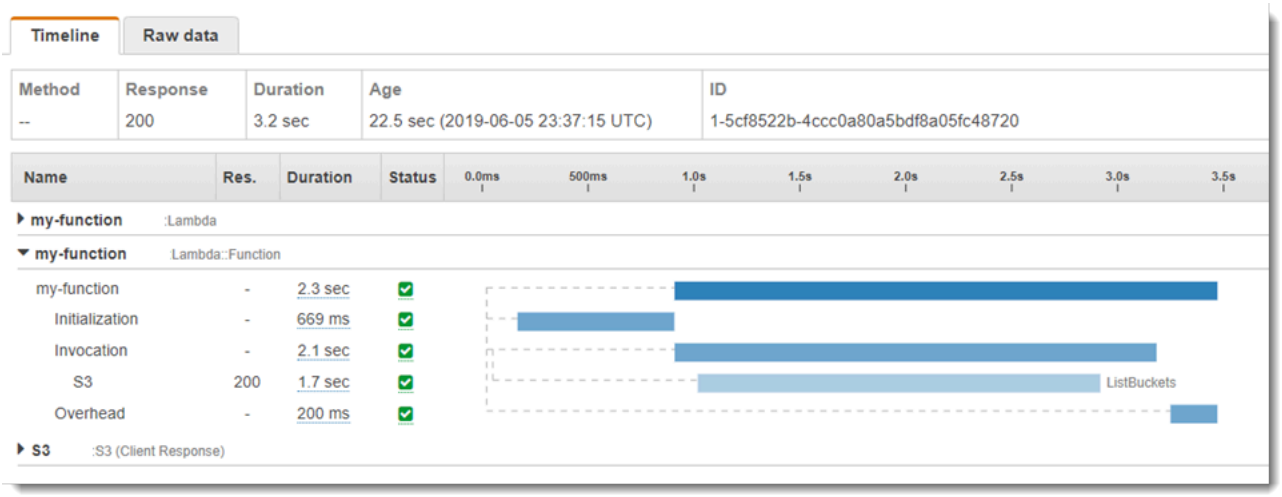
Example blank-ruby/function/lambda_function.rb: rastrear um cliente do AWS SDK

```
# lambda_function.rb
require 'logger'
require 'json'
require 'aws-sdk-lambda'
$client = Aws::Lambda::Client.new()
$client.get_account_settings()

require 'aws-xray-sdk/lambda'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  ...
```

No X-Ray, um rastreamento registra informações sobre uma solicitação que é processada por um ou mais serviços. O Lambda registra dois segmentos por rastreamento, o que cria dois nós no gráfico de serviços. A imagem a seguir destaca esses dois nós:



O primeiro nó à esquerda representa o serviço do Lambda, que recebe a solicitação de invocação. O segundo nó representa a sua função do Lambda específica. O exemplo a seguir mostra um rastreamento com esses dois segmentos. Ambos têm o nome my-function, mas um tem a origem `AWS::Lambda` e o outro, a origem `AWS::Lambda::Function`. Se o segmento `AWS::Lambda` mostrar um erro, o serviço Lambda teve um problema. Se o segmento `AWS::Lambda::Function` mostrar um erro, sua função teve um problema.



Este exemplo expande o segmento `AWS::Lambda::Function` para mostrar seus três subsegmentos:

- Inicialização: representa o tempo gasto carregando a função e executando o código de inicialização. Esse subsegmento aparece somente para o primeiro evento que cada instância da função processa.
- Invocação: representa o tempo gasto na execução do código do manipulador.
- Sobrecarga: representa o tempo gasto pelo runtime do Lambda preparando-se para lidar com o próximo evento.

Você também pode instrumentar clientes HTTP, registrar consultas SQL e criar subsegmentos personalizados com anotações e metadados. Para obter mais informações, consulte The X-Ray SDK for Ruby no Guia do desenvolvedor do AWS X-Ray.

Seções

Habilitar o rastreamento ativo com a API do Lambda

Para gerenciar a configuração de rastreamento com a AWS CLI ou com o AWS SDK, use as seguintes operações de API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration \
--function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Habilitar o rastreamento ativo com o AWS CloudFormation

Para ativar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example function-inline.yml: configuração de rastreamento

```
Resources:
  function:
```

```
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example template.yml: configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Armazenar dependências de runtime em uma camada

Se você usar o X-Ray SDK para instrumentar os clientes do AWS SDK com seu código de função, seu pacote de implantação poderá se tornar bastante grande. Para evitar o upload de dependências de runtime todas as vezes que você atualizar seu código de função, empacote o SDK do X-Ray em uma camada do Lambda.

O exemplo a seguir mostra um recurso `AWS::Serverless::LayerVersion` que armazena o X-Ray SDK for Ruby.

Example template.yml: camada de dependências

```
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
```

Com essa configuração, você atualizará a camada de biblioteca somente se alterar as dependências de runtime. Já que o pacote de implantação de função inclui apenas o seu código, isso pode ajudar a reduzir o tempo de upload.

A criação de uma camada de dependências requer alterações de compilação para gerar o arquivo da camada antes da implantação. Para obter um exemplo funcional, consulte o aplicativo de exemplo blank-ruby.

Construir funções do Lambda com Java

Você pode executar o código Java noAWS Lambda. O Lambda fornece runtimes para Java que executam seu código para processar eventos. O código é executado em um ambiente do Amazon Linux que inclui credenciais da AWS de uma função do AWS Identity and Access Management (IAM) que você gerencia.

O Lambda oferece suporte aos seguintes runtimes Java.

Java

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
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			

O Lambda fornece as seguintes bibliotecas para funções em Java:

- com.amazonaws:aws-lambda-java-core (obrigatória): define as interfaces do método do manipulador e o objeto de contexto que o runtime transmite ao manipulador. Se você definir seus próprios tipos de entrada, esta será a única biblioteca necessária.
- com.amazonaws:aws-lambda-java-events: tipos de entrada para eventos de serviços que invocam funções do Lambda.
- com.amazonaws:aws-lambda-java-log4j2: uma biblioteca appender do Apache Log4j 2 que você pode usar para adicionar o ID de solicitação da invocação atual aos logs de função.
- AWS SDK for Java 2.0: o AWS SDK oficial para a linguagem de programação Java.

> **Important**
>
> Não use componentes privados da API JDK, como campos, métodos ou classes privadas. Componentes de API não públicos podem ser alterados ou removidos em qualquer atualização, causando falha na aplicação.

Para criar uma função em Java

1. Abra o console do lambda.
2. Escolha a opção Criar função.
3. Configure as seguintes opções:
   - Nome da função: digite um nome para a função.
   - Runtime: escolha Java 21.
4. Escolha a opção Criar função.
5. Para configurar um evento de teste, escolha Test (Testar).
6. Em Nome do evento, insira **`test`**.
7. Escolha Salvar alterações.
8. Escolha Test (Testar) para invocar a função.

O console cria uma função do Lambda com uma classe de manipulador chamada `Hello`. Como Java é uma linguagem compilada, não é possível visualizar nem editar o código-fonte no console do Lambda, mas você pode modificar a configuração dele, invocá-lo e configurar acionadores.

> **Note**
>
> Para começar com o desenvolvimento de aplicativos no ambiente local, implante um dos aplicativos de exemplo disponíveis no repositório do GitHub deste guia.

A classe `Hello` tem uma função chamada `handleRequest` que utiliza um objeto de evento e um objeto de contexto. Esta é a função de manipulador que o Lambda chama quando a função é invocada. O runtime da função do Java recebe eventos de invocação do Lambda e os transmite ao handler. Na configuração da função, o valor do manipulador é `example.Hello::handleRequest`.

Para atualizar o código da função, crie um pacote de implantação, que é um arquivo .zip que contém o código da função. À medida que o desenvolvimento da função progredir, você desejará armazenar o código da função no controle de origem, adicionar bibliotecas e automatizar implantações. Comece criando um pacote de implantação e atualizando o seu código na linha de comando.

O runtime transmite um objeto de contexto para o handler, além do evento de invocação. O objeto de contexto contém informações adicionais sobre a invocação, a função e o ambiente de execução. Outras informações estão disponíveis de variáveis de ambiente.

Sua função do Lambda é fornecida com um grupo de logs do CloudWatch Logs. O runtime envia detalhes sobre cada invocação para o CloudWatch Logs. Ele retransmite quaisquer logs que sua função produz durante a invocação. Se a função retornar um erro, o Lambda formatará o erro e o retornará para o chamador.

Tópicos

Definir o manipulador da função do Lambda em Java

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. A função é executada até que o manipulador retorne uma resposta, seja encerrado ou atinja o tempo limite.

O repositório do GitHub para este guia fornece aplicativos de exemplo fáceis de implantar que demonstram uma variedade de tipos de manipuladores. Para obter detalhes, consulte o final deste tópico.

Seções

Exemplo de manipulador: runtimes do Java 17

No exemplo do Java 17 a seguir, uma classe chamada `HandlerIntegerJava17` define um método do manipulador chamado `handleRequest`. O método do manipulador recebe as seguintes entradas:

- Um `IntegerRecord`, que é um registro customizado do Java que representa os dados do evento. Neste exemplo, definimos `IntegerRecord` da seguinte forma:

```
record IntegerRecord(int x, int y, String message) {
}
```

- Um objeto de contexto, que disponibiliza métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de execução.

Suponha que desejemos escrever uma função que registre em log a `message` da entrada `IntegerRecord` e retorne a soma de `x` e `y`. Este é o código da função:

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) {
}
```

Você especifica qual método deseja que o Lambda invoque ao definir o parâmetro do manipulador na configuração da função. É possível expressar o manipulador nos seguintes formatos:

- *package*.*Class*::*method*: formato completo. Por exemplo: `example.Handler::handleRequest`.
- *package*.*Class*: formato abreviado para classes que implantam uma interface de manipulador. Por exemplo: `example.Handler`.

Quando o Lambda invoca o manipulador, o runtime do Lambda recebe um evento como uma string formatada em JSON e o converte em um objeto. Para o exemplo anterior, uma amostra de evento poderá se assemelhar ao seguinte:

Example event.json

```
{
  "x": 1,
  "y": 20,
  "message": "Hello World!"
}
```

É possível salvar esse arquivo e testar a função localmente com o seguinte comando da AWS Command Line Interface (CLI):

```
aws lambda invoke --function-name function_name --payload file://event.json out.json
```

Exemplo de manipulador: runtimes do Java 11 e versões anteriores

O Lambda é compatível com registros para runtimes do Java 17 e versões posteriores. Em todos os runtimes do Java, é possível usar uma classe para representar os dados do evento. O exemplo a seguir usa uma lista de inteiros e um objeto de contexto como entrada e retorna a soma de todos os inteiros na lista.

Example Handler.java

No exemplo a seguir, uma classe chamada `Handler` define um método do manipulador chamado `handleRequest`. O método do manipulador usa um evento e um objeto de contexto como entrada e retorna uma string.

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();
  }
}
```

Para obter mais exemplos, consulte Código de exemplo do manipulador.

Código de inicialização

O Lambda executa seu código estático e o construtor de classe durante a fase de inicialização antes de invocar a função pela primeira vez. Os recursos criados durante a inicialização permanecem na memória entre as invocações e podem ser reutilizados pelo manipulador milhares de vezes. Dessa forma, é possível adicionar código de inicialização de forma externa ao método do manipulador principal para economizar tempo de computação e reutilizar recursos em diversas invocações.

No exemplo a seguir, o código de inicialização do cliente é externo ao método do manipulador principal. O runtime inicializa o cliente antes que a função veicule o primeiro evento. Os eventos subsequentes são muito mais rápidos porque o Lambda não precisa inicializar o cliente novamente.

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";
    }
}
```

Escolher tipos de entrada e saída

Especifique o tipo de objeto para o qual o evento é mapeado na assinatura do método do manipulador. No exemplo anterior, o runtime do Java desserializa o evento em um tipo que implementa a interface `Map<String,String>`. Mapas de string a string funcionam para eventos simples como o seguinte:

Example Event.json: dados climáticos

```
{
  "temperatureK": 281,
  "windKmh": -3,
  "humidityPct": 0.55,
  "pressureHPa": 1020
}
```

No entanto, o valor de cada campo deve ser uma string ou um número. Se o evento incluir um campo que tenha um objeto como um valor, o runtime não poderá desserializá-lo e retornará um erro.

Escolha um tipo de entrada que funcione com os dados de evento que sua função processa. É possível usar um tipo básico, um tipo genérico ou um tipo bem definido.

Tipos de entrada

- `Integer`, `Long`, `Double` etc. – o evento é um número sem formatação adicional, por exemplo, `3.5`. O runtime converte o valor em um objeto do tipo especificado.
- `String`: o evento é uma string JSON, incluindo aspas, por exemplo, `"My string."`. O runtime converte o valor (sem aspas) em um objeto `String`.
- *`Type`*, `Map<String,`*`Type`*`>` etc. – o evento é um objeto JSON. O runtime o desserializa em um objeto da interface ou tipo especificado.
- `List<Integer>`, `List<String>`, `List<Object>` etc. – o evento é uma matriz JSON. O runtime o desserializa em um objeto da interface ou tipo especificado.
- `InputStream`: o evento é qualquer tipo JSON. O runtime transmite um fluxo de bytes do documento ao manipulador sem modificação. Desserialize a entrada e grave a saída em um fluxo de saída.
- Tipo de biblioteca: para eventos enviados pelos serviços da AWS, use os tipos na biblioteca aws-lambda-java-events.

Se você definir seu próprio tipo de entrada, ele deve ser um objeto Java simples e antigo (POJO) desserializável e mutável, com um construtor padrão e propriedades para cada campo no evento. Chaves no evento que não mapeiem para uma propriedade e propriedades que não estejam incluídas no evento são descartadas sem erro.

O tipo de saída pode ser um objeto ou `void`. O runtime serializa valores de retorno em texto. Se a saída for um objeto com campos, o runtime serializa-o em um documento JSON. Se for um tipo que envolve um valor primitivo, o runtime retornará uma representação de texto desse valor.

Interfaces do manipulador

A biblioteca aws-lambda-java-core define duas interfaces para métodos do manipulador. Use as interfaces fornecidas para simplificar a configuração do manipulador e validar a assinatura do método do manipulador no momento da compilação.

- com.amazonaws.services.lambda.runtime.requestHandler
- com.amazonaws.services.lambda.runtime.requestStreamHandler

A interface `RequestHandler` é um tipo genérico que usa dois parâmetros: o tipo de entrada e o tipo de saída. Ambos os tipos devem ser objetos. Quando você usa essa interface, o runtime do Java

desserializa o evento em um objeto com o tipo de entrada e serializa a saída em texto. Use essa interface quando a serialização interna funcionar com seus tipos de entrada e saída.

Example Handler.java: interface do manipulador

```
// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String,String>, String>{
  @Override
  public String handleRequest(Map<String,String> event, Context context)
```

Para usar sua própria serialização, implemente a interface `RequestStreamHandler`. Com essa interface, o Lambda transmite ao seu manipulador uma transmissão de entrada e uma de saída. O manipulador lê bytes do fluxo de entrada, grava no fluxo de saída e retorna um void.

O exemplo a seguir usa tipos de leitor e gravador em buffer para trabalhar com os fluxos de entrada e saída.

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();
    }
  }
}
```

Código de exemplo do manipulador

O repositório do GitHub para este guia inclui aplicativos de exemplo que demonstram o uso de vários tipos e interfaces de manipuladores. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS SAM e recursos de suporte.

Aplicações de exemplo do Lambda em Java

- java17-examples: uma função em Java que demonstra como usar um registro Java para representar um objeto de dados de evento de entrada.
- java-basic: uma coleção de funções Java mínimas com testes de unidade e configuração de registro em log variável.
- java-events: uma coleção de funções do Java contendo código básico sobre como lidar com eventos de vários serviços, como o Amazon API Gateway, o Amazon SQS e o Amazon Kinesis. Essas funções usam a versão mais recente da biblioteca aws-lambda-java-events (3.0.0 e versões mais recentes). Estes exemplos não exigem oAWS SDK como dependência.
- s3-java: uma função em Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.
- Use API Gateway to invoke a Lambda function (Usar o API Gateway para invocar uma função do Lambda): uma função Java que verifica uma tabela do Amazon DynamoDB contendo informações de funcionários. Em seguida, usa o Amazon Simple Notification Service para enviar uma mensagem de texto aos funcionários comemorando seus aniversários de empresa. Este exemplo usa o API Gateway para invocar a função.

As aplicações `s3-java` e `java-events` usam um evento de serviço da AWS como entrada e retornam uma string. O aplicativo `java-basic` inclui vários tipos de manipuladores:

- Handler.java: recebe `Map<String,String>` como entrada.
- HandlerInteger.java: recebe `Integer` como entrada.
- HandlerList.java: recebe `List<Integer>` como entrada.
- HandlerStream.java: recebe `InputStream` e `OutputStream` como entrada.
- HandlerString.java: recebe uma `String` como entrada.
- HandlerWeatherData.java: recebe um tipo personalizado como entrada.

Para testar diferentes tipos de manipuladores, basta alterar o valor do manipulador no modelo do AWS SAM. Para obter instruções detalhadas, consulte o arquivo readme do aplicativo de exemplo.

Implantar funções do Lambda em Java com arquivos .zip ou JAR

O código da função do AWS Lambda consiste em scripts ou programas compilados e as dependências deles. Você usa umpacote de implantaçãopara implantar seu código de função no Lambda. O Lambda é compatível com dois tipos de pacotes de implantação: imagens de contêiner e arquivos .zip.

Esta página descreve como criar o seu pacote de implantação como um arquivo .zip ou Jar e, em seguida, usar o pacote para implantar o seu código de função para o AWS Lambda usando a AWS Command Line Interface (AWS CLI).

Seções

Pré-requisitos

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- AWS CLI – Configuração rápida com `aws configure`

Ferramentas e bibliotecas

O Lambda fornece as seguintes bibliotecas para funções em Java:

- com.amazonaws:aws-lambda-java-core (obrigatória): define as interfaces do método do manipulador e o objeto de contexto que o runtime transmite ao manipulador. Se você definir seus próprios tipos de entrada, esta será a única biblioteca necessária.
- com.amazonaws:aws-lambda-java-events: tipos de entrada para eventos de serviços que invocam funções do Lambda.
- com.amazonaws:aws-lambda-java-log4j2: uma biblioteca appender do Apache Log4j 2 que você pode usar para adicionar o ID de solicitação da invocação atual aos logs de função.
- AWS SDK for Java 2.0: o AWSSDK oficial para a linguagem de programação Java.

Essas bibliotecas estão disponíveis no repositório central do Maven. Adicione-as à sua definição de compilação da seguinte forma:

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>
```

Para criar um pacote de implantação, compile o código e as dependências da sua função em um único arquivo .zip ou Java Archive (JAR). Para o Gradle, use o tipo de compilação `Zip`. Para o Apache Maven, use o plugin Maven Shade. Para fazer o upload do seu pacote de implantação, use o console do Lambda, a API do Lambda ou o AWS Serverless Application Model (AWS SAM).

> **Note**
>
> Para manter o pacote de implantação pequeno, empacote as dependências da função em camadas. As camadas permitem gerenciar as suas dependências de forma independente, podem ser usadas por várias funções e podem ser compartilhadas com outras contas. Para ter mais informações, consulte Camadas do Lambda.

Compilar um pacote de implantação com o Gradle

Para criar um pacote de implantação com o código e as dependências da sua função no Gradle, use o tipo de compilação `Zip`. Aqui está um exemplo de um arquivo build.gradle completo de amostra:

Example build.gradle: tarefa de compilação

```
task buildZip(type: Zip) {
    into('lib') {
        from(jar)
        from(configurations.runtimeClasspath)
    }
}
```

Essa configuração de compilação produz um pacote de implantação no diretório `build/distributions`. Na instrução `into('lib')`, a tarefa `jar` monta um arquivo jar contendo suas classes principais em uma pasta denominada `lib`. Além disso, a tarefa `configurations.runtimeClassPath` copia bibliotecas de dependência do caminho de classe da compilação para a mesma pasta `lib`.

Example build.gradle: dependências

```
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'
    ...
}
```

O Lambda carrega arquivos JAR em Unicode em ordem alfabética. Se vários arquivos JAR no diretório `lib` contiverem a mesma classe, a primeira será usada. Use o script de shell a seguir para identificar classes duplicadas.

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
```

Criar uma camada Java para suas dependências

> **Note**
>
> Usar camadas com funções em uma linguagem compilada, como Java, pode não oferecer o mesmo benefício que com uma linguagem interpretada, como Python. Como Java é uma linguagem compilada, suas funções ainda precisam carregar manualmente quaisquer montagens compartilhadas na memória durante a fase inicial, o que pode aumentar os tempos de inicialização a frio. Em vez disso, recomendamos incluir qualquer código compartilhado no momento da compilação para aproveitar as otimizações integradas do compilador.

As instruções nesta seção mostram como incluir suas dependências em uma camada. Para obter instruções sobre como incluir suas dependências em seu pacote de implantação, consulte the section called “Compilar um pacote de implantação com o Gradle” ou the section called “Compilar um pacote de implantação com o Maven”.

Quando você adiciona uma camada a uma função, o Lambda carrega o conteúdo da camada no diretório `/opt` desse ambiente de execução. Para cada runtime do Lambda, a variável `PATH` já inclui caminhos de pasta específica no diretório `/opt`. Para garantir que a variável `PATH` colete o conteúdo da camada, o arquivo .zip da camada deve ter suas dependências nos seguintes caminhos de pasta:

- `java/lib` (CLASSPATH)

Por exemplo, sua estrutura de arquivo .zip da sua camada pode ser assim:

```
jackson.zip
# java/lib/jackson-core-2.2.3.jar
```

Além disso, o Lambda detecta automaticamente todas as bibliotecas no diretório `/opt/lib` e quaisquer binários no diretório `/opt/bin`. Para garantir que o Lambda encontre corretamente o conteúdo da sua camada, você também pode criar uma camada com a seguinte estrutura:

```
custom-layer.zip
# lib
    | lib_1
    | lib_2
# bin
    | bin_1
    | bin_2
```

Depois de empacotar sua camada, consulte the section called “Criar e excluir camadas” e the section called “Adicionar camadas” para concluir sua configuração de camada.

Compilar um pacote de implantação com o Maven

Para compilar um pacote de implantação com o Maven, use o plugin Maven Shade. O plugin cria um arquivo JAR que contém o código de função compilado e todas as suas dependências.

Example pom.xml: configuração do 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>
```

Para compilar o pacote de implantação, use o 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] ------------------------------------------------------------------------
```

Esse comando gera um arquivo JAR no diretório `target`.

> **Note**
>
> Se você estiver trabalhando com um JAR de várias versões (MRJAR), é necessário incluir o MRJAR (ou seja, o JAR sombreado produzido pelo plugin Maven Shade) no diretório `lib` e compactá-lo antes de fazer upload do seu pacote de implantação para o Lambda. Caso contrário, o Lambda pode não descompactar adequadamente seu arquivo JAR, fazendo com que seu arquivo `MANIFEST.MF` seja ignorado.

Se você usar a biblioteca appender (`aws-lambda-java-log4j2`), também será necessário configurar um transformador para o plugin Maven Shade. A biblioteca do transformador combina versões de um arquivo de cache que aparecem na biblioteca appender e no Log4j.

Example pom.xml: configuração do plugin com o appender do 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>
```

Upload de um pacote de implantação com o console do Lambda

Para criar uma nova função, você deve primeiro criar a função no console e depois carregar o arquivo .zip ou JAR. Para atualizar uma função existente, abra a página da função e siga o mesmo procedimento para adicionar o arquivo .zip ou JAR atualizado.

Se o arquivo do pacote de implantação for menor que 50 MB, você poderá criar ou atualizar uma função carregando o arquivo diretamente da máquina local. Para arquivos .zip ou JAR maiores que 50 MB, você deve primeiro carregar o pacote para um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando o AWS Management Console, consulte Conceitos básicos do Amazon S3. Para carregar arquivos usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

> **Note**
>
> Você não pode alterar o tipo de pacote de implantação (.zip ou imagem de contêiner) de uma função existente. Por exemplo, você não pode converter uma função de imagem de contêiner para usar um arquivo compactado .zip. É necessário criar uma nova função.

Para criar uma função (console)

1. Abra a página Funções do console do Lambda e escolha Criar função.
2. Escolha Author from scratch (Criar do zero).
3. Em Basic information (Informações básicas), faça o seguinte:
   a. Em Nome da função, insira o nome da função.
   b. Em Runtime, selecione o runtime que você deseja usar.
   c. (Opcional) Em Arquitetura, escolha a arquitetura do conjunto de instruções para a função. O valor da arquitetura padrão é X86_64. Certifique-se de que o pacote de implantação .zip da função seja compatível com a arquitetura do conjunto de instruções que você escolheu.
4. (Opcional) Em Permissões, expanda Alterar função de execução padrão. Crie uma função de execução ou use uma existente.
5. Escolha a opção Criar função. O Lambda cria uma função básica “Hello world” usando o runtime escolhido.

Para carregar o arquivo .zip ou JAR da máquina local (console)

1. Na página Funções do console Lambda, escolha a função para a qual você deseja carregar o arquivo .zip ou JAR.
2. Selecione a guia Código.
3. No painel do Código-fonte, escolha Carregar de.

4. Escolha o arquivo .zip ou .jar.
5. Para carregar o arquivo .zip ou JAR, faça o seguinte:

   a. Selecione Carregar e, em seguida, selecione o arquivo .zip ou JAR no seletor de arquivos.
   b. Escolha Open (Abrir).
   c. Escolha Salvar.

Para carregar um arquivo .zip ou JAR de um bucket do Amazon S3 (console)

1. Na página Funções do console do Lambda, escolha a função para a qual você deseja carregar um novo arquivo .zip ou JAR.
2. Selecione a guia Código.
3. No painel do Código-fonte, escolha Carregar de.
4. Escolha Local do Amazon S3.
5. Cole o URL do link do Amazon S3 do arquivo .zip e escolha Salvar.

Carregar um pacote de implantação com a AWS CLI

Você pode usar a AWS CLI para criar uma função ou atualizar uma existente usando um arquivo .zip ou JAR. Use os comandos create-function e update-function-code para implantar o pacote .zip ou JAR. Se o arquivo for menor que 50 MB, você poderá carregar o pacote de um local do arquivo na máquina de compilação local. Para arquivos maiores, você deve carregar o pacote .zip ou JAR de um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

> **Note**
>
> Se você carregar o arquivo .zip ou JAR de um bucket do Amazon S3 usando a AWS CLI, o bucket deverá estar na mesma Região da AWS que sua função.

Para criar uma função usando um arquivo .zip ou JAR com a AWS CLI, você deve especificar o seguinte:

- O nome da função (`--function-name`)
- O runtime da função (`--runtime`)

- O nome do recurso da Amazon (ARN) da função de execução da função (`--role`)
- O nome do método do manipulador no código da função (`--handler`)

Você também deve especificar a local do arquivo .zip ou JAR. Se o arquivo .zip ou JAR estiver localizado em uma pasta da máquina de compilação local, use a opção `--zip-file` para especificar o caminho do arquivo, conforme mostrado no comando do exemplo a seguir.

```
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
```

Para especificar o local do arquivo .zip em um bucket do Amazon S3, use a opção `--code` conforme mostrado no comando do exemplo a seguir. Você só precisa usar o parâmetro `S3ObjectVersion` para objetos com versionamento.

```
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
```

Para atualizar uma função existente usando a CLI, especifique o nome da função usando o parâmetro `--function-name`. Você também deve especificar o local do arquivo .zip que deseja usar para atualizar o código da função. Se o arquivo .zip estiver localizado em uma pasta da máquina de compilação local, use a opção `--zip-file` para especificar o caminho do arquivo, conforme mostrado no comando do exemplo a seguir.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Para especificar o local do arquivo .zip em um bucket do Amazon S3, use as opções `--s3-bucket` e `--s3-key` conforme mostrado no comando do exemplo a seguir. Você só precisa usar o parâmetro `--s3-object-version` para objetos com versionamento.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket DOC-EXAMPLE-BUCKET --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Fazer upload de um pacote de implantação com o AWS SAM

É possível usar o AWS SAM para automatizar implantações do código, da configuração e das dependências da sua função. O AWS SAM é uma extensão do AWS CloudFormation que fornece uma sintaxe simplificada para definir aplicações sem servidor. O seguinte exemplo de modelo define uma função com um pacote de implantação no diretório `build/distributions` que o Gradle usa:

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
```

Para criar a função, use os comandos `deploy` e `package`. Esses comandos são personalizações para a AWS CLI. Eles envolvem outros comandos para fazer upload do pacote de implantação no Amazon S3, reescrevem o modelo com o URI do objeto e atualizam o código da função.

O exemplo de script a seguir executa uma compilação do Gradle e faz upload do pacote de implantação que ele cria. Ele cria uma pilha do AWS CloudFormation na primeira vez que você executá-lo. Se a pilha já existir, o script a atualizará.

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
```

Para obter um modelo funcional completo, consulte as seguintes aplicações de exemplo:

Aplicações de exemplo do Lambda em Java

- java17-examples: uma função em Java que demonstra como usar um registro Java para representar um objeto de dados de evento de entrada.
- java-basic: uma coleção de funções Java mínimas com testes de unidade e configuração de registro em log variável.
- java-events: uma coleção de funções do Java contendo código básico sobre como lidar com eventos de vários serviços, como o Amazon API Gateway, o Amazon SQS e o Amazon Kinesis. Essas funções usam a versão mais recente da biblioteca aws-lambda-java-events (3.0.0 e versões mais recentes). Estes exemplos não exigem oAWS SDK como dependência.
- s3-java: uma função em Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.
- Use API Gateway to invoke a Lambda function (Usar o API Gateway para invocar uma função do Lambda): uma função Java que verifica uma tabela do Amazon DynamoDB contendo informações de funcionários. Em seguida, usa o Amazon Simple Notification Service para enviar uma mensagem de texto aos funcionários comemorando seus aniversários de empresa. Este exemplo usa o API Gateway para invocar a função.

Implantar funções do Lambda em Java com imagens de contêiner

Existem três maneiras de criar uma imagem de contêiner para uma função do Lambda em Java:

- Usar uma imagem base da AWS para Java

  As imagens base da AWS são pré-carregadas com um runtime de linguagem, um cliente de interface de runtime para gerenciar a interação entre o Lambda e o código da sua função e um emulador de interface de runtime para testes locais.
- Usar uma imagem base somente para sistema operacional da AWS

  As imagens base somente para sistema operacional da AWS contêm uma distribuição do Amazon Linux e o emulador de interface de runtime. Essas imagens são comumente usadas para criar imagens de contêiner para linguagens compiladas, como Go e Rust e para uma linguagem ou versão de linguagem para a qual o Lambda não fornece uma imagem base, como Node.js 19. Você também pode usar imagens base somente para sistema operacional para implementar um runtime personalizado. Para tornar a imagem compatível com o Lambda, você deve incluir o cliente de interface de runtime do Java na imagem.
- Usar uma imagem base que não é da AWS

  Você também pode usar uma imagem base alternativa de outro registro de contêiner, como Alpine Linux ou Debian. Você também pode usar uma imagem personalizada criada por sua organização. Para tornar a imagem compatível com o Lambda, você deve incluir o cliente de interface de runtime do Java na imagem.

> **Tip**
>
> Para reduzir o tempo necessário para que as funções do contêiner do Lambda se tornem ativas, consulte Use multi-stage builds na documentação do Docker. Para criar imagens de contêiner eficientes, siga as Melhores práticas para gravar Dockerfiles.

Esta página explica como criar, testar e implantar imagens de contêiner para o Lambda.

Tópicos

- Usar uma imagem base alternativa com o cliente da interface de runtime

Imagens base da AWS para Java

A AWS oferece as seguintes imagens base para Java:

Tags	Runtime	Sistema operacional	Dockerfile	Desaprovação
21	Java 21	Amazon Linux 2023	Dockerfile para Java 21 no GitHub	
17	Java 17	Amazon Linux 2	Dockerfile para Java 17 no GitHub	
11	Java 11	Amazon Linux 2	Dockerfile para Java 11 no GitHub	
8.al2	Java 8	Amazon Linux 2	Dockerfile para Java 8 no GitHub	

Repositório do Amazon ECR: gallery.ecr.aws/lambda/java

As imagens base do Java 21 e posteriores são baseadas na imagem de contêiner mínimo do Amazon Linux 2023. Imagens base anteriores usam o Amazon Linux 2. O AL2023 oferece várias vantagens em relação ao Amazon Linux 2, incluindo uma área de implantação menor e versões atualizadas de bibliotecas, como `glibc`.

As imagens baseadas no AL2023 usam o `microdnf` (com link simbólico `dnf`) como o gerenciador de pacotes, em vez do `yum`, que é o gerenciador de pacotes padrão no Amazon Linux 2. O `microdnf` é uma implementação autônoma do `dnf`. Para obter uma lista dos pacotes incluídos nas imagens baseadas no AL2023, consulte as colunas Contêiner mínimo em Comparar pacotes instalados em imagens de contêiner do Amazon Linux 2023. Para obter mais informações sobre as diferenças entre o AL2023 e o Amazon Linux 2, consulte Introdução ao runtime do Amazon Linux 2023 para AWS Lambda no blog AWS Compute.

> **Note**
>
> Para executar imagens baseadas no AL2023 localmente, inclusive com o AWS Serverless Application Model (AWS SAM), você deve usar o Docker versão 20.10.10 ou posterior.

Usar uma imagem base da AWS para Java

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Java (por exemplo, Amazon Corretto)
- Docker (versão mínima 20.10.10 para Java 21 e imagens base posteriores)
- Apache Maven ou Gradle
- AWS Command Line Interface (AWS CLI) versão 2

Criação de uma imagem a partir de uma imagem base

Maven

1. Execute o comando a seguir para criar um projeto de Maven usando o arquétipo para Lambda. Os seguintes parâmetros são obrigatórios:

   - serviço: o cliente AWS service (Serviço da AWS) a ser usado na função do Lambda. Para obter uma lista das origens disponíveis, consulte aws-sdk-java-v2/services no GitHub.
   - região: a Região da AWS na qual você deseja criar a função do Lambda.
   - groupId: o namespace completo do pacote da sua aplicação.
   - artifactId: o nome do seu projeto. Esse será o nome do diretório do projeto.

   No Linux e no macOS, execute este comando:

   ```
   mvn -B archetype:generate \
      -DarchetypeGroupId=software.amazon.awssdk \
      -DarchetypeArtifactId=archetype-lambda -Dservice=s3 -Dregion=US_WEST_2 \
      -DgroupId=com.example.myapp \
      -DartifactId=myapp
   ```

No PowerShell, execute este comando:

```
mvn -B archetype:generate `
   "-DarchetypeGroupId=software.amazon.awssdk" `
   "-DarchetypeArtifactId=archetype-lambda" "-Dservice=s3" "-Dregion=US_WEST_2"
 `
   "-DgroupId=com.example.myapp" `
   "-DartifactId=myapp"
```

O arquétipo do Maven para Lambda é pré-configurado para compilar com o Java SE 8 e inclui uma dependência para o AWS SDK for Java. Se você criar seu projeto com um arquétipo diferente ou usando outro método, deverá configurar o compilador Java para Maven e declarar o SDK como uma dependência.

2. Abra o diretório *myapp*`/src/main/java/com/example/`*myapp* e localize o arquivo `App.java`. Este é o código da função do Lambda. É possível usar o código de exemplo fornecido para testes ou substituí-lo pelo seu.
3. Navegue de volta para o diretório raiz do projeto e, em seguida, crie um Dockerfile com a seguinte configuração:

   - Defina a propriedade `FROM` como o URI da imagem base.
   - Defina o argumento `CMD` como o manipulador de funções do 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. Compile o projeto e colete as dependências do runtime.

   ```
   mvn compile dependency:copy-dependencies -DincludeScope=runtime
   ```

5. Crie a imagem do Docker com o comando docker build. O exemplo a seguir nomeia a imagem como `docker-image` e atribui a ela a tag `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> O comando especifica a opção `--platform linux/amd64` para garantir que seu contêiner seja compatível com o ambiente de execução do Lambda, independentemente da arquitetura da sua máquina de compilação. Se você pretende criar uma função do Lambda usando a arquitetura do conjunto de instruções ARM64, certifique-se de alterar o comando para usar a opção `--platform linux/arm64` em vez disso.

Gradle

1. Crie um diretório para o projeto e depois mude para esse diretório.

```
mkdir example
cd example
```

2. Execute o comando a seguir para que o Gradle gere um novo projeto de aplicação Java no diretório `example` em seu ambiente. Em Selecionar DSL de script de compilação, escolha 2: Groovy.

```
gradle init --type java-application
```

3. Abra o diretório */example*`/app/src/main/java/`*example* e localize o arquivo `App.java`. Este é o código da função do Lambda. É possível usar o código de exemplo a seguir para fins de teste ou substituí-lo pelo seu próprio código.

   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. Abra o arquivo `build.gradle`. Se você estiver usando o exemplo de código de função da etapa anterior, substitua o conteúdo de `build.gradle` pelo conteúdo a seguir. Se você estiver usando seu próprio código de função, modifique seu arquivo `build.gradle` conforme necessário.

   Example build.gradle (DSL Groovy)

   ```
   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. O comando `gradle init` da etapa 2 também gerou um caso de teste fictício no diretório `app/test`. Para os fins deste tutorial, pule a execução de testes excluindo o diretório `/test`.

6. Crie o projeto.

   ```
   gradle build
   ```

7. No diretório raiz do projeto (`/example`), crie um Dockerfile com a seguinte configuração:

   - Defina a propriedade `FROM` como o URI da imagem base.
   - Use o comando COPY para copiar o código da função e as dependências do runtime para `{LAMBDA_TASK_ROOT}`, uma variável de ambiente definida pelo Lambda.

- Defina o argumento CMD como o manipulador de funções do 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. Crie a imagem do Docker com o comando docker build. O exemplo a seguir nomeia a imagem como `docker-image` e atribui a ela a tag `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> O comando especifica a opção `--platform linux/amd64` para garantir que seu contêiner seja compatível com o ambiente de execução do Lambda, independentemente da arquitetura da sua máquina de compilação. Se você pretende criar uma função do Lambda usando a arquitetura do conjunto de instruções ARM64, certifique-se de alterar o comando para usar a opção `--platform linux/arm64` em vez disso.

(Opcional) Teste a imagem localmente

1. Inicie a imagem do Docker com o comando docker run. Neste exemplo, `docker-image` é o nome da imagem e `test` é a tag.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Esse comando executa a imagem como um contêiner e cria um endpoint local em `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se você criou a imagem do Docker para a arquitetura do conjunto de instruções ARM64, certifique-se de usar a opção `--platform linux/arm64`, em vez de `--platform linux/amd64`.

2. Em uma nova janela de terminal, publique um evento no endpoint local.

   Linux/macOS

   No Linux e no MacOS, execute o seguinte comando `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   No PowerShell, execute o seguinte comando `Invoke-WebRequest`:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
    "application/json"
   ```

3. Obtenha o ID do contêiner.

```
docker ps
```

4. Use o comando docker kill para parar o contêiner. Nesse comando, substitua `3766c4ab331c` pelo ID do contêiner da etapa anterior.

```
docker kill 3766c4ab331c
```

Implantação da imagem

Para enviar a imagem ao Amazon ECR e criar a função do Lambda

1. Execute o comando get-login-password para autenticar a CLI do Docker no seu registro do Amazon ECR.

   - Defina o valor `--region` para a Região da AWS onde você deseja criar o repositório do Amazon ECR.
   - Substituir `111122223333` por seu ID da Conta da 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. Crie um repositório no Amazon ECR usando o 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**
>
> O repositório do Amazon ECR deve estar na mesma Região da AWS que a função do Lambda.

Se tiver êxito, você verá uma resposta como esta:

```
{
    "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. Copie o `repositoryUri` da saída na etapa anterior.
4. Execute o comando docker tag para aplicar uma tag na sua imagem local em seu repositório do Amazon ECR como a versão mais recente. Neste comando:

   - Substitua `docker-image:test` pelo nome e tag da sua imagem do Docker.
   - Substitua `<ECRrepositoryUri>` pelo `repositoryUri` que você copiou. Certifique-se de incluir `:latest` no final do URI.

   ```
   docker tag docker-image:test <ECRrepositoryUri>:latest
   ```

   Exemplo:

   ```
   docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
   world:latest
   ```

5. Execute o comando docker push para implantar a imagem local no repositório do Amazon ECR. Certifique-se de incluir `:latest` no final do URI do repositório.

   ```
   docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
   ```

6. Crie um perfil de execução para a função, caso ainda não tenha um. Você precisará do nome do recurso da Amazon (ARN) do perfil na próxima etapa.

7. Criar a função do Lambda. Em `ImageUri`, especifique o URI do repositório anterior. Certifique-se de incluir `:latest` no final do 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**
>
> É possível criar uma função usando uma imagem em uma conta da AWS diferente desde que a imagem esteja na mesma região da função do Lambda. Para ter mais informações, consulte Permissões entre contas do Amazon ECR.

8. Invoque a função.

```
aws lambda invoke --function-name hello-world response.json
```

Você obterá uma resposta parecida com esta:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Para ver a saída da função, verifique o arquivo `response.json`.

Para atualizar o código da função, você deve criar a imagem novamente, fazer upload da nova imagem no repositório do Amazon ECR e, em seguida, usar o comando update-function-code para implantar a imagem na função do Lambda.

O Lambda resolve a tag de imagem em um resumo de imagem específico. Isso significa que, se você apontar a tag de imagem que foi usada para implantar a função em uma nova imagem no Amazon ECR, o Lambda não atualizará automaticamente a função para usar a nova imagem. Para implantar a nova imagem na mesma função do Lambda, você deverá usar o comando `update-function-code`, mesmo que a tag da imagem no Amazon ECR permaneça a mesma.

Usar uma imagem base alternativa com o cliente da interface de runtime

Se você usar uma imagem base somente para sistema operacional ou uma imagem base alternativa, deverá incluir o cliente de interface de runtime na imagem. O cliente de interface de runtime estende API de tempo de execução do Lambda, que gerencia a interação entre o Lambda e o código da sua função.

Instale o cliente de interface de runtime para Java no Dockerfile ou como uma dependência no projeto. Por exemplo, para instalar o cliente de interface de runtime usando o gerenciador de pacotes Maven, adicione o seguinte ao seu arquivo `pom.xml`:

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-runtime-interface-client</artifactId>
    <version>2.3.2</version>
</dependency>
```

Para obter detalhes do pacote, consulte Cliente de interface de runtime do AWS Lambda em Java no Repositório central do Maven. Você também pode revisar o código-fonte do cliente de interface de runtime no repositório de Bibliotecas de suporte do Java do AWS Lambda no GitHub.

O exemplo a seguir demonstra como criar uma imagem de contêiner para Java usando uma imagem do Amazon Corretto. O Amazon Corretto é uma distribuição gratuita, multiplataforma e pronta para produção do Open Java Development Kit (OpenJDK). O projeto do Maven inclui o cliente de interface de runtime como uma dependência.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Java (por exemplo, Amazon Corretto)
- Docker
- Apache Maven
- AWS Command Line Interface (AWS CLI) versão 2

Criar uma imagem de uma imagem base alternativa

1. Criar um projeto do Maven. Os seguintes parâmetros são obrigatórios:

- groupId: o namespace completo do pacote da sua aplicação.
- artifactId: o nome do seu projeto. Esse será o nome do diretório do projeto.

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. Abra o diretório do projeto.

```
cd myapp
```

3. Abra o arquivo `pom.xml` e substitua o conteúdo pelo seguinte. Esse arquivo inclui o aws-lambda-java-runtime-interface-client como uma dependência. Como alternativa, você também pode instalar o cliente de interface de runtime no Dockerfile. No entanto, a abordagem mais simples é incluir a biblioteca como uma dependência.

```
<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. Abra o diretório *myapp*`/src/main/java/com/example/`*myapp* e localize o arquivo `App.java`. Este é o código da função do Lambda. Substitua o código pelo seguinte.

   Example manipulador de função

```
package example;

public class App {
    public static String sayHello() {
        return "Hello world!";
    }
}
```

5. O comando `mvn -B archetype:generate` da etapa 1 também gerou um caso de teste fictício no diretório `src/test`. Para fins deste tutorial, pule a execução de testes excluindo todo este diretório `/test` gerado.
6. Navegue de volta para o diretório raiz do projeto e, em seguida, crie um Dockerfile. O Dockerfile do exemplo a seguir usa uma imagem do Amazon Corretto. O Amazon Corretto é uma distribuição gratuita, multiplataforma e pronta para produção do OpenJDK.

   - Defina a propriedade `FROM` como o URI da imagem base.
   - Defina o `ENTRYPOINT` como o módulo em que você deseja que o contêiner do Docker seja executado quando for iniciado. Nesse caso, o módulo é o cliente de interface de runtime.
   - Defina o argumento `CMD` como o manipulador de funções do 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. Crie a imagem do Docker com o comando docker build. O exemplo a seguir nomeia a imagem como `docker-image` e atribui a ela a tag `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> O comando especifica a opção `--platform linux/amd64` para garantir que seu contêiner seja compatível com o ambiente de execução do Lambda, independentemente da arquitetura da sua máquina de compilação. Se você pretende criar uma função do Lambda usando a arquitetura do conjunto de instruções ARM64, certifique-se de alterar o comando para usar a opção `--platform linux/arm64` em vez disso.

(Opcional) Teste a imagem localmente

Use o emulador de interface de runtime para testar a imagem localmente. Você pode compilar o emulador em sua imagem ou usar o procedimento a seguir instalá-lo na sua máquina local.

Para instalar o emulador de interface de runtime na sua máquina local

1. No diretório do projeto, execute o comando a seguir para baixar o emulador de interface de runtime (arquitetura x86-64) do GitHub e instalá-lo na sua máquina local.

   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
   ```

   Para instalar o emulador arm64, substitua o URL do repositório do GitHub no comando anterior pelo seguinte:

   ```
   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
```

Para instalar o emulador de arm64, substitua `$downloadLink` pelo seguinte:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Inicie a imagem do Docker com o comando docker run. Observe o seguinte:

   - `docker-image` é o nome da imagem e `test` é a tag.
   - `/usr/bin/java -cp './*' com.amazonaws.services.lambda.runtime.api.client.AWSLambda example.App::sayHello` é o `ENTRYPOINT` seguido pelo `CMD` do 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
```

Esse comando executa a imagem como um contêiner e cria um endpoint local em `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se você criou a imagem do Docker para a arquitetura do conjunto de instruções ARM64, certifique-se de usar a opção `--platform linux/arm64`, em vez de `--platform linux/amd64`.

3. Publique um evento no endpoint local.

   Linux/macOS

   No Linux e no MacOS, execute o seguinte comando `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   No PowerShell, execute o seguinte comando `Invoke-WebRequest`:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Obtenha o ID do contêiner.

```
docker ps
```

5. Use o comando docker kill para parar o contêiner. Nesse comando, substitua `3766c4ab331c` pelo ID do contêiner da etapa anterior.

```
docker kill 3766c4ab331c
```

Implantação da imagem

Para enviar a imagem ao Amazon ECR e criar a função do Lambda

1. Execute o comando get-login-password para autenticar a CLI do Docker no seu registro do Amazon ECR.

   - Defina o valor `--region` para a Região da AWS onde você deseja criar o repositório do Amazon ECR.
   - Substituir `111122223333` por seu ID da Conta da 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. Crie um repositório no Amazon ECR usando o 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**
>
> O repositório do Amazon ECR deve estar na mesma Região da AWS que a função do Lambda.

Se tiver êxito, você verá uma resposta como esta:

```
{
    "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. Copie o `repositoryUri` da saída na etapa anterior.
4. Execute o comando docker tag para aplicar uma tag na sua imagem local em seu repositório do Amazon ECR como a versão mais recente. Neste comando:

   - Substitua `docker-image:test` pelo nome e tag da sua imagem do Docker.
   - Substitua `<ECRrepositoryUri>` pelo `repositoryUri` que você copiou. Certifique-se de incluir `:latest` no final do URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Exemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Execute o comando docker push para implantar a imagem local no repositório do Amazon ECR. Certifique-se de incluir `:latest` no final do URI do repositório.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crie um perfil de execução para a função, caso ainda não tenha um. Você precisará do nome do recurso da Amazon (ARN) do perfil na próxima etapa.
7. Criar a função do Lambda. Em `ImageUri`, especifique o URI do repositório anterior. Certifique-se de incluir `:latest` no final do 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**
>
> É possível criar uma função usando uma imagem em uma conta da AWS diferente desde que a imagem esteja na mesma região da função do Lambda. Para ter mais informações, consulte Permissões entre contas do Amazon ECR.

8. Invoque a função.

```
aws lambda invoke --function-name hello-world response.json
```

Você obterá uma resposta parecida com esta:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Para ver a saída da função, verifique o arquivo `response.json`.

Para atualizar o código da função, você deve criar a imagem novamente, fazer upload da nova imagem no repositório do Amazon ECR e, em seguida, usar o comando update-function-code para implantar a imagem na função do Lambda.

O Lambda resolve a tag de imagem em um resumo de imagem específico. Isso significa que, se você apontar a tag de imagem que foi usada para implantar a função em uma nova imagem no Amazon ECR, o Lambda não atualizará automaticamente a função para usar a nova imagem. Para implantar a nova imagem na mesma função do Lambda, você deverá usar o comando `update-function-code`, mesmo que a tag da imagem no Amazon ECR permaneça a mesma.

Como trabalhar com camadas para funções do Lambda em Java

Uma camada do Lambda é um arquivo .zip que contém código ou dados complementares. As camadas geralmente contêm dependências de biblioteca, um runtime personalizado ou arquivos de configuração. A criação de uma camada envolve três etapas gerais:

1. Empacotar o conteúdo da sua camada. Isso significa criar um arquivo .zip contendo as dependências que você deseja usar em suas funções.
2. Criar a camada no Lambda.
3. Adicionar a camada às suas funções.

Este tópico contém etapas e orientações sobre como empacotar e criar adequadamente uma camada do Lambda em Java com dependências externas de bibliotecas.

Tópicos

Pré-requisitos

Para seguir as etapas desta seção, você deve ter o seguinte:

- Java 21
- Apache Maven 3.8.6 ou posterior
- AWS Command Line Interface (AWS CLI) versão 2

> **Note**
>
> Certifique-se de que a versão Java à qual o Maven se refere seja a mesma versão Java da função que você pretende implantar. Por exemplo, para uma função em Java 21, o comando `mvn -v` deverá listar a versão 21 do Java na saída:

```
Apache Maven 3.8.6
...
Java version: 21.0.2, vendor: Oracle Corporation, runtime: /Library/Java/
JavaVirtualMachines/jdk-21.jdk/Contents/Home
...
```

Neste tópico, mencionamos a aplicação de amostra `layer-java` no repositório awsdocs no GitHub. Essa aplicação contém scripts que baixam as dependências e geram a camada. A aplicação também contém uma função correspondente que usa dependências da camada. Após criar uma camada, você pode implantar e invocar a função correspondente para verificar se tudo funciona corretamente. Como você usa o runtime do Java 21 para as funções, as camadas também devem ser compatíveis com o Java 21.

A amostra de aplicação `layer-java` contém um único exemplo em dois subdiretórios. O diretório `layer` contém um arquivo `pom.xml` que define as dependências da camada, bem como scripts para gerar a camada. O diretório `function` contém uma amostra de função para ajudar a testar se a camada funciona. Este tutorial explica como criar e empacotar essa camada.

Compatibilidade da camada em Java com o Amazon Linux

A primeira etapa para criar uma camada é agrupar todo o conteúdo da camada em um arquivo .zip. Devido às funções do Lambda serem executadas no Amazon Linux, seu conteúdo da camada deve ser capaz de compilar e criar em um ambiente Linux.

O código Java foi projetado para ser independente da plataforma, de modo que você possa empacotar suas camadas em sua máquina local mesmo que ela não use um ambiente Linux. Após fazer o upload da camada Java para o Lambda, ela ainda será compatível com o Amazon Linux.

Caminhos de camada para runtimes do Java

Quando você adiciona uma camada a uma função, o Lambda carrega o conteúdo da camada no diretório `/opt` desse ambiente de execução. Para cada runtime do Lambda, a variável `PATH` já inclui caminhos de pasta específica no diretório `/opt`. Para garantir que a variável `PATH` colete o conteúdo da camada, o arquivo .zip da camada deve ter suas dependências nos seguintes caminhos de pasta:

- `java/lib`

Por exemplo, o arquivo .zip de camada resultante que você cria neste tutorial tem a seguinte estrutura de diretórios:

```
layer_content.zip
# java
    # lib
        # layer-java-layer-1.0-SNAPSHOT.jar
```

O arquivo JAR `layer-java-layer-1.0-SNAPSHOT.jar` (um uber-jar que contém todas as dependências necessárias) está localizado corretamente no diretório `java/lib`. Isso garante que o Lambda possa localizar a biblioteca durante as invocações da função.

Empacotar o conteúdo de camada

Neste exemplo, você empacota as duas seguintes bibliotecas Java em um único arquivo JAR:

- aws-lambda-java-core: um conjunto mínimo de definições de interface para trabalhar com Java no AWS Lambda
- Jackson: um conjunto popular de ferramentas de processamento de dados, especialmente para trabalhar com JSON.

Conclua as etapas a seguir para instalar e empacotar o conteúdo de camada.

Para instalar e empacotar seu conteúdo de camada

1. Clone o repositório `aws-lambda-developer-guide` do GitHub, que contém a amostra de código de que você precisa no diretório `sample-apps/layer-java`.

   ```
   git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
   ```

2. Acesse o diretório `layer` do exemplo de aplicação `layer-java`. Esse diretório contém os scripts que você usa para criar e empacotar a camada corretamente.

   ```
   cd aws-lambda-developer-guide/sample-apps/layer-java/layer
   ```

3. Examine os arquivos `pom.xml` Na seção `<dependencies>`, você define as dependências que deseja incluir na camada, ou seja, as bibliotecas `aws-lambda-java-core` e `jackson-databind`. Você pode atualizar esse arquivo para incluir quaisquer dependências que quiser incluir em sua própria camada.

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**
>
> A seção `<build>` desse arquivo `pom.xml` contém dois plug-ins. O maven-compiler-plugin compila o código-fonte. O maven-shade-plugin empacota seus artefatos em um único uber-jar.

4. Verifique se você tem permissões para executar os dois scripts.

```
chmod 744 1-install.sh && chmod 744 2-package.sh
```

5. Execute o script `1-install.sh` usando o comando a seguir:

```
./1-install.sh
```

Esse script executa `mvn clean install` no diretório atual. Isso criará o uber-jar com todas as dependências necessárias no diretório `target/`.

Example 1-install.sh

```
mvn clean install
```

6. Execute o script `2-package.sh` usando o comando a seguir:

```
./2-package.sh
```

Esse script cria a estrutura de diretórios `java/lib` que é necessária para empacotar adequadamente o conteúdo da camada. Em seguida, ele copia o uber-jar do diretório `/target` para o diretório `java/lib` recém-criado. Por fim, o script compacta o conteúdo do diretório `java` em um arquivo chamado `layer_content.zip`. Esse é o arquivo .zip da sua camada. É possível descompactar o arquivo e verificar se ele contém a estrutura de arquivo correta, conforme apresentado na seção the section called “Caminhos de camada para runtimes do Java”.

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
```

Como criar a camada

Nesta seção, você pega o arquivo `layer_content.zip` gerado na seção anterior e o carrega como uma camada do Lambda. É possível fazer upload de uma camada usando o AWS Management Console ou a API do Lambda por meio da AWS Command Line Interface (AWS CLI). Ao carregar seu arquivo .zip de camada, no seguinte comando PublishLayerVersion da AWS CLI, especifique `java21` como o runtime compatível e `arm64` como a arquitetura compatível.

```
aws lambda publish-layer-version --layer-name java-jackson-layer \
    --zip-file fileb://layer_content.zip \
    --compatible-runtimes java21 \
    --compatible-architectures "arm64"
```

Na resposta, anote o `LayerVersionArn`, que será algo como `arn:aws:lambda:us-east-1:`*`123456789012`*`:layer:java-jackson-layer:1`. Você precisará desse nome do recurso da Amazon (ARN) na próxima etapa deste tutorial, ao adicionar a camada à função.

Como adicionar a camada à sua função

Nesta seção, você implanta uma amostra de função do Lambda que usa a biblioteca Jackson em seu código de função e, em seguida, você anexa a camada. Para implantar a função, você precisará de um the section called “Perfil de execução (permissões para funções acessarem outros recursos)”. Se ainda não tiver um perfil de execução, siga as etapas na seção recolhível. Caso contrário, pule para a próxima seção para implantar a função.

(Opcional) Criar um perfil de execução

Para criar uma função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione Create role (Criar função).
3. Crie uma função com as propriedades a seguir.

   - Entidade confiável–Lambda.
   - Permissions (Permissões): AWSLambdaBasicExecutionRole.
   - Role name (Nome da função): **`lambda-role`**.

   A política AWSLambdaBasicExecutionRole tem as permissões necessárias para a função gravar logs no CloudWatch Logs.

Para implantar a função do Lambda

1. Navegue até o diretório `function/`. Se você estiver no diretório `layer/`, execute o seguinte comando:

   ```
   cd ../function
   ```

2. Revise o código de função. A função recebe um `Map<String, String>` como entrada e usa a biblioteca Jackson para gravar a entrada como uma string JSON antes de convertê-la em um objeto Java F1Car predefinido. Por fim, a função usa campos do objeto F1Car para estruturar uma string que a função retorna.

   ```
   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. Compile o projeto usando o seguinte comando do Maven:

```
mvn package
```

Esse comando produzirá um arquivo JAR chamado `layer-java-function-1.0-SNAPSHOT.jar` no diretório `target/`.

4. Implantar a função. No seguinte comando da AWS CLI, substitua o parâmetro `--role` pelo ARN do seu perfil de execução:

```
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
```

(Opcional) Invocar sua função sem anexar uma camada

Nessa fase, é possível, opcionalmente, tentar invocar sua função antes de anexar a camada. Se tentar fazer isso, você deverá receber um `ClassNotFoundException` porque sua função não pode fazer referência ao pacote `requests`. Para invocar sua função, use o seguinte comando da 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
```

Você deve ver uma saída semelhante a:

```
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

Para ver o erro específico, abra o arquivo `response.json` de saída. Você deverá ver um `ClassNotFoundException` com a seguinte mensagem de erro:

```
"errorMessage":"com.fasterxml.jackson.databind.ObjectMapper","errorType":"java.lang.ClassNotFou
```

Em seguida, anexe a camada à sua função. No seguinte comando da AWS CLI, substitua o parâmetro `--layers` pelo ARN da versão de camada que você anotou anteriormente:

```
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"
```

Por fim, tente invocar sua função usando o seguinte comando da 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
```

Você deve ver uma saída semelhante a:

```
{
```

```
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Isso indica que a função conseguiu usar a dependência da biblioteca Jackson para executar adequadamente a função. Você pode verificar se o arquivo `response.json` de saída contém a string retornada correta:

```
"Max Verstappen is a driver for team Red Bull"
```

(Opcional) Limpar os recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Para excluir a camada do Lambda

1. Abra a página Camadas do console do Lambda.
2. Selecione a camada que você criou.
3. Escolha Excluir, depois escolha Excluir novamente.

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Aprimoramento da performance de inicialização com o Lambda SnapStart

O Lambda SnapStart para Java pode aprimorar a performance de inicialização de aplicações sensíveis à latência em até dez vezes sem custos extras e, geralmente, sem alterações no código da função. O maior contribuinte para a latência de inicialização (geralmente chamado de tempo de inicialização a frio) é o tempo que o Lambda demora para inicializar a função, que inclui carregar o código da função, iniciar o runtime e inicializar o código da função.

Com o SnapStart, o Lambda inicializará a função quando você publicar uma versão da função. O Lambda usa um snapshot microVM do Firecracker do estado da memória e do disco do ambiente de execução inicializado, criptografa o snapshot e o armazena em cache para acesso de baixa latência. Quando você invoca a versão da função pela primeira vez e à medida que aumenta a escala verticalmente das invocações, o Lambda retoma novos ambientes de execução do snapshot armazenado em cache, em vez de realizar a inicialização do zero, melhorando a latência de inicialização.

> **Important**
>
> Se suas aplicações dependerem da exclusividade de estado, você deverá avaliar o código da função e verificar se ele é resiliente às operações de snapshots. Para ter mais informações, consulte Lidando com a exclusividade com o Lambda SnapStart.

Tópicos

- Modelo de segurança para Lambda SnapStart
- Maximizar a performance do SnapStart do Lambda

Recursos compatíveis e limitações

O SnapStart é compatível com os runtimes gerenciados do Java 11 e versões posteriores. Outros runtimes gerenciados (como `nodejs20.x` e `python3.12`), Runtimes somente para sistema operacional e imagens de contêiner não são compatíveis.

O SnapStart não oferece suporte à simultaneidade provisionada, à arquitetura arm64, ao Amazon Elastic File System (Amazon EFS) ou a armazenamento temporário superior a 512 MB.

Para trabalhar com o SnapStart, é possível usar o console do Lambda, a AWS Command Line Interface (AWS CLI), a API do Lambda, o AWS SDK for Java, o AWS CloudFormation, o AWS Serverless Application Model (AWS SAM) e o AWS Cloud Development Kit (AWS CDK). Para ter mais informações, consulte Ativando e gerenciando o Lambda SnapStart.

> **Note**
>
> É possível usar o SnapStart somente em versões de funções publicadas e aliases que direcionam para versões. Não é possível usar o SnapStart em uma versão não publicada de uma função ($LATEST).

Regiões compatíveis

O SnapStart está disponível nas seguintes Regiões da AWS:

- Leste dos EUA (Norte da Virgínia)
- Leste dos EUA (Ohio)
- Oeste dos EUA (N. da Califórnia)
- Oeste dos EUA (Oregon)
- África (Cidade do Cabo)
- Ásia-Pacífico (Hong Kong)
- Asia Pacific (Mumbai)
- Ásia-Pacífico (Hyderabad)

- Ásia-Pacífico (Tóquio)
- Ásia-Pacífico (Seul)
- Asia Pacific (Osaka)
- Ásia-Pacífico (Singapura)
- Ásia-Pacífico (Sydney)
- Ásia-Pacífico (Jacarta)
- Ásia-Pacífico (Melbourne)
- Canadá (Central)
- Europa (Estocolmo)
- Europa (Frankfurt)
- Europa (Zurique)
- Europa (Irlanda)
- Europa (Londres)
- Europa (Paris)
- Europa (Milão)
- Europa (Espanha)
- Oriente Médio (Emirados Árabes Unidos)
- Middle East (Bahrain)
- South America (São Paulo)

Considerações sobre compatibilidade

Com o SnapStart, o Lambda usa um único snapshot como estado inicial para diversos ambientes de execução. Se sua função usar qualquer um dos itens a seguir durante a fase de inicialização, talvez seja necessário realizar algumas alterações antes de usar o SnapStart:

Exclusividade

Se o código de inicialização gerar conteúdo exclusivo que é incluído no snapshot, o conteúdo poderá não ser exclusivo quando for reutilizado em ambientes de execução. Para manter a exclusividade ao usar o SnapStart, você deve gerar conteúdo exclusivo após a inicialização. Isso inclui IDs exclusivos, segredos exclusivos e entropia que são usados para gerar a

pseudoaleatoriedade. Para saber como restaurar a exclusividade, consulte Lidando com a exclusividade com o Lambda SnapStart.

Conexões de rede

O estado das conexões que a função estabelece durante a fase de inicialização não é garantido quando o Lambda retoma a função de um snapshot. Valide o estado das conexões de rede e restabeleça-as conforme necessário. Na maioria dos casos, as conexões de rede que um SDK da AWS estabelece são retomadas automaticamente. Para outras conexões, analise as práticas recomendadas.

Dados temporários

Algumas funções realizam o download ou inicializam dados efêmeros, como credenciais temporárias ou carimbos de data e hora em cache, durante a fase de inicialização. Atualize os dados efêmeros no manipulador de função antes de usá-los, mesmo quando não estiver usando o SnapStart.

Preços do SnapStart

Não há custos adicionais para o SnapStart. Você será cobrado com base no número de solicitações para as funções, no tempo que o código demora para ser executado e na memória configurada para a função. A duração é calculada a partir do momento em que o código começa a ser executado até que ele retorne ou seja encerrado, com arredondamento para o 1 ms mais próximo.

As cobranças por duração se aplicam ao código executado no manipulador de função, ao código de inicialização declarado fora do manipulador, ao tempo que demora para o runtime (JVM) carregar e aos códigos executados em um hook de runtime. Para obter mais informações sobre como o Lambda calcula a duração, consulte Monitoramento para Lambda SnapStart.

Para funções configuradas com o SnapStart, o Lambda recicla periodicamente os ambientes de execução e executa novamente o código de inicialização. Para obter resiliência, o Lambda cria snapshots em diversas zonas de disponibilidade. As cobranças são aplicadas sempre que o Lambda executa novamente o código de inicialização em outra zona de disponibilidade. Para obter mais informações sobre como o Lambda calcula as cobranças, consulte Definição de preço do AWS Lambda.

Comparação entre o Lambda SnapStart e a simultaneidade provisionada

Tanto o Lambda SnapStart quanto a simultaneidade provisionada poderão reduzir inicializações a frio e latências discrepantes quando uma função aumentar a escala verticalmente. O SnapStart ajuda você a aprimorar a performance da inicialização em até dez vezes sem custos extras. A simultaneidade provisionada mantém as funções inicializadas e prontas para responder em milissegundos de dois dígitos. A configuração da simultaneidade provisionada gera cobranças em sua conta da Conta da AWS. Use a simultaneidade provisionada se sua aplicação tiver requisitos rígidos de latência de inicialização a frio. Não é possível usar o SnapStart e a simultaneidade provisionada na mesma versão da função.

> **Note**
>
> O SnapStart funciona melhor quando usado com invocações de funções em escala. As funções que são invocadas com pouca frequência podem não ter as mesmas melhorias de desempenho.

Recursos adicionais do

Além de ler os outros tópicos deste capítulo, também recomendamos que você veja o workshop Início mais rápido com o AWS Lambda SnapStart e assista à sessão Rápidas inicializações a frio para suas funções Java do AWS re:Invent 2022.

Ativando e gerenciando o Lambda SnapStart

Para usar SnapStart, ative SnapStart em uma função Lambda nova ou existente. Em seguida, publique e invoque uma versão da função.

Tópicos

Ativando SnapStart (console)

SnapStart Para ativar uma função

1. Abra a página Funções do console do Lambda.
2. Escolha o nome de uma função.
3. Escolha Configuration (Configuração) e, em seguida, selecione General configuration (Configuração geral).
4. No painel General configuration (Configuração geral), escolha Edit (Editar).
5. Na página Editar configurações básicas, para SnapStart, escolha Versões publicadas.
6. Escolha Salvar.
7. Publique uma versão da função. O Lambda inicializa o código, cria um snapshot do ambiente de execução inicializado e, em seguida, armazena em cache o snapshot para acesso de baixa latência.
8. Invoque a versão da função.

Ativando SnapStart () AWS CLI

SnapStart Para ativar uma função existente

1. Atualize a configuração da função executando o update-function-configurationcomando com a --snap-start opção.

```
aws lambda update-function-configuration \
  --function-name my-function \
  --snap-start ApplyOn=PublishedVersions
```

2. Publique uma versão de função com o comando publish-version.

```
aws lambda publish-version \
  --function-name my-function
```

3. Confirme se SnapStart está ativado para a versão da função executando o get-function-configurationcomando e especificando o número da versão. O exemplo a seguir especifica a versão 1.

```
aws lambda get-function-configuration \
  --function-name my-function:1
```

   Se a resposta mostrar que OptimizationStatusé On e o estado é`Active`, ela SnapStart é ativada e um instantâneo está disponível para a versão da função especificada.

```
"SnapStart": {
    "ApplyOn": "PublishedVersions",
    "OptimizationStatus": "On"
 },
 "State": "Active",
```

4. Invoque a versão da função ao executar o comando invoke e especificar a versão. O exemplo a seguir invoca a versão 1.

```
aws lambda invoke \
  --cli-binary-format raw-in-base64-out \
  --function-name my-function:1 \
  --payload '{ "name": "Bob" }' \
  response.json
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

Para ativar SnapStart quando você cria uma nova função

1. Crie uma função ao executar o comando create-function com a opção --snap-start. Para --role, especifique o nome do recurso da Amazon (ARN) do perfil de execução.

```
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. Crie uma versão com o comando publish-version.

```
aws lambda publish-version \
  --function-name my-function
```

3. Confirme se SnapStart está ativado para a versão da função executando o get-function-configurationcomando e especificando o número da versão. O exemplo a seguir especifica a versão 1.

```
aws lambda get-function-configuration \
  --function-name my-function:1
```

Se a resposta mostrar que OptimizationStatusé 0n e o estado é`Active`, ela SnapStart é ativada e um instantâneo está disponível para a versão da função especificada.

```
"SnapStart": {
     "ApplyOn": "PublishedVersions",
     "OptimizationStatus": "On"
  },
  "State": "Active",
```

4. Invoque a versão da função ao executar o comando invoke e especificar a versão. O exemplo a seguir invoca a versão 1.

```
aws lambda invoke \
  --cli-binary-format raw-in-base64-out \
  --function-name my-function:1 \
  --payload '{ "name": "Bob" }' \
  response.json
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

Ativando SnapStart (API)

Para ativar SnapStart

1. Execute um destes procedimentos:
   - Crie uma nova função SnapStart ativada usando a ação CreateFunctionda API com o SnapStartparâmetro.
   - Ative SnapStart para uma função existente usando a UpdateFunctionConfigurationação com o SnapStartparâmetro.
2. Publique uma versão da função com a PublishVersionação. O Lambda inicializa o código, cria um snapshot do ambiente de execução inicializado e, em seguida, armazena em cache o snapshot para acesso de baixa latência.
3. Confirme se SnapStart está ativado para a versão da função usando a GetFunctionConfigurationação. Especifique um número de versão para confirmar que SnapStart está ativado para essa versão. Se a resposta mostrar que OptimizationStatusé 0n e o estado é`Active`, ela SnapStart é ativada e um instantâneo está disponível para a versão da função especificada.

```
"SnapStart": {
        "ApplyOn": "PublishedVersions",
        "OptimizationStatus": "On"
    },
    "State": "Active",
```

4. Invoque a versão de função com a ação Invoke.

Lambda SnapStart e estados de função

Os seguintes estados de função podem ocorrer quando você usa SnapStart. Eles também podem ocorrer quando o Lambda recicla periodicamente o ambiente de execução e executa novamente o código de inicialização de uma função configurada com. SnapStart

- `Pending`: o Lambda está inicializando o código e obtendo um snapshot do ambiente de execução inicializado. Quaisquer invocações ou outras ações de API que operam na versão de função falharão.
- `Active`: a criação do snapshot está concluída e é possível invocar a função. Para usar SnapStart, você deve invocar a versão publicada da função, não a versão não publicada ($LATEST).
- `Inactive`: a versão de função não é invocada há 14 dias. Quando a versão da função se torna `Inactive`, o Lambda exclui o snapshot. Se você invocar a versão de função após 14 dias, o Lambda retornará uma resposta `SnapStartNotReadyException` e começará a inicializar um novo snapshot. Aguarde até que a versão de função atinja o estado `Active` e, em seguida, invoque-a novamente.
- `Failed`: o Lambda encontrou um erro ao executar o código de inicialização ou criar o snapshot.

Atualização de um snapshot

O Lambda cria um snapshot para cada versão de função publicada. Para atualizar um snapshot, publique uma nova versão da função. O Lambda atualiza automaticamente os snapshots com os patches de segurança e runtime mais recentes.

Usando SnapStart com o AWS SDK for Java

Para fazer chamadas para o SDK da AWS usando sua função, o Lambda gera um conjunto efêmero de credenciais ao assumir o perfil de execução da função. Essas credenciais estão disponíveis como variáveis de ambiente durante a invocação da sua função. Não é necessário fornecer credenciais para o SDK diretamente no código. Por padrão, a cadeia de provedores de credenciais verifica sequencialmente cada local em que você pode definir credenciais e escolhe o primeiro disponível, que geralmente corresponde às variáveis de ambiente (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` e `AWS_SESSION_TOKEN`).

> **Note**
>
> Quando SnapStart ativado, o Java Runtime usa automaticamente as credenciais do contêiner (`AWS_CONTAINER_CREDENTIALS_FULL_URI`e`AWS_CONTAINER_AUTHORIZATION_TOKEN`) em vez das variáveis de ambiente da chave de acesso. Isso evita que as credenciais expirem antes que a função seja restaurada.

Usando SnapStart com AWS CloudFormationAWS SAM, e AWS CDK

- AWS CloudFormation: declare a SnapStartentidade em seu modelo.
- AWS Serverless Application Model(AWS SAM): Declare a SnapStartpropriedade em seu modelo.
- AWS Cloud Development Kit (AWS CDK): Use o SnapStartPropertytipo.

Exclusão de snapshots

O Lambda exclui snapshots quando:

- Você exclui a função ou a versão de função.
- Você não invoca a versão de função por 14 dias. Após 14 dias sem uma invocação, a versão de função transita para o estado Inactive (Inativo). Se você invocar a versão de função após 14 dias, o Lambda retornará uma resposta `SnapStartNotReadyException` e começará a inicializar um novo snapshot. Aguarde até que a versão de função atinja o estado Active (Ativo) e, em seguida, invoque-a novamente.

O Lambda remove todos os recursos associados aos snapshots excluídos em conformidade com o Regulamento Geral sobre a Proteção de Dados (GDPR).

Lidando com a exclusividade com o Lambda SnapStart

Quando as invocações aumentam em uma SnapStart função, o Lambda usa um único snapshot inicializado para retomar vários ambientes de execução. Se o código de inicialização gerar conteúdo exclusivo que é incluído no snapshot, o conteúdo poderá não ser exclusivo quando for reutilizado em ambientes de execução. Para manter a exclusividade durante o uso SnapStart, você deve gerar conteúdo exclusivo após a inicialização. Isso inclui IDs exclusivos, segredos exclusivos e entropia que são usados para gerar a pseudoaleatoriedade.

Recomendamos as práticas recomendadas a seguir para ajudar você a manter a exclusividade em seu código. O Lambda também fornece uma ferramenta de SnapStart digitalização de código aberto para ajudar a verificar se há código que pressupõe exclusividade. Se você gerar dados exclusivos durante a fase de inicialização, poderá usar um gancho de tempo de execução para restaurar a exclusividade. Com ganchos de tempo de execução, é possível executar um código específico imediatamente antes que o Lambda obtenha um snapshot ou imediatamente após o Lambda retornar uma função de um snapshot.

Evite salvar um estado que depende da exclusividade durante a inicialização

Durante a fase de inicialização da função, evite armazenar em cache dados que devem ser exclusivos, como a geração de um ID exclusivo para registro em log. Em vez disso, recomendamos gerar dados exclusivos dentro do manipulador de função ou usar um gancho de tempo de execução.

Example : geração de um ID exclusivo no manipulador de função

O exemplo a seguir demonstra como gerar um UUID no manipulador de função.

```
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!";
    }
  }
```

Use geradores de números pseudoaleatórios criptograficamente seguros (CSPRNGs)

Se a aplicação depender da aleatoriedade, recomendamos usar geradores de números pseudoaleatórios criptograficamente seguros (CSPRNGs). O tempo de execução gerenciado do Lambda para Java inclui dois CSPRNGs integrados (OpenSSL 1.0.2 e) que mantêm automaticamente a aleatoriedade com. `java.security.SecureRandom` SnapStart Software que sempre obtém números aleatórios de `/dev/random` ou `/dev/urandom` também mantém a aleatoriedade com SnapStart.

Example — java.security. SecureRandom

O exemplo a seguir usa `java.security.SecureRandom`, que gera sequências numéricas exclusivas mesmo quando a função é restaurada de um 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 ferramenta de digitalização

O Lambda fornece uma ferramenta de verificação para ajudar você a verificar o código que assume exclusividade. A ferramenta de SnapStart digitalização é um SpotBugsplug-in de código aberto que executa uma análise estática em relação a um conjunto de regras. A ferramenta de verificação ajuda a identificar possíveis implementações de código que podem quebrar suposições em relação à exclusividade. Para obter instruções de instalação e uma lista das verificações que a ferramenta de escaneamento executa, consulte o repositório aws-lambda-snapstart-java-rules em. GitHub

Para saber mais sobre como lidar com a exclusividade com SnapStart, consulte Como começar mais rápido com AWS Lambda SnapStart no blog de AWS computação.

Implementar código antes ou depois dos snapshots da função do Lambda

É possível usar hooks de runtime para implementar o código antes que o Lambda crie um snapshot ou depois que o Lambda retorna uma função de um snapshot. Os hooks de runtime estão disponíveis como parte do projeto de código aberto Coordinated Restore at Checkpoint (CRaC). O CRaC está em desenvolvimento para o Open Java Development Kit (OpenJDK). Para obter um exemplo de como usar o CRaC com uma aplicação de referência, consulte o repositório CRaC no GitHub. O CRaC usa três elementos principais:

- `Resource`: uma interface com dois métodos, `beforeCheckpoint()` e `afterRestore()`. Use esses métodos para implementar o código deseja executar antes de um snapshot e depois de uma restauração.
- `Context <R extends Resource>`: para receber notificações de pontos de verificação e restaurações, `Resource` deve estar registrado em um `Context`.
- `Core`: o serviço de coordenação, que fornece o padrão global `Context` por meio do método estático `Core.getGlobalContext()`.

Para obter mais informações sobre `Context` e `Resource`, consulte Package org.crac (Pacote org.crac) na documentação do CRaC.

Use as etapas a seguir para implementar os hooks de runtime com o pacote org.crac. O runtime do Lambda contém uma implementação de contexto CRaC personalizada que chama os hooks de runtime antes do ponto de verificação e depois da restauração.

Etapa 1: atualizar a configuração de compilação

Adicione a dependência `org.crac` à configuração de compilação. O exemplo a seguir usa o Gradle. Para obter exemplos de outros sistemas de compilação, consulte a documentação do 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'
}
```

Etapa 2: atualizar o manipulador do Lambda

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. A função é executada até que o manipulador retorne uma resposta, seja encerrado ou atinja o tempo limite.

Para ter mais informações, consulte Definir o manipulador da função do Lambda em Java.

O manipulador exemplificado a seguir mostra como executar o código antes do ponto de verificação (`beforeCheckpoint()`) e depois da restauração (`afterRestore()`). Esse manipulador também registra o `Resource` no `Context` global gerenciado pelo runtime.

> **Note**
>
> Quando o Lambda cria um snapshot, o código de inicialização pode ser executado por até 15 minutos. O limite de tempo é de 130 segundos ou o tempo limite da função configurada (máximo de 900 segundos), o que for maior. Seus hooks de runtime de `beforeCheckpoint()` contam até o limite de tempo do código de inicialização. Quando o Lambda restaura um snapshot, o runtime (JVM) deve ser carregado e os hooks de runtime `afterRestore()` devem ser concluídos dentro do limite de tempo-limite (dez segundos). Caso contrário, você obterá uma 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` mantém somente uma `WeakReference` para o objeto registrado. Se um `Resource` estiver na coleta de resíduos, os hooks de runtime não serão executados. O código deve manter uma referência forte para o `Resource` com a finalidade de garantir que o hook de runtime seja executado.

Veja a seguir dois exemplos de padrões a serem evitados:

Example : objeto sem uma referência forte

```
Core.getGlobalContext().register( new MyResource() );
```

Example : objetos de classes anônimas

```
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 {
    // ...
   }

} );
```

Em vez disso, mantenha uma referência forte. No exemplo a seguir, o recurso registrado não está na coleta de resíduos e os hooks de runtime são executados de forma consistente.

Example : objeto com uma referência forte

```
Resource myResource = new MyResource(); // This reference must be maintained to prevent
 the registered resource from being garbage collected
Core.getGlobalContext().register( myResource );
```

Monitoramento para Lambda SnapStart

Você pode monitorar suas SnapStart funções do Lambda usando Amazon CloudWatch,AWS X-Ray, e. API de Telemetria do Lambda

> **Note**
>
> As variáveis de `AWS_LAMBDA_LOG_STREAM_NAME` ambiente `AWS_LAMBDA_LOG_GROUP_NAME` e não estão disponíveis nas funções Lambda SnapStart .

CloudWatch para SnapStart

Há algumas diferenças com o formato do fluxo de CloudWatch log para SnapStart funções:

- Logs de inicialização: quando um novo ambiente de execução é criado, o `REPORT` não inclui o campo `Init Duration`. Isso porque o Lambda inicializa SnapStart funções quando você cria uma versão em vez de durante a invocação da função. Para SnapStart funções, o `Init Duration` campo está no `INIT_REPORT` registro. Este registro mostra detalhes de duração para Fase de inicialização, incluindo a duração de quaisquer ganchos de runtime `beforeCheckpoint`.
- Logs de invocação: quando um novo ambiente de execução é criado, o `REPORT` inclui os campos `Restore Duration` e `Billed Restore Duration`:
  - `Restore Duration`: o tempo necessário para o Lambda restaurar um snapshot, carregar o runtime (JVM) e executar qualquer hook de runtime de `afterRestore`. O processo de restauração de snapshots pode incluir o tempo gasto em atividades fora da MicroVM. Esse tempo é relatado em `Restore Duration`.
  - `Billed Restore Duration`: o tempo necessário para o Lambda carregar o runtime (JVM) e executar qualquer gancho de `afterRestore`. Você não é cobrado pelo tempo necessário para restaurar um snapshot.

> **Note**
>
> As cobranças por duração se aplicam ao código executado no handler de função, ao código de inicialização declarado fora do handler, ao tempo que demora para o runtime (JVM) carregar e aos códigos executados em um hook de runtime. Para ter mais informações, consulte Preços do SnapStart.

A duração da inicialização a frio corresponde a soma de `Restore Duration` + `Duration`.

O exemplo a seguir é uma consulta do Lambda Insights que retorna os percentis de latência das funções. SnapStart Para obter mais informações sobre consultas do Lambda Insights, consulte Exemplo de fluxo de trabalho usando consultas para solucionar problemas de uma função.

```
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
```

Rastreamento ativo X-Ray para SnapStart

Você pode usar o X-Ray para rastrear solicitações às funções do Lambda SnapStart . Há algumas diferenças entre os subsegmentos de X-Ray para SnapStart funções:

- Não há `Initialization` subsegmento para SnapStart funções.
- O subsegmento `Restore` mostra o tempo necessário para o Lambda restaurar um snapshot, carregar o runtime (JVM) e executar qualquer hook de runtime do `afterRestore`. O processo de restauração de snapshots pode incluir o tempo gasto em atividades fora da MicroVM. Esse tempo é relatado no subsegmento `Restore`. Você não é cobrado pelo tempo gasto fora da microVM para restaurar um snapshot.

Eventos da API de telemetria para SnapStart

O Lambda envia os seguintes SnapStart eventos para o: API de telemetria

- `platform.restoreStart`: mostra o horário em que a Fase `Restore` foi iniciada.
- `platform.restoreRuntimeDone`: mostra se a fase `Restore` ocorreu com êxito. mO Lambda envia essa mensagem quando o runtime envia uma solicitação de API de runtime `restore/next`. Existem três status possíveis: com êxito, com falha e tempo limite.

- `platform.restoreReport`: mostra quanto tempo durou a fase `Restore` e por quantos milissegundos você foi cobrado durante essa fase.

Amazon API Gateway e métricas de URL da função

Se você criar uma API da web usando o API Gateway, poderá usar a IntegrationLatencymétrica para medir a end-to-end latência (o tempo entre o momento em que o API Gateway retransmite uma solicitação para o back-end e o momento em que recebe uma resposta do back-end).

Se você estiver usando um URL de função Lambda, poderá usar a UrlRequestLatencymétrica para medir a end-to-end latência (o tempo entre o momento em que o URL da função recebe uma solicitação e o momento em que o URL da função retorna uma resposta).

Modelo de segurança para Lambda SnapStart

O Lambda SnapStart oferece suporte à criptografia em repouso. O Lambda criptografa snapshots com uma AWS KMS key. Por padrão, o Lambda usa um Chave gerenciada pela AWS. Se esse comportamento padrão for adequado ao seu fluxo de trabalho, você não precisará configurar mais nada. Caso contrário, você pode usar a `--kms-key-arn` opção na função de criação ou no update-function-configurationcomando para fornecer uma chave gerenciada AWS KMS pelo cliente. É possível fazer isso para controlar a alternância da chave KMS ou para atender aos requisitos da organização para gerenciamento de chaves KMS. As chaves gerenciadas pelo cliente incorrem em cobranças do AWS KMS padrão. Para obter mais informações, consulte Definição de preços do AWS Key Management Service.

Quando você exclui uma SnapStart função ou versão da função, todas as `Invoke` solicitações para essa função ou versão da função falham. O Lambda exclui automaticamente os snapshots que não são invocados por 14 dias. O Lambda remove todos os recursos associados aos snapshots excluídos em conformidade com o Regulamento Geral sobre a Proteção de Dados (GDPR).

Maximizar a performance do SnapStart do Lambda

Tópicos

Ajuste de performance

> **Note**
>
> O SnapStart funciona melhor quando usado com invocações de funções em escala. As funções que são invocadas com pouca frequência podem não ter as mesmas melhorias de desempenho.

Para maximizar os benefícios do SnapStart, recomendamos carregar previamente as classes que contribuem para a latência de startup em seu código de inicialização, em vez de no manipulador de função. Isso remove a latência associada ao carregamento pesado de classes do caminho de invocação, otimizando a performance de startup com o SnapStart.

Se não for possível carregar previamente as classes durante a inicialização, recomendamos carregar previamente as classes com invocações fictícias. Para fazer isso, atualize o código do manipulador de função, conforme mostrado no exemplo a seguir da função “pet store” no repositório GitHub do AWS Labs.

```
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);
        }
    }
```

Práticas recomendadas de rede

O estado das conexões que a função estabelece durante a fase de inicialização não é garantido quando o Lambda retoma a função de um snapshot. Na maioria dos casos, as conexões de rede que um SDK da AWS estabelece são retomadas automaticamente. Para outras conexões, recomendamos as práticas recomendadas a seguir.

Restabelecer as conexões de rede

Sempre restabeleça as conexões de rede quando a função for retomada de um snapshot. Recomendamos restabelecer as conexões de rede no manipulador de função. Como alternativa, você pode usar um hook de runtime `afterRestore`.

Não usar o nome do host como um identificador exclusivo do ambiente de execução

Recomendamos não usar o `hostname` para identificar o ambiente de execução como um nó ou um contêiner exclusivo em suas aplicações. Com o SnapStart, um único snapshot é usado como estado inicial para diversos ambientes de execução, e todos os ambientes de execução retornam o mesmo valor do `hostname` para `InetAddress.getLocalHost()`. Para aplicações que requerem uma identidade de ambiente de execução ou um valor do `hostname` exclusivo, recomendamos gerar um ID exclusivo no manipulador de função. Como alternativa, use um hook de runtime `afterRestore` para gerar um ID exclusivo e, em seguida, use o ID exclusivo como identificador para o ambiente de execução.

Evitar vincular conexões a portas de origem fixas

Recomendamos evitar vincular conexões de rede a portas de origem fixas. As conexões são restabelecidas quando uma função é retomada de um snapshot, e as conexões de rede vinculadas a uma porta de origem fixa podem apresentar falhas.

Evitar usar o cache do DNS Java

As funções do Lambda já armazenam as respostas de DNS em cache. Se você usar outro cache DNS com o SnapStart, poderá experimentar tempos limite de conexão quando a função for retomada de um snapshot.

A classe `java.util.logging.Logger` pode habilitar indiretamente o cache do DNS da JVM. Para substituir as configurações padrão, defina networkaddress.cache.ttl como 0 antes de inicializar o `logger`. Exemplo:

```
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);
}
```

Para evitar falhas de `UnknownHostException`, é recomendável definir `networkaddress.cache.negative.ttl` como 0. Você pode definir essa propriedade para uma função do Lambda com a variável de ambiente `AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0`.

Desabilitar o cache do DNS da JVM não desabilita o cache do DNS gerenciado do Lambda.

Configurações de personalização da função do Lambda em Java

Esta página descreve as configurações específicas das funções do Java no AWS Lambda. Você pode usar essas configurações para personalizar o comportamento de inicialização de runtime do Java. Isso pode reduzir a latência geral da função e melhorar a performance geral da função sem a necessidade de modificar qualquer código.

Seções

`JAVA_TOOL_OPTIONS` variável de ambiente

No Java, o Lambda é compatível com a variável de ambiente `JAVA_TOOL_OPTIONS` para definir variáveis adicionais da linha de comando no Lambda. Você pode usar essa variável de ambiente de várias maneiras, como para personalizar configurações de compilação em camadas. O exemplo a seguir demonstra como usar a variável de ambiente `JAVA_TOOL_OPTIONS` para esse caso de uso.

Exemplo: personalizar configurações de compilação em camadas

A compilação em camadas é um recurso da máquina virtual Java (JVM). É possível usar configurações específicas de compilação em camadas para fazer o melhor uso dos compiladores just-in-time (JIT) da JVM. Normalmente, o compilador C1 é otimizado para um tempo de inicialização rápido. O compilador C2 é otimizado para a melhor performance geral, mas também usa mais memória e leva mais tempo para alcançá-la.

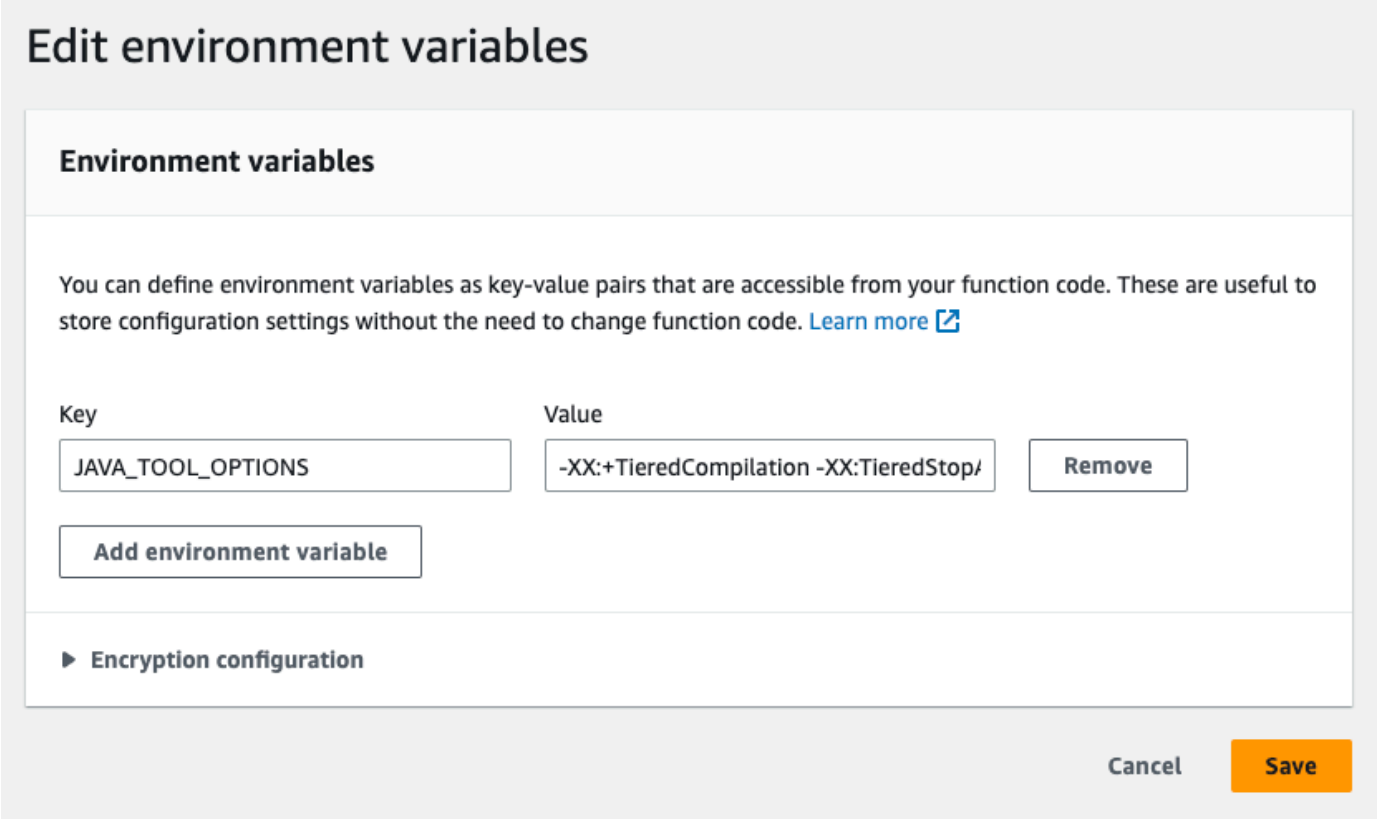
Existem cinco níveis diferentes de compilação em camadas. No nível 0, a JVM interpreta o código de bytes do Java. No nível 4, a JVM usa o compilador C2 para analisar os dados de criação de perfil coletados durante a inicialização da aplicação. Com o tempo, ele monitora o uso do código para identificar as melhores otimizações.

Personalizar o nível de compilação em camadas pode ajudar a reduzir a latência de inicialização a frio da função do Java. Por exemplo, defina o nível de compilação em camadas como 1 para que a JVM use o compilador C1. Esse compilador produz rapidamente código nativo otimizado, mas não gera qualquer dado de criação de perfil e nunca usa o compilador C2.

Por padrão, no runtime do Java 17, o sinalizador da JVM para compilação em camadas é configurado para ser interrompido no nível 1. No runtime do Java 11 e versões inferiores, você pode definir o nível de compilação em camadas como 1 ao executar as seguintes etapas:

Para personalizar as configurações de compilação em camadas (console)

1. Abra a página Funções no console do Lambda.
2. Escolha uma função do Java para a qual você deseja personalizar a compilação em camadas.
3. Escolha a guia Configuração e, em seguida, escolha Variáveis de ambiente no menu à esquerda.
4. Selecione a opção Editar.
5. Escolha Add environment variable (Adicionar variável de ambiente).
6. Na chave, insira `JAVA_TOOL_OPTIONS`. No valor, insira `-XX:+TieredCompilation -XX:TieredStopAtLevel=1`.



7. Escolha Salvar.

> **Note**
>
> Você também pode usar o Lambda SnapStart para mitigar problemas de inicialização a frio. O SnapStart usa snapshots em cache do ambiente de execução para melhorar significativamente a performance da inicialização. Para obter mais informações sobre os

recursos, limitações e regiões compatíveis com o SnapStart, consulte Aprimoramento da performance de inicialização com o Lambda SnapStart.

Exemplo: personalização do comportamento do GC usando JAVA_TOOL_OPTIONS

Os runtimes do Java 11 usam o Serial garbage collector (GC) para a coleta de resíduos. Por padrão, os runtimes do Java 17 também usam o Serial GC. No entanto, com o Java 17, você também pode usar a variável de ambiente `JAVA_TOOL_OPTIONS` para alterar o GC padrão. Você pode escolher entre o Parallel GC e o Shenandoah GC.

Por exemplo, se sua workload usa mais memória e diversas CPUs, considere usar o Parallel GC para obter melhor performance. Você pode fazer isso ao anexar o seguinte ao valor da variável de ambiente `JAVA_TOOL_OPTIONS`:

```
-XX:+UseParallelGC
```

Objeto de contexto do AWS Lambda em Java

Quando o Lambda executa a função, ele transmite um objeto de contexto para o handler. Esse objeto fornece métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de execução.

Métodos de contexto

- `getRemainingTimeInMillis()`— Retorna o número de milissegundos restantes antes do tempo limite da execução.
- `getFunctionName()`: retorna o nome da função do Lambda.
- `getFunctionVersion()`— Retorna oversionda função do.
- `getInvokedFunctionArn()`: retorna o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um número de versão ou alias.
- `getMemoryLimitInMB()`— Retorna a quantidade de memória alocada para a função.
- `getAwsRequestId()`— Retorna o identificador de invocação da solicitação.
- `getLogGroupName()`— Retorna o grupo de logs para a função do.
- `getLogStreamName()`— Retorna o stream de log para a instância da função.
- `getIdentity()`— (aplicativos móveis) Retorna informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
- `getClientContext()`— (aplicativos móveis) Retorna o contexto do cliente que é fornecido ao Lambda pelo aplicativo cliente.
- `getLogger()`: retorna o objeto logger para a função.

O exemplo a seguir mostra uma função que usa o objeto de contexto para acessar o logger do 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;
  }
}
```

A função registra em log o tipo de classe do evento de entrada antes de retornar `null`.

Example saída do log

```
EVENT TYPE: class java.util.LinkedHashMap
```

A interface para o objeto de contexto está disponível na biblioteca aws-lambda-java-core. É possível implementar essa interface para criar uma classe de contexto para teste. O exemplo a seguir mostra uma classe de contexto que retorna valores fictícios para a maioria das propriedades e um logger de teste funcional.

Example src/test/java/example/TestContext.java

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.CognitoIdentity;
import com.amazonaws.services.lambda.runtime.ClientContext;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

public class TestContext implements Context{

  public TestContext() {}
  public String getAwsRequestId(){
    return new String("495b12a8-xmpl-4eca-8168-160484189f99");
  }
  public String getLogGroupName(){
    return new String("/aws/lambda/my-function");
  }
```

```
  public String getLogStreamName(){
    return new String("2020/02/26/[$LATEST]704f8dxmpla04097b9134246b8438f1a");
  }
  public String getFunctionName(){
    return new String("my-function");
  }
  public String getFunctionVersion(){
    return new String("$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();
  }

}
```

Para obter mais informações sobre registro em log, consulte Registro em log da função do AWS Lambda em Java.

Contexto em aplicativos de exemplo

O repositório do GitHub para este guia inclui aplicativos de exemplo que demonstram o uso do objeto de contexto. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS Serverless Application Model (AWS SAM) e recursos de suporte.

Aplicações de exemplo do Lambda em Java

- java17-examples: uma função em Java que demonstra como usar um registro Java para representar um objeto de dados de evento de entrada.

- java-basic: uma coleção de funções Java mínimas com testes de unidade e configuração de registro em log variável.
- java-events: uma coleção de funções do Java contendo código básico sobre como lidar com eventos de vários serviços, como o Amazon API Gateway, o Amazon SQS e o Amazon Kinesis. Essas funções usam a versão mais recente da biblioteca aws-lambda-java-events (3.0.0 e versões mais recentes). Estes exemplos não exigem oAWS SDK como dependência.
- s3-java: uma função em Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.
- Use API Gateway to invoke a Lambda function (Usar o API Gateway para invocar uma função do Lambda): uma função Java que verifica uma tabela do Amazon DynamoDB contendo informações de funcionários. Em seguida, usa o Amazon Simple Notification Service para enviar uma mensagem de texto aos funcionários comemorando seus aniversários de empresa. Este exemplo usa o API Gateway para invocar a função.

Registro em log da função do AWS Lambda em Java

O AWS Lambda monitora automaticamente as funções do Lambda e envia entradas de logs para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente de runtime do Lambda envia detalhes sobre cada invocação e outras saídas do código da função para o fluxo de logs. Para obter mais informações sobre o CloudWatch Logs, consulte Usar logs do Amazon CloudWatch com o AWS Lambda.

Para gerar os logs do código da sua função, use métodos no java.lang.System ou qualquer módulo de registro em log que grave em stdout ou stderr.

Seções

Criar uma função que retorna logs

Para gerar os logs do código da função, use métodos no java.lang.System ou qualquer módulo de registro em log que grave no `stdout` ou no `stderr`. A biblioteca aws-lambda-java-core fornece uma classe do logger chamada `LambdaLogger` que você pode acessar a partir do objeto de contexto. A classe do logger oferece suporte a logs de várias linhas.

O exemplo a seguir usa o logger `LambdaLogger` fornecido pelo objeto de contexto.

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 do 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
```

O runtime do Java registra em log as linhas `START`, `END` e `REPORT` para cada invocação. A linha do relatório fornece os seguintes detalhes:

RELATAR campos de dados de linha

- RequestId: o ID de solicitação exclusivo para a invocação.
- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.
- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o runtime levou para carregar a função e executar o código fora do método do handler.
- XRAY TraceId: para solicitações rastreadas, o ID de rastreamento do AWS X-Ray.
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Usar controles avançados de registro em log do Lambda com Java

Para dar mais controle sobre como os logs das funções são capturados, processados e consumidos, você pode configurar as seguintes opções de log para runtimes compatíveis do Java:

- Formato do log: selecione entre texto simples e formato JSON estruturado para os logs da sua função.
- Nível de log: para logs no formato JSON, escolha o nível de detalhe dos logs enviados pelo Lambda para o CloudWatch, como ERROR, DEBUG ou INFO.
- Grupo de logs: escolha o grupo de logs do CloudWatch para o qual sua função envia logs.

Para obter mais informações sobre essas opções de registro em log e instruções sobre como configurar a função para usá-las, consulte the section called “Configurar controles avançados de registro em log para a função do Lambda”.

Para usar as opções de formato de log e nível de log com as funções do Lambda para Java, consulte as orientações nas seções a seguir.

Usar logs em formato JSON estruturado com Java

Se você selecionar JSON para o formato de log da função, o Lambda enviará a saída de logs usando a classe `LambdaLogger` como JSON estruturado. Cada objeto de log JSON contém pelo menos quatro pares de valores-chave com as seguintes chaves:

- `"timestamp"`: o horário em que a mensagem de log foi gerada.
- `"level"`: o nível de log atribuído à mensagem.
- `"message"`: o conteúdo da mensagem de log.
- `"AWSrequestId"`: o ID de solicitação exclusivo para invocar a função.

Dependendo do método de registro em log usado, as saídas de log da função capturadas no formato JSON também podem conter pares de valores de chave adicionais.

Para atribuir um nível aos logs que você criar usando o logger `LambdaLogger`, você precisa fornecer um argumento `LogLevel` no comando de registro em log, conforme mostrado no exemplo a seguir.

Example Código de registro em log do Java

```
LambdaLogger logger = context.getLogger();
logger.log("This is a debug log", LogLevel.DEBUG);
```

A saída de log desse exemplo de código seria capturada no CloudWatch Logs da seguinte forma:

Example Registro em log JSON

```
{
    "timestamp":"2023-11-01T00:21:51.358Z",
    "level":"DEBUG",
    "message":"This is a debug log",
    "AWSrequestId":"93f25699-2cbf-4976-8f94-336a0aa98c6f"
}
```

Se você não atribuir um nível à saída de log, o Lambda atribuirá automaticamente o nível INFO a ele.

Se o código já usa outra biblioteca de logs para produzir logs JSON estruturados, você não precisa fazer nenhuma alteração. O Lambda não codifica duas vezes nenhum log que já esteja codificado em JSON. Mesmo que a função seja configurada para usar logs em formato JSON, suas saídas de log serão exibidas no CloudWatch com a estrutura JSON que você definiu.

Usar a filtragem em nível de log com Java

Para o AWS Lambda filtrar os logs de aplicação de acordo com o nível de log, a função precisa usar logs em formato JSON. Isso pode ser feito de duas maneiras:

- Crie saídas de log usando o padrão `LambdaLogger` e configure a função para usar logs em formato JSON. Assim, o Lambda filtrará as saídas de log usando o par de valores-chave "nível" no objeto JSON descrito em the section called "Usar logs em formato JSON estruturado com Java". Para saber como configurar o formato de log da função, consulte the section called "Configurar controles avançados de registro em log para a função do Lambda".
- Use outro método ou biblioteca de logs para criar logs JSON estruturados no código, de modo que incluam um par de valores-chave de "nível" para definir o nível da saída de log. Você também pode usar qualquer biblioteca de logs que permita a gravação em logs JSON em `stdout` ou `stderr`. Por exemplo, é possível usar o Powertools para AWS Lambda ou o pacote Log4j2 para gerar saídas de log JSON estruturado do seu código. Consulte the section called "Uso do Powertools para AWS Lambda (Java) e do AWS SAM para registro em log estruturado" e the section called "Registro em log avançado com Log4j2 e SLF4J" para saber mais.

Ao configurar sua função para usar a filtragem em nível de log, você precisa selecionar entre as seguintes opções para o nível do logs enviados pelo Lambda para o CloudWatch Logs:

Nível de log	Uso padrão
TRACE (mais detalhes)	As informações mais detalhadas usadas para rastrear o caminho da execução do código
DEBUG	Informações detalhadas para depuração do sistema
INFO	Mensagens que registram a operação normal da função

Nível de log	Uso padrão
WARN	Mensagens sobre possíveis erros que podem levar a um comportamento inesperado se não forem corrigidos
ERRO	Mensagens sobre problemas que impedem que o código funcione conforme o esperado
FATAL (menos detalhes)	Mensagens sobre erros graves que fazem a aplicação parar de funcionar

Para que o Lambda filtre os logs da função, você também precisa incluir um par de valores-chave `"timestamp"` na saída do log JSON. A hora deve ser especificada em um formato RFC 3339 válido de carimbo de data/hora. Se você não fornecer um carimbo de data/hora válido, o Lambda atribuirá ao log o nível INFO e adicionará um carimbo de data/hora.

O Lambda envia logs do nível selecionado, e dos níveis inferiores, para o CloudWatch. Por exemplo, se você configurar um nível de log de WARN, o Lambda enviará logs correspondentes aos níveis WARN, ERROR e FATAL.

Registro em log avançado com Log4j2 e SLF4J

> **Note**
>
> O AWS Lambda não inclui o Log4j2 em seus runtimes gerenciados ou imagens de contêiner base. Portanto, estes não são afetados pelos problemas descritos em CVE-2021-44228, CVE-2021-45046 e CVE-2021-45105.
> Para casos em que uma função do cliente contém uma versão do Log4j2 impactada, aplicamos uma alteração aos runtimes gerenciados e imagens do contêiner base do Lambda Java que ajuda a mitigar os problemas em CVE-2021-44228, CVE-2021-45046 e CVE-2021-45105. Como resultado dessa alteração, os clientes que usam o Log4J2 podem ver mais uma entrada de log, semelhante a “`Transforming org/apache/logging/log4j/core/lookup/JndiLookup (java.net.URLClassLoader@...)`”. Todas as strings de log que referenciem o mapeador jndi na saída Log4J2 serão substituídas por “`Patched JndiLookup::lookup()`”.

Independentemente dessa mudança, é altamente recomendável que todos os clientes cujas funções incluam o Log4j2 atualizem para a versão mais recente. Especificamente, os clientes que usam a biblioteca aws-lambda-java-log4j2 em suas funções devem atualizar para a versão 1.5.0 (ou posterior) e reimplantar suas funções. Essa versão atualiza as dependências do utilitário Log4j2 subjacentes para a versão 2.17.0 (ou posterior). O binário aws-lambda-java-log4j2 atualizado está disponível no repositório do Maven, e seu código-fonte está disponível no Github.
Por fim, observe que qualquer biblioteca relacionada ao aws-lambda-java-log4j (v1.0.0 or 1.0.1) não deve ser usada em nenhuma circunstância. Essas bibliotecas estão relacionadas à versão 1.x do log4j, que foi encerrada em 2015. Não há suporte, manutenção nem patches para essas bibliotecas, e elas apresentam vulnerabilidades de segurança conhecidas.

Para personalizar a saída de log, oferecer suporte ao registro em log durante testes de unidade e registrar chamadas do AWS SDK em log, use o Apache Log4j2 com o SLF4J. O Log4j é uma biblioteca de registro em log para programas Java que permite configurar níveis de log e usar bibliotecas de appender. O SLF4J é uma biblioteca de fachada que permite alterar qual biblioteca você usa sem alterar o código da função.

Para adicionar o ID de solicitação aos logs da sua função, use o appender na biblioteca aws-lambda-java-log4j2.

Example src/main/resources/log4j2.xml: configuração do 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>
```

Você pode decidir como os logs do Log4j2 são configurados para saídas em texto simples ou JSON especificando um layout nas tags `<LambdaTextFormat>` e `<LambdaJSONFormat>`.

Neste exemplo, no modo de texto, cada linha é precedida com a data, a hora, o ID da solicitação, o nível do log e o nome da classe. No modo JSON, `<JsonTemplateLayout>` é usado com uma configuração que está incluída na biblioteca `aws-lambda-java-log4j2`.

O SLF4J é uma biblioteca de fachada para registro em log em código Java. No código da função, use o logger factory SLF4J para recuperar um logger com métodos para níveis de log como `info()` e `warn()`. Na configuração da compilação, inclua a biblioteca de registro em log e o adaptador SLF4J no classpath. Alterando as bibliotecas na configuração de compilação, é possível alterar o tipo de logger sem alterar o código da função. O SLF4J é necessário para capturar logs do SDK for Java.

No exemplo de código a seguir, a classe de manipulador usa SLF4J para recuperar um logger.

Example src/main/java/example/HandlerS3.java: registro em log com o 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";
    }
}
```

Esse código gera uma saída de log semelhante à seguinte:

Example formato do 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/"
}
```

A configuração de compilação usa dependências de runtime no appender e no adaptador SLF4J do Lambda, além de dependências de implementação no Log4J2.

Example build.gradle: dependências de registro em log

```
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 você executa seu código localmente para testes, o objeto de contexto com o logger do Lambda não está disponível e não há nenhum ID de solicitação para o appender do Lambda usar. Para obter exemplos de configurações de teste, consulte as aplicações de exemplo na próxima seção.

Outras ferramentas e bibliotecas

O Powertools para AWS Lambda (Java) é um kit de ferramentas para desenvolvedores para implementar as práticas recomendadas da tecnologia sem servidor e aumentar a velocidade do desenvolvedor. O utilitário de logs fornece um registrador de logs otimizado para Lambda que inclui informações adicionais sobre o contexto da função em todas as suas funções, com saída estruturada como JSON. Use esse utilitário para fazer o seguinte:

- Capturar campos-chave do contexto do Lambda, inicialização a frio e estruturas registrando em log a saída como JSON
- Registrar em log eventos de invocação do Lambda quando instruído (desativado por padrão)
- Imprimir todos os logs somente para uma porcentagem das invocações por meio da amostragem de logs (desativado por padrão)
- Anexar chaves adicionais ao log estruturado a qualquer momento
- Use um formatador de log personalizado (Bring Your Own Formatter) para gerar logs em uma estrutura compatível com o RFC de logs da sua organização

Uso do Powertools para AWS Lambda (Java) e do AWS SAM para registro em log estruturado

Siga as etapas abaixo para baixar, criar e implantar um exemplo da aplicação Hello World em Java com os módulos integrados do Powertools para AWS Lambda (Java)~ usando o AWS SAM. Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET ao endpoint do API Gateway, a função do Lambda invoca, envia logs e métricas usando o formato de métricas incorporadas ao CloudWatch e envia rastreamentos ao AWS X-Ray. A função retorna uma mensagem `hello world`.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Java 11
- AWS CLI versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS SAM

1. Inicialize a aplicação usando o modelo Java do Hello World.

```
sam init --app-template hello-world-powertools-java --name sam-app --package-type
 Zip --runtime java11 --no-tracing
```

2. Crie a aplicação.

```
cd sam-app && sam build
```

3. Implante o aplicativo.

```
sam deploy --guided
```

4. Siga as instruções na tela. Para aceitar as opções padrão fornecidas na experiência interativa, pressione `Enter`.

> **Note**
>
> Em HelloWorldFunction pode não ter autorização definida, tudo bem?, certifique-se de inserir y.

5. Obtenha o URL da aplicação implantada:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Invoque o endpoint da API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Se tiver êxito, você verá esta resposta:

```
{"message":"hello world"}
```

7. Para obter os logs da função, execute sam logs. Para obter mais informações, consulte Trabalhar com logs no Guia do desenvolvedor do AWS Serverless Application Model.

```
sam logs --stack-name sam-app
```

O resultado de saída do log se parece com:

```
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. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
sam delete
```

Gerenciar a retenção de logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs por tempo indeterminado, exclua o grupo de logs ou configure um período de retenção após o qual o CloudWatch excluirá os logs automaticamente. Para configurar a retenção de logs, adicione o seguinte ao seu modelo do 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
```

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda.

Se seu código puder ser testado no editor de Código incorporado, você encontrará logs nos resultados de execução. Ao usar o recurso de teste do console para invocar uma função, você encontrará Saída de log na seção Detalhes.

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a página Log groups (Grupos de log).
2. Escolha o grupo de logs de sua função (/aws/lambda/***nome-de-sua-função***).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma instância da sua função. Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos intrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- AWS CLI – Configuração rápida com `aws configure`

Você pode usar a AWS CLI para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

A seguinte saída deverá ser mostrada:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `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
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

A seguinte saída deverá ser mostrada:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e Ubuntu no Windows. Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa `sed` para remover as aspas do arquivo de saída e fica inativo por 15 segundos para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 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 macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod -R 755 get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
}
```

Excluir logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ouconfigurar um período de retençãoapós o qual os logs são excluídos automaticamente.

Código de exemplo de registro em log

O repositório do GitHub para este guia inclui aplicativos de exemplo que demonstram o uso de várias configurações de registro em log. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS SAM e recursos de suporte.

Aplicações de exemplo do Lambda em Java

- java17-examples: uma função em Java que demonstra como usar um registro Java para representar um objeto de dados de evento de entrada.
- java-basic: uma coleção de funções Java mínimas com testes de unidade e configuração de registro em log variável.
- java-events: uma coleção de funções do Java contendo código básico sobre como lidar com eventos de vários serviços, como o Amazon API Gateway, o Amazon SQS e o Amazon Kinesis. Essas funções usam a versão mais recente da biblioteca aws-lambda-java-events (3.0.0 e versões mais recentes). Estes exemplos não exigem oAWS SDK como dependência.
- s3-java: uma função em Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.
- Use API Gateway to invoke a Lambda function (Usar o API Gateway para invocar uma função do Lambda): uma função Java que verifica uma tabela do Amazon DynamoDB contendo informações de funcionários. Em seguida, usa o Amazon Simple Notification Service para enviar uma mensagem de texto aos funcionários comemorando seus aniversários de empresa. Este exemplo usa o API Gateway para invocar a função.

O aplicativo de exemplo `java-basic` mostra uma configuração mínima de registro em log que oferece suporte a testes de registro em log. O código do manipulador usa o logger `LambdaLogger` fornecido pelo objeto de contexto. Para testes, a aplicação usa uma classe `TestLogger` personalizada que implementa a interface `LambdaLogger` com um logger Log4j2. Ele usa o

SLF4J como fachada para compatibilidade com o AWS SDK. As bibliotecas de registro em log são excluídas da saída de compilação para manter o pacote de implantação pequeno.

Instrumentação do código Java no AWS Lambda

O Lambda se integra ao AWS X-Ray para ajudar você a rastrear, depurar e otimizar aplicações do Lambda. É possível usar o X-Ray para rastrear uma solicitação enquanto ela atravessa recursos na aplicação, o que pode incluir funções Lambda e outros produtos da AWS.

Para enviar dados de rastreamento ao X-Ray, você pode usar uma das duas bibliotecas SDK:

- AWS Distro for OpenTelemetry (ADOT): uma distribuição segura, pronta para produção e com suporte na AWS do SDK OpenTelemetry (OTel).
- SDK do AWS X-Ray for Java: um SDK para geração e envio de dados de rastreamento ao X-Ray.
- Powertools para AWS Lambda (Java): kit de ferramentas para desenvolvedores para implementar as práticas recomendadas da tecnologia sem servidor e aumentar a velocidade do desenvolvedor.

Cada um dos SDKs oferece maneiras de enviar dados de telemetria ao serviço do X-Ray. Em seguida, é possível usar o X-Ray para visualizar, filtrar e obter insights sobre as métricas de performance da aplicação para identificar problemas e oportunidades de otimização.

> **Important**
>
> Os SDKs do X-Ray e do Powertools para AWS Lambda fazem parte de uma solução de instrumentação totalmente integrada oferecida pela AWS. As camadas do Lambda para ADOT fazem parte de um padrão em todo o setor para instrumentação de rastreamento que coleta mais dados em geral, mas pode não ser adequado para todos os casos de uso. É possível implementar o rastreamento de ponta a ponta no X-Ray usando ambas as soluções. Para saber mais sobre como escolher entre elas, consulte Como escolher entre os SDKs do AWS Distro para OpenTelemetry e do X-Ray.

Seções

- Ativar o rastreamento com o AWS CloudFormation
- Interpretar um rastreamento do X-Ray
- Armazenar dependências de runtime em uma camada (SDK do X-Ray)
- Rastreamento do X-Ray em aplicações de exemplo (SDK do X-Ray)

Uso do Powertools para AWS Lambda (Java) e do AWS SAM para rastreamento

Siga as etapas abaixo para baixar, criar e implantar um exemplo da aplicação Hello World em Java com os módulos integrados do Powertools para AWS Lambda (Java) usando o AWS SAM. Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET ao endpoint do API Gateway, a função do Lambda invoca, envia logs e métricas usando o formato de métricas incorporadas ao CloudWatch e envia rastreamentos ao AWS X-Ray. A função retorna uma mensagem `hello world`.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Java 11
- AWS CLI versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS SAM

1. Inicialize a aplicação usando o modelo Java do Hello World.

```
sam init --app-template hello-world-powertools-java --name sam-app --package-type
 Zip --runtime java11 --no-tracing
```

2. Crie a aplicação.

```
cd sam-app && sam build
```

3. Implante o aplicativo.

```
sam deploy --guided
```

4. Siga as instruções na tela. Para aceitar as opções padrão fornecidas na experiência interativa, pressione `Enter`.

> **Note**
>
> Em HelloWorldFunction pode não ter autorização definida, tudo bem?, certifique-se de inserir `y`.

5. Obtenha o URL da aplicação implantada:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Invoque o endpoint da API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Se tiver êxito, você verá esta resposta:

```
{"message":"hello world"}
```

7. Para obter os rastreamentos da função, execute sam traces.

```
sam traces
```

A saída de rastreamento se parece com:

```
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. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
sam delete
```

Uso do Powertools para AWS Lambda (Java) e do AWS CDK para rastreamento

Siga as etapas abaixo para baixar, criar e implantar um exemplo da aplicação Hello World em Java com os módulos integrados do Powertools para AWS Lambda (Java) usando o AWS CDK. Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET ao endpoint do API Gateway, a função do Lambda invoca, envia logs e métricas usando o formato de métricas incorporadas ao CloudWatch e envia rastreamentos ao AWS X-Ray. A função retorna uma mensagem de hello world.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Java 11

- AWS CLI versão 2
- AWS CDK versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS CDK

1. Crie um diretório de projeto para sua aplicação.

```
mkdir hello-world
cd hello-world
```

2. Inicialize a aplicação.

```
cdk init app --language java
```

3. Crie o projeto maven com o seguinte comando:

```
mkdir app
cd app
mvn archetype:generate -DgroupId=helloworld -DartifactId=Function -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

4. Abra `pom.xml` no diretório `hello-world\app\Function` e substitua o código existente pelo código a seguir, que inclui dependências e plugins do maven para 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. Crie o diretório `hello-world\app\src\main\resource` e crie `log4j.xml` para a configuração do log.

```
mkdir -p src/main/resource
cd src/main/resource
touch log4j.xml
```

6. Abra `log4j.xml` e adicione o código a seguir.

```
<?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. Abra o `App.java` partir do diretório `hello-world\app\Function\src\main\java\helloworld` e substitua o código existente pelo código a seguir. Este é o código da função do 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. Abra o `HelloWorldStack.java` partir do diretório `hello-world\src\main\java\com\myorg` e substitua o código existente pelo código a seguir. Esse código usa o construtor Lambda e o construtor ApiGatewayv2 para criar uma API REST e uma função do 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. Abra o `pom.xml` partir do diretório `hello-world` e substitua o código existente pelo código a seguir.

```
<?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. Verifique se você está no diretório `hello-world` e implante a sua aplicação.

```
cdk deploy
```

11. Obtenha o URL da aplicação implantada:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?OutputKey==`HttpApi`].OutputValue' --output text
```

12. Invoque o endpoint da API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

    Se tiver êxito, você verá esta resposta:

```
{"message":"hello world"}
```

13. Para obter os rastreamentos da função, execute sam traces.

```
sam traces
```

A saída de rastreamento se parece com:

```
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. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
cdk destroy
```

Usar o ADOT para instrumentar funções do Java

O ADOT fornece camadas do Lambda totalmente gerenciadas que empacotam tudo o que você precisa para coletar dados de telemetria usando o SDK do OTel. Ao consumir essa camada, é possível instrumentar suas funções Lambda sem precisar modificar nenhum código de função. Você também pode configurar sua camada para fazer a inicialização personalizada do OTel. Para obter mais informações, consulte Custom configuration for the ADOT Collector on Lambda (Configuração personalizada para o ADOT Collector no Lambda) na documentação do ADOT.

Para runtimes do Java, é possível escolher entre duas camadas para consumir:

- Camada do Lambda gerenciada pela AWS para ADOT Java (agente de instrumentação automática): esta camada transforma automaticamente o código da função na inicialização para coletar dados de rastreamento. Para obter instruções detalhadas sobre como consumir essa camada junto com o agente Java do ADOT, consulte Suporte do AWS Distro for OpenTelemetry Lambda para Java (agente de instrumentação automática), na documentação do ADOT.
- Camada Lambda gerenciada pela AWS para ADOT Java: esta camada também fornece instrumentação integrada para funções Lambda, mas requer algumas alterações manuais de código para inicializar o SDK do OTel. Para obter instruções detalhadas sobre como consumir essa camada, consulte Suporte do AWS Distro for OpenTelemetry Lambda para Java, na documentação do ADOT.

Usar o SDK do X-Ray para instrumentar suas funções Java

Para registrar dados sobre chamadas que sua função faz para outros recursos e serviços em sua aplicação, adicione o X-Ray SDK for Java à sua configuração de compilação. O exemplo a seguir mostra uma configuração de compilação do Gradle que inclui as bibliotecas que ativam a instrumentação automática de clientes do AWS SDK for Java 2.x.

Example build.gradle: rastrear dependências

```
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'
    ...
}
```

Depois de adicionar as dependências corretas e fazer as devidas mudanças de código, ative o rastreamento na configuração da sua função usando o console do Lambda ou a API.

Ativar o rastreamento com o console do Lambda

Para alternar o rastreamento ativo na sua função do Lambda usando o console, siga as etapas abaixo:

Para ativar o rastreamento ativo

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e depois Monitoring and operations tools (Ferramentas de monitoramento e operações).
4. Selecione a opção Editar.
5. Em X-Ray, ative a opção Active tracing (Rastreamento ativo).
6. Escolha Salvar.

Ativar o rastreamento com a API do Lambda

Configure o rastreamento na sua função do Lambda com aAWS CLI ou o AWS SDK, usando as seguintes operações de API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration \
--function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Ativar o rastreamento com o AWS CloudFormation

Para ativar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example function-inline.yml: configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example template.yml: configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretar um rastreamento do X-Ray

Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você ativa o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à função

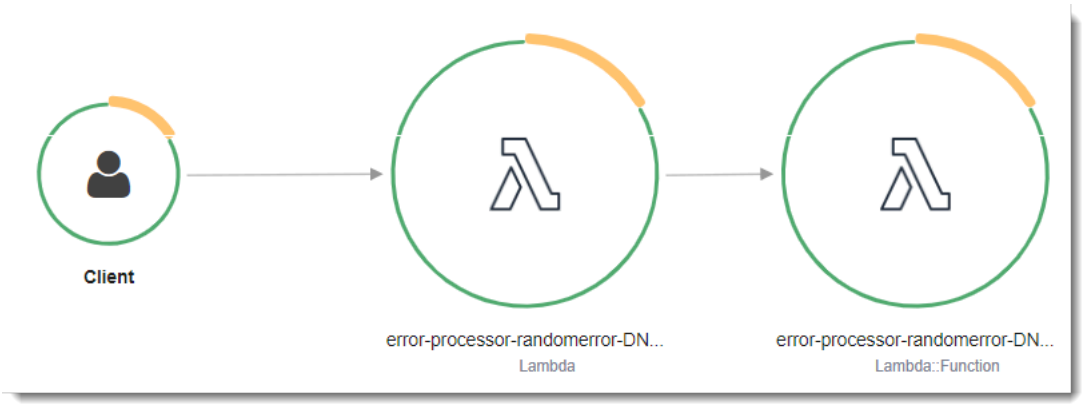
de execução da função. Caso contrário, adicione a política AWSXRayDaemonWriteAccess à função de execução.

Após configurar o rastreamento ativo, você pode observar solicitações específicas por meio da aplicação. O grafo de serviço do X-Ray exibe informações sobre sua aplicação e todos os componentes. A imagem a seguir demonstra uma aplicação com duas funções. A função principal processa eventos e, às vezes, retorna erros. A segunda função de cima para baixo processa erros que aparecem no primeiro grupo de logs e usa o AWS SDK para chamar o X-Ray, o Amazon Simple Storage Service (Amazon S3) e o Amazon CloudWatch Logs.

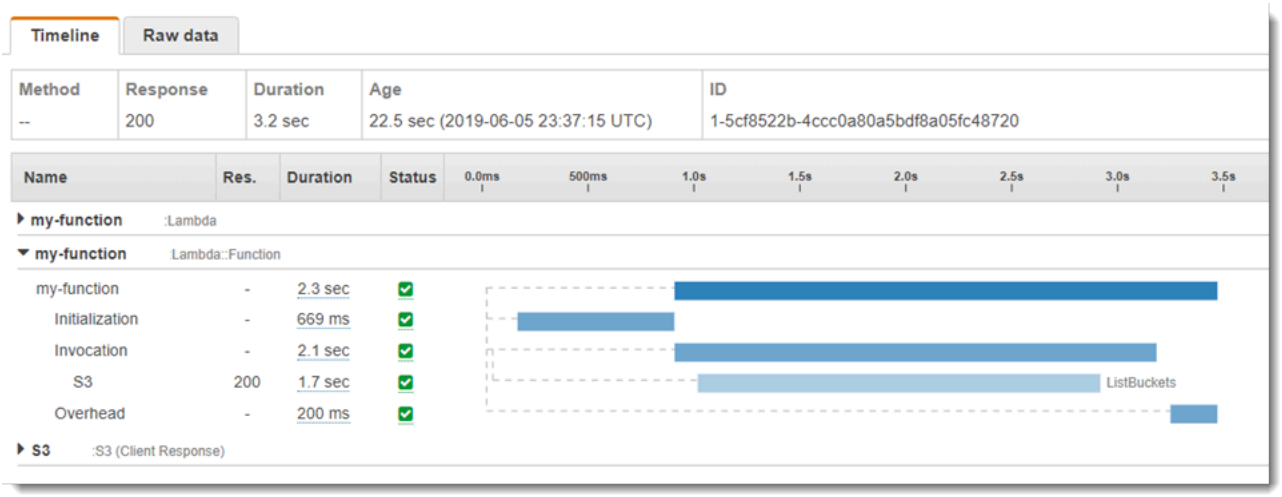
O X-Ray não rastreia todas as solicitações para sua aplicação. O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento seja eficiente, enquanto ainda fornece uma amostra representativa das solicitações. A taxa de amostragem é uma solicitação por segundo e 5% de solicitações adicionais.

> **Note**
>
> Não é possível configurar a taxa de amostragem do X-Ray para suas funções.

No X-Ray, um rastreamento registra informações sobre uma solicitação que é processada por um ou mais serviços. O Lambda registra dois segmentos por rastreamento, o que cria dois nós no gráfico de serviços. A imagem a seguir destaca esses dois nós:



O primeiro nó à esquerda representa o serviço do Lambda, que recebe a solicitação de invocação. O segundo nó representa a sua função do Lambda específica. O exemplo a seguir mostra um rastreamento com esses dois segmentos. Ambos têm o nome my-function, mas um tem a origem `AWS::Lambda` e o outro, a origem `AWS::Lambda::Function`. Se o segmento `AWS::Lambda` mostrar um erro, o serviço Lambda teve um problema. Se o segmento `AWS::Lambda::Function` mostrar um erro, sua função teve um problema.



Este exemplo expande o segmento `AWS::Lambda::Function` para mostrar seus três subsegmentos:

- Inicialização: representa o tempo gasto carregando a função e executando o código de inicialização. Esse subsegmento aparece somente para o primeiro evento que cada instância da função processa.
- Invocação: representa o tempo gasto na execução do código do manipulador.
- Sobrecarga: representa o tempo gasto pelo runtime do Lambda preparando-se para lidar com o próximo evento.

> **Note**
>
> As funções Lambda SnapStart também incluem um subsegmento `Restore`. O subsegmento `Restore` mostra o tempo necessário para o Lambda restaurar um snapshot, carregar o runtime (JVM) e executar qualquer hook de runtime do `afterRestore`. O processo de restauração de snapshots pode incluir o tempo gasto em atividades fora da MicroVM. Esse tempo é relatado no subsegmento `Restore`. Você não é cobrado pelo tempo gasto fora da microVM para restaurar um snapshot.

Você também pode instrumentar clientes HTTP, registrar consultas SQL e criar subsegmentos personalizados com anotações e metadados. Para obter mais informações, consulte SDK do AWS X-Ray for Java no Guia do desenvolvedor do AWS X-Ray.

> **Definição de preço**
>
> Você pode usar o rastreamento do X-Ray gratuitamente todos os meses até determinado limite como parte do nível gratuito da AWS. Além do limite, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter mais informações, consulte Preços do AWS X-Ray.

Armazenar dependências de runtime em uma camada (SDK do X-Ray)

Se você usar o X-Ray SDK para instrumentar os clientes do AWS SDK com seu código de função, seu pacote de implantação poderá se tornar bastante grande. Para evitar o upload de dependências de runtime todas as vezes que você atualizar seu código de função, empacote o SDK do X-Ray em uma camada do Lambda.

O exemplo a seguir mostra um recurso `AWS::Serverless::LayerVersion` que armazena o AWS SDK for Java e o X-Ray SDK for Java.

Example template.yml: camada de dependências

```
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
```

Com essa configuração, você atualizará a camada de biblioteca somente se alterar as dependências de runtime. Já que o pacote de implantação de função inclui apenas o seu código, isso pode ajudar a reduzir o tempo de upload.

A criação de uma camada de dependências requer alterações de configuração de compilação para gerar o arquivo de camada antes da implantação. Para obter um exemplo funcional, consulte a aplicação de exemplo java-basic no GitHub.

Rastreamento do X-Ray em aplicações de exemplo (SDK do X-Ray)

O repositório do GitHub para este guia inclui aplicações de exemplo que demonstram o uso do rastreamento do X-Ray. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS SAM e recursos de suporte.

Aplicações de exemplo do Lambda em Java

- java17-examples: uma função em Java que demonstra como usar um registro Java para representar um objeto de dados de evento de entrada.
- java-basic: uma coleção de funções Java mínimas com testes de unidade e configuração de registro em log variável.
- java-events: uma coleção de funções do Java contendo código básico sobre como lidar com eventos de vários serviços, como o Amazon API Gateway, o Amazon SQS e o Amazon Kinesis. Essas funções usam a versão mais recente da biblioteca aws-lambda-java-events (3.0.0 e versões mais recentes). Estes exemplos não exigem oAWS SDK como dependência.
- s3-java: uma função em Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.

- Use API Gateway to invoke a Lambda function (Usar o API Gateway para invocar uma função do Lambda): uma função Java que verifica uma tabela do Amazon DynamoDB contendo informações de funcionários. Em seguida, usa o Amazon Simple Notification Service para enviar uma mensagem de texto aos funcionários comemorando seus aniversários de empresa. Este exemplo usa o API Gateway para invocar a função.

Todas as aplicações de exemplo têm rastreamento ativo habilitado para funções do Lambda. Por exemplo, a aplicação `s3-java` mostra instrumentação automática de clientes AWS SDK for Java 2.x, gerenciamento de segmentos para testes, subsegmentos personalizados e o uso de camadas do Lambda para armazenar dependências de runtime.

Aplicativos de exemplo Java para o AWS Lambda

O repositório do GitHub para este guia inclui aplicativos de exemplo que demonstram o uso do Java no AWS Lambda. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS CloudFormation e recursos de suporte.

Aplicações de exemplo do Lambda em Java

- java17-examples: uma função em Java que demonstra como usar um registro Java para representar um objeto de dados de evento de entrada.
- java-basic: uma coleção de funções Java mínimas com testes de unidade e configuração de registro em log variável.
- java-events: uma coleção de funções do Java contendo código básico sobre como lidar com eventos de vários serviços, como o Amazon API Gateway, o Amazon SQS e o Amazon Kinesis. Essas funções usam a versão mais recente da biblioteca aws-lambda-java-events (3.0.0 e versões mais recentes). Estes exemplos não exigem oAWS SDK como dependência.
- s3-java: uma função em Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.
- Use API Gateway to invoke a Lambda function (Usar o API Gateway para invocar uma função do Lambda): uma função Java que verifica uma tabela do Amazon DynamoDB contendo informações de funcionários. Em seguida, usa o Amazon Simple Notification Service para enviar uma mensagem de texto aos funcionários comemorando seus aniversários de empresa. Este exemplo usa o API Gateway para invocar a função.

Executar estruturas Java populares no Lambda

- spring-cloud-function-samples: um exemplo da Spring que mostra como usar a estrutura Spring Cloud Function para criar funções do AWS Lambda.
- Demonstração da aplicação Spring Boot sem servidor: um exemplo que mostra como configurar uma aplicação Spring Boot típica em um runtime Java gerenciado com e sem o SnapStart, ou como uma imagem nativa do GraalVM com um runtime personalizado.
- Demonstração da aplicação Micronaut sem servidor: um exemplo que mostra como usar o Micronaut em um runtime Java gerenciado com e sem o SnapStart, ou como uma imagem nativa do GraalVM com um runtime personalizado. Saiba mais nos guias do Micronaut/Lambda.
- Demonstração da aplicação Quarkus sem servidor: um exemplo que mostra como usar o Quarkus em um runtime Java gerenciado com e sem o SnapStart, ou como uma imagem nativa do GraalVM

com um runtime personalizado. Saiba mais no guia do Quarkus/Lambda e no guia do Quarkus/SnapStart.

Se você ainda é iniciante com as funções do Lambda em Java, comece com os exemplos de `java-basic`. Para começar a usar as origens de eventos do Lambda, consulte os exemplos de `java-events`. Os dois exemplos mostram o uso das bibliotecas Java do Lambda, das variáveis de ambiente, do AWS SDK e do SDK do AWS X-Ray. Estes exemplos requerem configuração mínima e podem ser implantados pela linha de comando em menos de um minuto.

Criar funções do Lambda com Go

O Go é implementado de forma diferente de outros runtimes gerenciados. Como o Go é compilado nativamente com um binário executável, ele não requer um runtime de linguagem dedicado. Use um runtime somente de sistema operacional (a família de runtime `provided`) para implantar funções do Go no Lambda.

Tópicos

Suporte do runtime do Go

O runtime gerenciado do Go 1.x para Lambda foi descontinuado. Se tiver funções que usem o runtime do Go 1.x, você deverá migrar suas funções para `provided.al2023` ou `provided.al2`. Os runtimes `provided.al2023` e `provided.al2` oferecem várias vantagens em comparação ao `go1.x`, incluindo compatibilidade com a arquitetura arm64 (processadores AWS Graviton2), binários menores e tempos de invocação um pouco mais rápidos.

Nenhuma alteração de código é necessária para essa migração. As únicas alterações necessárias estão relacionadas à forma como você cria seu pacote de implantação e ao runtime que você usa para criar sua função. Para obter mais informações, consulte Migrating AWS Lambda functions from the Go1.x runtime to the custom runtime on Amazon Linux 2 no AWS Compute Blog.

Somente SO

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
Runtime somente para sistema operacional	`provided.al2023`	Amazon Linux 2023			
Runtime somente para sistema operacional	`provided.al2`	Amazon Linux 2			

Ferramentas e bibliotecas

O Lambda fornece as seguintes ferramentas e bibliotecas para o runtime do Go:

- AWS SDK for Go: o SDK oficial da AWS para a linguagem de programação Go.
- github.com/aws/aws-lambda-go/lambda: a implementação do modelo de programação do Lambda para Go. Esse pacote é usado pelo AWS Lambda para invocar o handler.
- github.com/aws/aws-lambda-go/lambdacontext: auxiliares para acesso a informações do objeto de contexto.
- github.com/aws/aws-lambda-go/events: esta biblioteca fornece definições de tipos para integrações comuns de origens de eventos.
- github.com/aws/aws-lambda-go/cmd/build-lambda-zip: Esta ferramenta pode ser usada para criar um archive com arquivo .zip no Windows.

Para obter mais informações, consulte aws-lambda-go no GitHub.

O Lambda fornece as seguintes aplicações de exemplo para o runtime do Go:

Aplicativos do Lambda de exemplo do em Go

- go-al2: uma função olá, mundo que retorna o endereço IP público. Esta aplicação usa o runtime `provided.al2` personalizado.
- blank-go: uma função do Go que mostra o uso das bibliotecas do Go do Lambda, o registro em log, as variáveis de ambiente e o AWS SDK. Esta aplicação usa o runtime `go1.x`.

Definir o manipulador de função do Lambda em Go

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. A função é executada até que o manipulador retorne uma resposta, seja encerrado ou atinja o tempo limite.

Uma função do Lambda escrita em Go é criada como um executável do Go. No código da função do Lambda, você precisa incluir o pacote github.com/aws/aws-lambda-go/lambda que implementa o modelo de programação do Lambda para Go. Além disso, você precisa implementar o código da função do manipulador e uma função `main()`.

Example Função do Lambda em Go

```
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)
}
```

Um exemplo de entrada para esta função é:

```
{
  "name": "Jane"
```

```
}
```

Observe o seguinte:

- pacote main: no Go, o pacote que contém `func main()` sempre deve ser denominado `main`.
- import: use essa opção para incluir as bibliotecas necessárias para a função do Lambda. Neste caso, ela inclui:
  - context: Objeto de contexto do AWS Lambda em Go.
  - fmt: o objeto de Formatação do Go usado para formatar o valor de retorno da função.
  - github.com/aws/aws-lambda-go/lambda: conforme mencionado anteriormente, implementa o modelo de programação do Lambda para Go.
- func HandleRequest(ctx context.Context, event *MyEvent) (*string, error): esta é a assinatura do manipulador do Lambda. É o ponto de entrada para sua função do Lambda e contém a lógica que é executada quando sua função é invocada. Além disso, os parâmetros incluídos indicam o seguinte:
  - ctx context.Context: fornece informações do runtime da invocação da função do Lambda. `ctx` é a variável que você declara para utilizar as informações disponíveis por meio do Objeto de contexto do AWS Lambda em Go.
  - event *MyEvent: um parâmetro chamado `event` que aponta para `MyEvent`. Ela representa a entrada para a função do Lambda.
  - *string, error: o manipulador retorna dois valores. O primeiro é um ponteiro para uma string que contém o resultado da função do Lambda. O segundo é um tipo de erro, que é `nil` quando não há erro e contém informações de erro padrão se algo errado ocorre.
  - return &message, nil: retorna dois valores. O primeiro é um ponteiro para uma mensagem de string que consiste em uma saudação construída usando o campo `Name` do evento de entrada. O segundo valor, `nil`, indica que a função não encontrou nenhum erro.
- func main(): o ponto de entrada que executa o código da função do Lambda. Isso é obrigatório.

  Com a adição de `lambda.Start(HandleRequest)` entre colchetes do código `func main() {}`, a função do Lambda será executada. De acordo com os padrões da linguagem Go, o colchete de abertura, `{` deve ser colocado diretamente no final da assinatura da função `main`.

Nomenclatura

Runtimes provided.al2 e provided.al2023

Para funções Go que usam o runtime `provided.al2` ou `provided.al2023` em um pacote de implantação .zip, o arquivo executável que contém o código da função deve ser denominado `bootstrap`. Se você estiver implantando a função com um arquivo .zip, o arquivo `bootstrap` deverá estar na raiz do arquivo .zip. Para funções Go que usam o runtime `provided.al2` ou`provided.al2023` em uma imagem de contêiner, você pode usar qualquer nome para o arquivo executável.

Você pode usar qualquer nome para o manipulador. Para referenciar o valor do manipulador em seu código, você pode usar a variável de ambiente `_HANDLER`.

runtime do go1.x

Para funções Go que usam o runtime `go1.x`, o arquivo executável e o manipulador podem compartilhar qualquer nome. Por exemplo, se você definir o valor do manipulador como `Handler`, o Lambda chamará a função `main()` no arquivo executável `Handler`.

Para alterar o nome do manipulador de função no console do Lambda no painel de Configurações de runtime, escolha Editar.

Manipulador de função do Lambda usando tipos estruturados

No exemplo acima, o tipo de entrada era uma sequência simples. Mas você também pode passar eventos estruturados para o manipulador da função:

```
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)
}
```

Um exemplo de entrada para esta função é:

```
{
    "What is your name?": "Jim",
    "How old are you?": 33
}
```

A resposta é semelhante a:

```
{
    "Answer": "Jim is 33 years old!"
}
```

Para serem exportados, os nomes do campo no struct do evento devem estar em letra maiúscula. Para obter mais informações sobre a manipulação de eventos de fontes de eventos da AWS, consulte aws-lambda-go/events.

Assinaturas válidas de manipulador

Há várias opções ao criar um manipulador de função do Lambda no Go, mas você deve observar as seguintes regras:

- O manipulador deve ser uma função.
- O manipulador pode usar de 0 a 2 argumentos. Se houver dois argumentos, o primeiro argumento deverá implementar `context.Context`.

- O manipulador pode retornar de 0 a 2 argumentos. Se houver um único valor de retorno, ele deverá implementar `error`. Se houver dois valores de retorno, o segundo valor deverá implementar `error`.

As assinaturas válidas de manipulador são listadas a seguir. `TIn` e `TOut` representam tipos compatíveis com a biblioteca encoding/json padrão. Para obter mais informações, consulte func Unmarshal para saber como esses tipos são desserializados.

- `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)`

Usar o estado global

Você pode declarar e modificar variáveis globais que são independentes do código do manipulador da função do Lambda. Além disso, o manipulador pode declarar uma função `init` que é executada quando o manipulador é carregado. Esse comportamento é o mesmo no AWS Lambda e nos programas Go padrão. Uma única instância de sua função do Lambda nunca manipulará vários eventos simultaneamente.

Example Função Go com variáveis globais

> **Note**
>
> Esse código usa o AWS SDK for Go V2. Para obter mais informações, consulte Conceitos básicos do 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)
}
```

Objeto de contexto do AWS Lambda em Go

Quando o Lambda executa a função, ele transmite um objeto de contexto para o manipulador. Esse objeto fornece métodos e propriedades com informações sobre a invocação, a função e o ambiente de execução.

A biblioteca de contexto do Lambda fornece as seguintes variáveis globais, métodos e propriedades.

Variáveis globais

- `FunctionName`: o nome da função do Lambda.
- `FunctionVersion`: a versão da função.
- `MemoryLimitInMB`: a quantidade de memória alocada para a função.
- `LogGroupName`: o grupo de logs da função.
- `LogStreamName`: a transmissão de log para a instância da função.

Métodos de contexto

- `Deadline`— Retorna a data em que a execução expira o tempo em milissegundos do Unix.

Propriedades de contexto

- `InvokedFunctionArn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `AwsRequestID`: o identificador da solicitação de invocação.
- `Identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
- `ClientContext`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.

Acessar informações do contexto de invocação

As funções do Lambda têm acesso aos metadados sobre seu ambiente e a solicitação da invocação. Isso pode ser acessado no Contexto do pacote. Se o manipulador incluir `context.Context` como um parâmetro, o Lambda inserirá informações sobre sua função na propriedade `Value` do contexto.

Observe que você precisa importar a biblioteca `lambdacontext` para acessar o conteúdo do objeto `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)
}
```

No exemplo acima, `lc` é a variável usada para consumir as informações que o objeto de contexto capturou e `log.Print(lc.Identity.CognitoIdentityPoolID)` imprimir essas informações, nesse caso, o CognitoIdentityPool ID.

O exemplo a seguir mostra como usar o objeto de contexto para monitorar o tempo necessário para concluir a função do Lambda. Isso permite que você analise as expectativas de performance e ajuste seu código de função de maneira correspondente, se necessário.

```
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)
}
```

Implantar funções do Lambda em Go com arquivos .zip

O código da função do AWS Lambda consiste em scripts ou programas compilados e as dependências deles. Você usa umpacote de implantaçãopara implantar seu código de função no Lambda. O Lambda é compatível com dois tipos de pacotes de implantação: imagens de contêiner e arquivos .zip.

Esta página descreve como criar um arquivo .zip como seu pacote de implantação do runtime do Go e, em seguida, usar o arquivo para implantar o código de sua função no AWS Lambda utilizando o AWS Management Console, o AWS Command Line Interface, a (AWS CLI) e o AWS Serverless Application Model (AWS SAM).

Observe que o Lambda usa permissões de arquivo POSIX, então pode ser necessário definir permissões para a pasta do pacote de implantação antes da criação do arquivo .zip.

Seções

Criando um arquivo .zip no macOS e no Linux

As etapas a seguir mostram como compilar o executável usando o comando `go build` e criar um pacote de implantação do arquivo .zip para o Lambda. Antes de compilar seu código, verifique se você instalou o pacote lambda do GitHub. Este módulo fornece uma implementação da interface de runtime, que gerencia a interação entre o Lambda e seu código da função. Para baixar essa biblioteca, execute o comando a seguir.

```
go get github.com/aws/aws-lambda-go/lambda
```

Se sua função usa o AWS SDK for Go, baixe o conjunto padrão de módulos do SDK e qualquer cliente de API de serviço da AWS exigido pela sua aplicação. Para saber como instalar o SDK para Go, consulte Conceitos básicos do AWS SDK for Go V2.

Usar o runtime da família fornecido

O Go é implementado de forma diferente de outros runtimes gerenciados. Como o Go é compilado nativamente com um binário executável, ele não requer um runtime de linguagem dedicado. Use um runtime somente de sistema operacional (a família de runtime `provided`) para implantar funções do Go no Lambda.

Para criar um pacote de implantação de arquivo .zip (macOs/Linux)

1. No diretório do projeto que contém o arquivo `main.go` da sua aplicação, compile seu executável. Observe o seguinte:

   - O executável deve ser nomeado `bootstrap`. Para ter mais informações, consulte Nomenclatura.
   - Defina sua arquitetura do conjunto de instruções de destino. Os runtimes somente de SO são compatíveis com arm64 e x86_64.
   - Você pode usar a tag `lambda.norpc` opcional para excluir o componente Remote Procedure Call (RPC) da biblioteca do Lambda. O componente RPC só será necessário se você estiver usando o runtime do Go 1.x. descontinuado. A exclusão do RPC reduz o tamanho do pacote de implantação.

   Para a arquitetura arm64:

   ```
   GOOS=linux GOARCH=arm64 go build -tags lambda.norpc -o bootstrap main.go
   ```

   Para a arquitetura x86_64:

   ```
   GOOS=linux GOARCH=amd64 go build -tags lambda.norpc -o bootstrap main.go
   ```

2. (Opcional) Talvez seja necessário compilar pacotes com o `CGO_ENABLED=0` configurado no Linux:

   ```
   GOOS=linux GOARCH=arm64 CGO_ENABLED=0 go build -o bootstrap -tags lambda.norpc
    main.go
   ```

   Este comando cria um pacote binário estável para versões padrão da biblioteca C (`libc`), que podem ser diferentes no Lambda e em outros dispositivos.
3. Crie um pacote de implantação empacotando o executável em um arquivo .zip.

```
zip myFunction.zip bootstrap
```

> **Note**
>
> O arquivo `bootstrap` deve estar na raiz do arquivo .zip.

4. Criar a função do Observe o seguinte:

   - O binário deve ser chamado de `bootstrap`, mas o nome do manipulador pode ser qualquer coisa. Para ter mais informações, consulte Nomenclatura.
   - A opção `--architectures` será necessária apenas se você estiver usando arm64. O valor padrão é x86_64.
   - Para `--role`, especifique o nome do recurso da Amazon (ARN) da função de execução.

```
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
```

Criando um arquivo .zip no Windows

As etapas a seguir mostram como baixar a ferramenta build-lambda-zip para Windows do GitHub, compilar seu executável e criar um pacote de implantação .zip.

> **Note**
>
> Se ainda não tiver feito isso, você precisará instalar o Git e, em seguida, adicionar o executável `git` à variável de ambiente `%PATH%` do Windows.

Antes de compilar o código, verifique se você instalou a biblioteca lambda do GitHub. Para baixar essa biblioteca, execute o comando a seguir.

```
go get github.com/aws/aws-lambda-go/lambda
```

Se sua função usa o AWS SDK for Go, baixe o conjunto padrão de módulos do SDK e qualquer cliente de API de serviço da AWS exigido pela sua aplicação. Para saber como instalar o SDK para Go, consulte Conceitos básicos do AWS SDK for Go V2.

Usar o runtime da família fornecido

O Go é implementado de forma diferente de outros runtimes gerenciados. Como o Go é compilado nativamente com um binário executável, ele não requer um runtime de linguagem dedicado. Use um runtime somente de sistema operacional (a família de runtime `provided`) para implantar funções do Go no Lambda.

Para criar um pacote de implantação .zip (Windows)

1. Faça download da ferramenta build-lambda-zip do GitHub.

   ```
   go install github.com/aws/aws-lambda-go/cmd/build-lambda-zip@latest
   ```

2. Use a ferramenta do seu `GOPATH` para criar um arquivo .zip. Se você tiver uma instalação padrão do Go, a ferramenta geralmente estará em `%USERPROFILE%\Go\bin`. Caso contrário, navegue até o local em que o runtime do Go foi instalado e faça o seguinte:

   cmd.exe

   Em cmd.exe, execute uma das seguintes ações, dependendo da sua arquitetura do conjunto de instruções de destino. Os runtimes somente de SO são compatíveis com arm64 e x86_64.

   Você pode usar a tag `lambda.norpc` opcional para excluir o componente Remote Procedure Call (RPC) da biblioteca do Lambda. O componente RPC só será necessário se você estiver usando o runtime do Go 1.x. descontinuado. A exclusão do RPC reduz o tamanho do pacote de implantação.

   Example : para a arquitetura 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 : para a arquitetura 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

No PowerShell, execute uma das seguintes ações, dependendo da sua arquitetura do conjunto de instruções de destino. Os runtimes somente de SO são compatíveis com arm64 e x86_64.

Você pode usar a tag `lambda.norpc` opcional para excluir o componente Remote Procedure Call (RPC) da biblioteca do Lambda. O componente RPC só será necessário se você estiver usando o runtime do Go 1.x. descontinuado. A exclusão do RPC reduz o tamanho do pacote de implantação.

Para a arquitetura 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
```

Para a arquitetura 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. Criar a função do Observe o seguinte:

- O binário deve ser chamado de `bootstrap`, mas o nome do manipulador pode ser qualquer coisa. Para ter mais informações, consulte Nomenclatura.

- A opção `--architectures` será necessária apenas se você estiver usando arm64. O valor padrão é x86_64.
- Para `--role`, especifique o nome do recurso da Amazon (ARN) da função de execução.

```
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
```

Criar e atualizar funções do Lambda em Go usando arquivos .zip

Depois de criar seu pacote de implantação .zip, você poderá usá-lo para criar uma nova função do Lambda ou atualizar uma existente. É possível implantar o pacote .zip usando o console do Lambda, a AWS Command Line Interface e a API do Lambda. Você também pode criar e atualizar funções do Lambda usando o AWS Serverless Application Model (AWS SAM) e o AWS CloudFormation.

O tamanho máximo de um pacote de implantação .zip para o Lambda é 250 MB (descompactado). Esse limite se aplica ao tamanho combinado de todos os arquivos que você carrega, inclusive qualquer camada do Lambda.

O runtime do Lambda precisa de permissão para ler os arquivos no pacote de implantação. Na notação octal de permissões do Linux, o Lambda precisa de 644 permissões para arquivos não executáveis (rw-r--r--) e 755 permissões (rwxr-xr-x) para diretórios e arquivos executáveis.

No Linux e no MacOS, use o comando `chmod` para alterar as permissões de arquivo em arquivos e diretórios do seu pacote de implantação. Por exemplo, para dar a um arquivo executável as permissões corretas, execute o comando a seguir.

```
chmod 755 <filepath>
```

Para alterar as permissões de arquivo no Windows, consulte Set, View, Change, or Remove Permissions on an Object na documentação do Microsoft Windows.

Criar e atualizar funções com arquivos .zip usando o console

Para criar uma nova função, você deve primeiro criar a função no console e depois carregar o arquivo .zip. Para atualizar uma função existente, abra a página da função e siga o mesmo procedimento para adicionar o arquivo .zip atualizado.

Se o arquivo .zip for menor que 50 MB, você poderá criar ou atualizar uma função carregando o arquivo diretamente da máquina local. Para arquivos .zip maiores que 50 MB, você deve primeiro carregar o pacote para um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando o AWS Management Console, consulte Conceitos básicos do Amazon S3. Para carregar arquivos usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

> **Note**
>
> Não é possível converter uma função de imagem de contêiner existente para usar um arquivo .zip. É necessário criar uma nova função.

Para criar uma função (console)

1. Abra a página Funções do console do Lambda e escolha Criar função.
2. Escolha Author from scratch (Criar do zero).
3. Em Basic information (Informações básicas), faça o seguinte:
   a. Em Nome da função, insira o nome da função.
   b. Em Runtime (Tempo de execução), escolha `provided.al2023`.
4. (Opcional) Em Permissões, expanda Alterar função de execução padrão. Crie uma função de execução ou use uma existente.
5. Escolha a opção Criar função. O Lambda cria uma função básica “Hello world” usando o runtime escolhido.

Você pode carregar o arquivo .zip da máquina local (console)

1. Na página Funções do console Lambda, escolha a função para a qual você deseja carregar o arquivo .zip.
2. Selecione a guia Código.

3. No painel do Código-fonte, escolha Carregar de.
4. Escolha o arquivo .zip.
5. Para carregar o arquivo .zip, faça o seguinte:

    a. Selecione Carregar e, em seguida, selecione o arquivo .zip no seletor de arquivos.
    b. Escolha Open (Abrir).
    c. Escolha Salvar.

Para carregar um arquivo .zip de um bucket do Amazon S3 (console)

1. Na página Funções do console do Lambda, escolha a função para a qual você deseja carregar um novo arquivo .zip.
2. Selecione a guia Código.
3. No painel do Código-fonte, escolha Carregar de.
4. Escolha Local do Amazon S3.
5. Cole o URL do link do Amazon S3 do arquivo .zip e escolha Salvar.

Criar e atualizar funções com arquivos .zip usando a AWS CLI

Você pode usar a AWS CLI para criar uma função ou atualizar uma existente usando um arquivo .zip. Use os comandos create-function e update-function-code para implantar o pacote .zip. Se o arquivo .zip for menor que 50 MB, você poderá carregar o pacote .zip de um local do arquivo na máquina de compilação local. Para arquivos .zip maiores, você deve carregar o pacote .zip de um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

> **Note**
>
> Se você carregar o arquivo .zip de um bucket do Amazon S3 usando a AWS CLI, o bucket deverá estar na mesma Região da AWS que sua função.

Para criar uma função usando um arquivo .zip com a AWS CLI, você deve especificar o seguinte:

- O nome da função (`--function-name`)
- O runtime da função (`--runtime`)

- O nome do recurso da Amazon (ARN) da função de execução da função (`--role`)
- O nome do método do manipulador no código da função (`--handler`)

Você também deve especificar a local do arquivo .zip. Se o arquivo .zip estiver localizado em uma pasta da máquina de compilação local, use a opção `--zip-file` para especificar o caminho do arquivo, conforme mostrado no comando do exemplo a seguir.

```
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
```

Para especificar o local do arquivo .zip em um bucket do Amazon S3, use a opção `--code` conforme mostrado no comando do exemplo a seguir. Você só precisa usar o parâmetro `S3ObjectVersion` para objetos com versionamento.

```
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
```

Para atualizar uma função existente usando a CLI, especifique o nome da função usando o parâmetro `--function-name`. Você também deve especificar o local do arquivo .zip que deseja usar para atualizar o código da função. Se o arquivo .zip estiver localizado em uma pasta da máquina de compilação local, use a opção `--zip-file` para especificar o caminho do arquivo, conforme mostrado no comando do exemplo a seguir.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Para especificar o local do arquivo .zip em um bucket do Amazon S3, use as opções `--s3-bucket` e `--s3-key` conforme mostrado no comando do exemplo a seguir. Você só precisa usar o parâmetro `--s3-object-version` para objetos com versionamento.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket DOC-EXAMPLE-BUCKET --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Criar e atualizar funções com arquivos .zip usando a API do Lamba

Para criar e atualizar funções usando um arquivo .zip, use as seguintes operações de API:

- CreateFunction
- UpdateFunctionCode

Criar e atualizar funções com arquivos .zip usando o AWS SAM

O AWS Serverless Application Model (AWS SAM) é um kit de ferramentas que ajuda a simplificar o processo de criação e execução de aplicações com tecnologia sem servidor na AWS. Você define os recursos para a aplicação em um modelo YAML ou JSON e usa a interface da linha de comando do AWS SAM (CLI do AWS SAM) para criar, empacotar e implantar aplicações. Quando você cria uma função do Lambda com base em um modelo do AWS SAM, o AWS SAM cria automaticamente um pacote de implantação .zip ou uma imagem de contêiner com o código da função e quaisquer dependências que você especificar. Para saber mais sobre como usar o AWS SAM para criar e implantar funções do Lambda, consulte Conceitos básicos do AWS SAM no Guia do desenvolvedor do AWS Serverless Application Model.

Você também pode usar o AWS SAM para criar uma função do Lambda usando um arquivo .zip existente. Para criar uma função do Lambda usando o AWS SAM, salve o arquivo .zip em um bucket do Amazon S3 ou em uma pasta local na máquina de compilação. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

No modelo do AWS SAM, o recurso `AWS::Serverless::Function` especifica a função do Lambda. Nesse recurso, defina as seguintes propriedades para criar uma função usando um arquivo .zip:

- `PackageType`: definir como `Zip`
- `CodeUri`: definir como o URI do Amazon S3 do código da função, o caminho para a pasta local ou o objeto FunctionCode
- `Runtime`: definir como o runtime escolhido

Com o AWS SAM, se o arquivo .zip for maior que 50 MB, você não precisará carregá-lo primeiro em um bucket do Amazon S3. O AWS SAM poderá carregar pacotes .zip com o tamanho máximo permitido de 250 MB (descompactados) de um local da máquina de compilação local.

Para saber mais sobre a implantação de funções usando o arquivo .zip no AWS SAM, consulte AWS::Serverless::Function no Guia do desenvolvedor do AWS SAM.

Exemplo: usar o AWS SAM para criar uma função do Go com provided.al2023

1. Crie um modelo do AWS SAM com as seguintes propriedades:

   - BuildMethod: especifica o compilador para sua aplicação. Usar `go1.x`.
   - Runtime: use `provided.al2023`.
   - CodeUri: insira o caminho para seu código.
   - Arquiteturas: use `[arm64]` para a arquitetura arm64. Para a arquitetura do conjunto de instruções x86_64, use `[amd64]` ou remova a propriedade `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. Use o comando sam build para compilar o executável.

   ```
   sam build
   ```

3. Use o comando sam deploy para implantar a função no Lambda.

   ```
   sam deploy --guided
   ```

Criar e atualizar funções com arquivos .zip usando o AWS CloudFormation

Você pode usar o AWS CloudFormation para criar uma função do Lambda usando um arquivo .zip. Para criar uma função do Lambda de um arquivo.zip, primeiro carregue o arquivo em um bucket do Amazon S3. Para obter instruções sobre como carregar um arquivo para um bucket do Amazon S3 usando a AWS CLI, consulte Mover objetos no Guia do usuário da AWS CLI.

No modelo do AWS CloudFormation, o recurso `AWS::Lambda::Function` especifica a função do Lambda. Nesse recurso, defina as seguintes propriedades para criar uma função usando um arquivo .zip:

- `PackageType`: definir como `Zip`
- `Code`: inserir o nome do bucket do Amazon S3 e o nome do arquivo .zip nos campos `S3Bucket` e `S3Key`
- `Runtime`: definir como o runtime escolhido

O arquivo .zip que o AWS CloudFormation gera não pode exceder 4 MB. Para saber mais sobre a implantação de funções usando o arquivo .zip no AWS CloudFormation, consulte AWS::Lambda::Function no Guia do desenvolvedor do AWS CloudFormation.

Criar uma camada Go para suas dependências

> **Note**
>
> Usar camadas com funções em uma linguagem compilada, como Go, pode não oferecer o mesmo benefício de uma linguagem interpretada como Python. Como o Go é uma linguagem compilada, suas funções ainda precisam carregar manualmente quaisquer montagens compartilhadas na memória durante a fase inicial, o que pode aumentar os tempos de inicialização a frio. Em vez disso, recomendamos incluir qualquer código compartilhado no momento da compilação para aproveitar as otimizações integradas do compilador.

As instruções nesta seção mostram como incluir suas dependências em uma camada.

O Lambda detecta automaticamente todas as bibliotecas no diretório `/opt/lib` e quaisquer binários no diretório `/opt/bin`. Para garantir que o Lambda encontre corretamente o conteúdo da sua camada, crie uma camada com a seguinte estrutura:

```
custom-layer.zip
```

```
# lib
    | lib_1
    | lib_2
# bin
    | bin_1
    | bin_2
```

Depois de empacotar sua camada, consulte the section called “Criar e excluir camadas” e the section called “Adicionar camadas” para concluir sua configuração de camada.

Implantar funções do Lambda em Go com imagens de contêiner

Existem duas maneiras de criar uma imagem de contêiner para uma função do Lambda do Go:

- Usar uma imagem base somente para sistema operacional da AWS

  O Go é implementado de forma diferente de outros runtimes gerenciados. Como o Go é compilado nativamente com um binário executável, ele não requer um runtime de linguagem dedicado. Use uma imagem de base somente do sistema operacional para criar imagens do Go para o Lambda. Para tornar a imagem compatível com o Lambda, você deve incluir o pacote `aws-lambda-go/lambda` na imagem.
- Usar uma imagem base que não é da AWS

  Você também pode usar uma imagem base alternativa de outro registro de contêiner, como Alpine Linux ou Debian. Você também pode usar uma imagem personalizada criada por sua organização. Para tornar a imagem compatível com o Lambda, você deve incluir o pacote `aws-lambda-go/lambda` na imagem.

> **Tip**
>
> Para reduzir o tempo necessário para que as funções do contêiner do Lambda se tornem ativas, consulte Use multi-stage builds na documentação do Docker. Para criar imagens de contêiner eficientes, siga as Melhores práticas para gravar Dockerfiles.

Esta página explica como criar, testar e implantar imagens de contêiner para o Lambda.

Imagens de base da AWS para implantar funções do Go

O Go é implementado de forma diferente de outros runtimes gerenciados. Como o Go é compilado nativamente com um binário executável, ele não requer um runtime de linguagem dedicado. Use uma imagem de base somente do sistema operacional para implantar funções do Go no Lambda.

Somente SO

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
Runtime somente para sistema operacional	`provided.al2023`	Amazon Linux 2023			
Runtime somente para sistema operacional	`provided.al2`	Amazon Linux 2			

Galeria pública do Amazon Elastic Container Registry: gallery.ecr.aws/lambda/provided

Clientes de interface de runtime do Go

O pacote `aws-lambda-go/lambda` inclui uma implementação da interface de runtime. Para obter exemplos de como usar `aws-lambda-go/lambda` na sua imagem, consulte Usar uma imagem base somente para sistema operacional da AWS ou Usar uma imagem base que não é da AWS.

Usar uma imagem base somente para sistema operacional da AWS

O Go é implementado de forma diferente de outros runtimes gerenciados. Como o Go é compilado nativamente com um binário executável, ele não requer um runtime de linguagem dedicado. Use uma imagem de base somente do sistema operacional para criar imagens de contêiner para as funções do Go.

Tags	Runtime	Sistema operacional	Dockerfile	Desaprovação
al2023	Runtime somente para	Amazon Linux 2023	Dockerfile para runtime somente para sistema operacional no GitHub	

Tags	Runtime	Sistema operacional	Dockerfile	Desaprovação
	sistema operacional			
al2	Runtime somente para sistema operacional	Amazon Linux 2	Dockerfile para runtime somente para sistema operacional no GitHub	

Para obter mais informações sobre esta imagem base, consulte fornecido na galeria pública do Amazon ECR.

Você deve incluir o pacote aws-lambda-go/lambda com seu manipulador de Go. Esse pacote implementa o modelo de programação para Go, incluindo a interface de runtime.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Go
- Docker
- AWS Command Line Interface (AWS CLI) versão 2

Criar uma imagem baseada na imagem base provided.al2023

Criar e implantar uma função do Go com a imagem base **`provided.al2023`**

1. Crie um diretório para o projeto e depois mude para esse diretório.

```
mkdir hello
cd hello
```

2. Inicialize um novo módulo do Go.

```
go mod init example.com/hello-world
```

3. Adicione a biblioteca lambda como uma dependência do seu novo módulo.

```
go get github.com/aws/aws-lambda-go/lambda
```

4. Crie um arquivo denominado `main.go` e abra-o em um editor de texto. Este é o código da função do Lambda. É possível usar o código de exemplo a seguir para fins de teste ou substituí-lo pelo seu próprio código.

```
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. Use um editor de texto para criar um Dockerfile no diretório do projeto. O exemplo de Dockerfile a seguir usa uma compilação em várias etapas. Isso permite que você use uma imagem base diferente em cada etapa. É possível usar uma imagem, como uma imagem base Go, para compilar seu código e construir o binário executável. Em seguida, é possível usar uma imagem diferente, como `provided.al2023`, na declaração `FROM` final, para definir a imagem implantada no Lambda. O processo de construção é separado da imagem final de implantação, portanto, a imagem final contém apenas os arquivos necessários para executar a aplicação.

   Você pode usar a tag `lambda.norpc` opcional para excluir o componente Remote Procedure Call (RPC) da biblioteca do Lambda. O componente RPC só será necessário se você estiver usando o runtime do Go 1.x. descontinuado. A exclusão do RPC reduz o tamanho do pacote de implantação.

Example — Dockerfile de construção em vários estágios

> **Note**
>
> Certifique-se de que a versão do Go que você especifica em seu Dockerfile (por exemplo, `golang:1.20`) seja a mesma versão do Go que você usou para criar sua aplicação.

```
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. Crie a imagem do Docker com o comando docker build. O exemplo a seguir nomeia a imagem como `docker-image` e atribui a ela a tag `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> O comando especifica a opção `--platform linux/amd64` para garantir que seu contêiner seja compatível com o ambiente de execução do Lambda, independentemente da arquitetura da sua máquina de compilação. Se você pretende criar uma função do Lambda usando a arquitetura do conjunto de instruções ARM64, certifique-se de alterar o comando para usar a opção `--platform linux/arm64` em vez disso.

(Opcional) Teste a imagem localmente

Use o emulador de interface de runtime para testar sua imagem localmente. A imagem base `provided.al2023` inclui o emulador de interface de runtime.

Para executar o emulador de interface de runtime na sua máquina local

1. Inicie a imagem do Docker com o comando docker run. Observe o seguinte:

   - `docker-image` é o nome da imagem e `test` é a tag.
   - `./main` é o `ENTRYPOINT` do seu Dockerfile.

   ```
   docker run -d -p 9000:8080 \
   --entrypoint /usr/local/bin/aws-lambda-rie \
   docker-image:test ./main
   ```

   Esse comando executa a imagem como um contêiner e cria um endpoint local em `localhost:9000/2015-03-31/functions/function/invocations`.

2. Em uma nova janela de terminal, publique um evento no seguinte endpoint usando um comando curl:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Algumas funções podem exigir uma carga útil JSON. Exemplo:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

3. Obtenha o ID do contêiner.

   ```
   docker ps
   ```

4. Use o comando docker kill para parar o contêiner. Nesse comando, substitua `3766c4ab331c` pelo ID do contêiner da etapa anterior.

   ```
   docker kill 3766c4ab331c
   ```

Implantação da imagem

Para enviar a imagem ao Amazon ECR e criar a função do Lambda

1. Execute o comando get-login-password para autenticar a CLI do Docker no seu registro do Amazon ECR.

   - Defina o valor `--region` para a Região da AWS onde você deseja criar o repositório do Amazon ECR.
   - Substituir `111122223333` por seu ID da Conta da 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. Crie um repositório no Amazon ECR usando o 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**
   >
   > O repositório do Amazon ECR deve estar na mesma Região da AWS que a função do Lambda.

   Se tiver êxito, você verá uma resposta como esta:

   ```
   {
       "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. Copie o `repositoryUri` da saída na etapa anterior.
4. Execute o comando docker tag para aplicar uma tag na sua imagem local em seu repositório do Amazon ECR como a versão mais recente. Neste comando:
   - Substitua `docker-image:test` pelo nome e tag da sua imagem do Docker.
   - Substitua `<ECRrepositoryUri>` pelo `repositoryUri` que você copiou. Certifique-se de incluir `:latest` no final do URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

   Exemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Execute o comando docker push para implantar a imagem local no repositório do Amazon ECR. Certifique-se de incluir `:latest` no final do URI do repositório.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crie um perfil de execução para a função, caso ainda não tenha um. Você precisará do nome do recurso da Amazon (ARN) do perfil na próxima etapa.
7. Criar a função do Lambda. Em `ImageUri`, especifique o URI do repositório anterior. Certifique-se de incluir `:latest` no final do 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**
>
> É possível criar uma função usando uma imagem em uma conta da AWS diferente desde que a imagem esteja na mesma região da função do Lambda. Para ter mais informações, consulte Permissões entre contas do Amazon ECR.

8. Invoque a função.

```
aws lambda invoke --function-name hello-world response.json
```

Você obterá uma resposta parecida com esta:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Para ver a saída da função, verifique o arquivo `response.json`.

Para atualizar o código da função, você deve criar a imagem novamente, fazer upload da nova imagem no repositório do Amazon ECR e, em seguida, usar o comando update-function-code para implantar a imagem na função do Lambda.

O Lambda resolve a tag de imagem em um resumo de imagem específico. Isso significa que, se você apontar a tag de imagem que foi usada para implantar a função em uma nova imagem no Amazon ECR, o Lambda não atualizará automaticamente a função para usar a nova imagem. Para implantar a nova imagem na mesma função do Lambda, você deverá usar o comando `update-function-code`, mesmo que a tag da imagem no Amazon ECR permaneça a mesma.

Usar uma imagem base que não é da AWS

Você pode criar uma imagem de contêiner para Go baseada em uma imagem base que não seja da AWS. O Dockerfile de exemplo das etapas a seguir usa uma imagem base do Alpine.

Você deve incluir o pacote aws-lambda-go/lambda com seu manipulador de Go. Esse pacote implementa o modelo de programação para Go, incluindo a interface de runtime.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- Go
- Docker
- AWS Command Line Interface (AWS CLI) versão 2

Criar uma imagem de uma imagem base alternativa

Para criar e implantar uma função do Go com uma imagem base do Alpine

1. Crie um diretório para o projeto e depois mude para esse diretório.

```
mkdir hello
cd hello
```

2. Inicialize um novo módulo do Go.

```
go mod init example.com/hello-world
```

3. Adicione a biblioteca lambda como uma dependência do seu novo módulo.

```
go get github.com/aws/aws-lambda-go/lambda
```

4. Crie um arquivo denominado `main.go` e abra-o em um editor de texto. Este é o código da função do Lambda. É possível usar o código de exemplo a seguir para fins de teste ou substituí-lo pelo seu próprio código.

```
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. Use um editor de texto para criar um Dockerfile no diretório do projeto. O Dockerfile de exemplo a seguir usa uma imagem base do Alpine.

   Example Dockerfile

   > **Note**
   >
   > Certifique-se de que a versão do Go que você especifica em seu Dockerfile (por exemplo, `golang:1.20`) seja a mesma versão do Go que você usou para criar sua aplicação.

```
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. Crie a imagem do Docker com o comando docker build. O exemplo a seguir nomeia a imagem como `docker-image` e atribui a ela a tag `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

> **Note**
>
> O comando especifica a opção `--platform linux/amd64` para garantir que seu contêiner seja compatível com o ambiente de execução do Lambda, independentemente da arquitetura da sua máquina de compilação. Se você pretende criar uma função do Lambda usando a arquitetura do conjunto de instruções ARM64, certifique-se de alterar o comando para usar a opção `--platform linux/arm64` em vez disso.

(Opcional) Teste a imagem localmente

Use o emulador de interface de runtime para testar a imagem localmente. Você pode compilar o emulador em sua imagem ou usar o procedimento a seguir instalá-lo na sua máquina local.

Para instalar o emulador de interface de runtime na sua máquina local

1. No diretório do projeto, execute o comando a seguir para baixar o emulador de interface de runtime (arquitetura x86-64) do GitHub e instalá-lo na sua máquina local.

   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
   ```

   Para instalar o emulador arm64, substitua o URL do repositório do GitHub no comando anterior pelo seguinte:

   ```
   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
```

Para instalar o emulador de arm64, substitua `$downloadLink` pelo seguinte:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Inicie a imagem do Docker com o comando docker run. Observe o seguinte:

   - `docker-image` é o nome da imagem e `test` é a tag.
   - `/main` é o `ENTRYPOINT` do seu 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
   ```

   Esse comando executa a imagem como um contêiner e cria um endpoint local em `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se você criou a imagem do Docker para a arquitetura do conjunto de instruções ARM64, certifique-se de usar a opção `--platform linux/arm64`, em vez de `--platform linux/amd64`.

3. Publique um evento no endpoint local.

   Linux/macOS

   No Linux e no MacOS, execute o seguinte comando `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   No PowerShell, execute o seguinte comando `Invoke-WebRequest`:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

   Esse comando invoca a função com um evento vazio e retorna uma resposta. Caso esteja usando seu próprio código de função em vez do código de função de exemplo, você talvez queira invocar a função com uma carga útil JSON. Exemplo:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
    "application/json"
   ```

4. Obtenha o ID do contêiner.

```
docker ps
```

5. Use o comando docker kill para parar o contêiner. Nesse comando, substitua `3766c4ab331c` pelo ID do contêiner da etapa anterior.

```
docker kill 3766c4ab331c
```

Implantação da imagem

Para enviar a imagem ao Amazon ECR e criar a função do Lambda

1. Execute o comando get-login-password para autenticar a CLI do Docker no seu registro do Amazon ECR.

   - Defina o valor `--region` para a Região da AWS onde você deseja criar o repositório do Amazon ECR.
   - Substituir `111122223333` por seu ID da Conta da 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. Crie um repositório no Amazon ECR usando o 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**
>
> O repositório do Amazon ECR deve estar na mesma Região da AWS que a função do Lambda.

Se tiver êxito, você verá uma resposta como esta:

```
{
    "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. Copie o `repositoryUri` da saída na etapa anterior.
4. Execute o comando docker tag para aplicar uma tag na sua imagem local em seu repositório do Amazon ECR como a versão mais recente. Neste comando:

   - Substitua `docker-image:test` pelo nome e tag da sua imagem do Docker.
   - Substitua `<ECRrepositoryUri>` pelo `repositoryUri` que você copiou. Certifique-se de incluir `:latest` no final do URI.

   ```
   docker tag docker-image:test <ECRrepositoryUri>:latest
   ```

   Exemplo:

   ```
   docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
   world:latest
   ```

5. Execute o comando docker push para implantar a imagem local no repositório do Amazon ECR. Certifique-se de incluir `:latest` no final do URI do repositório.

   ```
   docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
   ```

6. Crie um perfil de execução para a função, caso ainda não tenha um. Você precisará do nome do recurso da Amazon (ARN) do perfil na próxima etapa.

7. Criar a função do Lambda. Em `ImageUri`, especifique o URI do repositório anterior. Certifique-se de incluir `:latest` no final do 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**
>
> É possível criar uma função usando uma imagem em uma conta da AWS diferente desde que a imagem esteja na mesma região da função do Lambda. Para ter mais informações, consulte Permissões entre contas do Amazon ECR.

8. Invoque a função.

```
aws lambda invoke --function-name hello-world response.json
```

Você obterá uma resposta parecida com esta:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Para ver a saída da função, verifique o arquivo `response.json`.

Para atualizar o código da função, você deve criar a imagem novamente, fazer upload da nova imagem no repositório do Amazon ECR e, em seguida, usar o comando update-function-code para implantar a imagem na função do Lambda.

O Lambda resolve a tag de imagem em um resumo de imagem específico. Isso significa que, se você apontar a tag de imagem que foi usada para implantar a função em uma nova imagem no Amazon ECR, o Lambda não atualizará automaticamente a função para usar a nova imagem. Para implantar a nova imagem na mesma função do Lambda, você deverá usar o comando `update-function-code`, mesmo que a tag da imagem no Amazon ECR permaneça a mesma.

Registro em log da função do AWS Lambda em Go

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome e envia logs para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente do runtime do Lambda envia detalhes sobre cada invocação à transmissão de logs e transmite os logs e outras saídas do código de sua função. Para ter mais informações, consulte Usar logs do Amazon CloudWatch com o AWS Lambda.

Esta página descreve como produzir a saída de logs usando o código de sua função do Lambda ou acessar os logs usando a AWS Command Line Interface, o console do Lambda ou o console do CloudWatch.

Seções

Criar uma função que retorna logs

Para gerar os logs do código de função, você pode usar métodos no pacote fmt ou qualquer biblioteca de logs que grave em `stdout` ou em `stderr`. O exemplo a seguir usa o pacote de logs.

Example main.go: registro em log

```
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 do 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
```

O runtime do Go registra em log as linhas `START`, `END` e `REPORT` para cada invocação. A linha do relatório fornece os detalhes a seguir.

RELATAR campos de dados de linha

- RequestId: o ID de solicitação exclusivo para a invocação.
- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.

- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o runtime levou para carregar a função e executar o código fora do método do handler.
- XRAY TraceId: para solicitações rastreadas, o ID de rastreamento do AWS X-Ray.
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda.

Se seu código puder ser testado no editor de Código incorporado, você encontrará logs nos resultados de execução. Ao usar o recurso de teste do console para invocar uma função, você encontrará Saída de log na seção Detalhes.

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a página Log groups (Grupos de log).
2. Escolha o grupo de logs de sua função (/aws/lambda/***nome-de-sua-função***).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma instância da sua função. Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos intrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- AWS CLI – Configuração rápida com `aws configure`

Você pode usar a AWS CLI para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

A seguinte saída deverá ser mostrada:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `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
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

A seguinte saída deverá ser mostrada:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e Ubuntu no Windows. Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa `sed` para remover as aspas do arquivo de saída e fica inativo por 15 segundos para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 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 macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod -R 755 get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
}
```


Excluir logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ouConfigurar um período de retençãoapós o qual os logs são excluídos automaticamente.

Instrumentação do código Go no AWS Lambda

O Lambda se integra ao AWS X-Ray para ajudar você a rastrear, depurar e otimizar aplicações do Lambda. É possível usar o X-Ray para rastrear uma solicitação enquanto ela atravessa recursos na aplicação, o que pode incluir funções Lambda e outros produtos da AWS.

Para enviar dados de rastreamento ao X-Ray, você pode usar uma das duas bibliotecas SDK:

- AWS Distro for OpenTelemetry (ADOT): uma distribuição segura, pronta para produção e com suporte na AWS do SDK OpenTelemetry (OTel).
- AWS X-Ray SDK para Go: um SDK para geração e envio de dados de rastreamento ao X-Ray.

Cada um dos SDKs oferece maneiras de enviar dados de telemetria ao serviço do X-Ray. Em seguida, é possível usar o X-Ray para visualizar, filtrar e obter insights sobre as métricas de performance da aplicação para identificar problemas e oportunidades de otimização.

> **Important**
>
> Os SDKs do X-Ray e do Powertools para AWS Lambda fazem parte de uma solução de instrumentação totalmente integrada oferecida pela AWS. As camadas do Lambda para ADOT fazem parte de um padrão em todo o setor para instrumentação de rastreamento que coleta mais dados em geral, mas pode não ser adequado para todos os casos de uso. É possível implementar o rastreamento de ponta a ponta no X-Ray usando ambas as soluções. Para saber mais sobre como escolher entre elas, consulte Como escolher entre os SDKs do AWS Distro para OpenTelemetry e do X-Ray.

Seções

Usar o ADOT para instrumentar funções Go

O ADOT fornece camadas do Lambda totalmente gerenciadas que empacotam tudo o que você precisa para coletar dados de telemetria usando o SDK do OTel. Ao consumir essa camada, é possível instrumentar suas funções Lambda sem precisar modificar nenhum código de função. Você também pode configurar sua camada para fazer a inicialização personalizada do OTel. Para obter mais informações, consulte Custom configuration for the ADOT Collector on Lambda (Configuração personalizada para o ADOT Collector no Lambda) na documentação do ADOT.

Para runtimes Go, você pode adicionar a camada do Lambda gerenciada pela AWS para ADOT Go a fim de instrumentar suas funções automaticamente. Para obter instruções detalhadas sobre como adicionar essa camada, consulte Suporte do AWS Distro for OpenTelemetry Lambda para Go, na documentação do ADOT.

Usar o SDK do X-Ray para instrumentar suas funções Go

Para registrar detalhes sobre as chamadas feitas pela sua função do Lambda para outros recursos na sua aplicação, você também pode usar o AWS X-Ray SDK para Go. Para obter o SDK, baixe-o do Repositório do GitHub com `go get`:

```
go get github.com/aws/aws-xray-sdk-go
```

Para instrumentar clientes do AWS SDK, passe o cliente para o método `xray.AWS()`. Em seguida, você pode rastrear chamadas usando a versão `WithContext` do método.

```
svc := s3.New(session.New())
xray.AWS(svc.Client)
...
svc.ListBucketsWithContext(ctx aws.Context, input *ListBucketsInput)
```

Depois de adicionar as dependências corretas e fazer as devidas mudanças de código, ative o rastreamento na configuração da sua função usando o console do Lambda ou a API.

Ativar o rastreamento com o console do Lambda

Para alternar o rastreamento ativo na sua função do Lambda usando o console, siga as etapas abaixo:

Para ativar o rastreamento ativo

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e depois Monitoring and operations tools (Ferramentas de monitoramento e operações).
4. Selecione a opção Editar.
5. Em X-Ray, ative a opção Active tracing (Rastreamento ativo).
6. Escolha Salvar.

Ativar o rastreamento com a API do Lambda

Configure o rastreamento na sua função do Lambda com aAWS CLI ou o AWS SDK, usando as seguintes operações de API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration \
--function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Ativar o rastreamento com o AWS CloudFormation

Para ativar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example function-inline.yml: configuração de rastreamento

```
Resources:
```

```
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example template.yml: configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretar um rastreamento do X-Ray

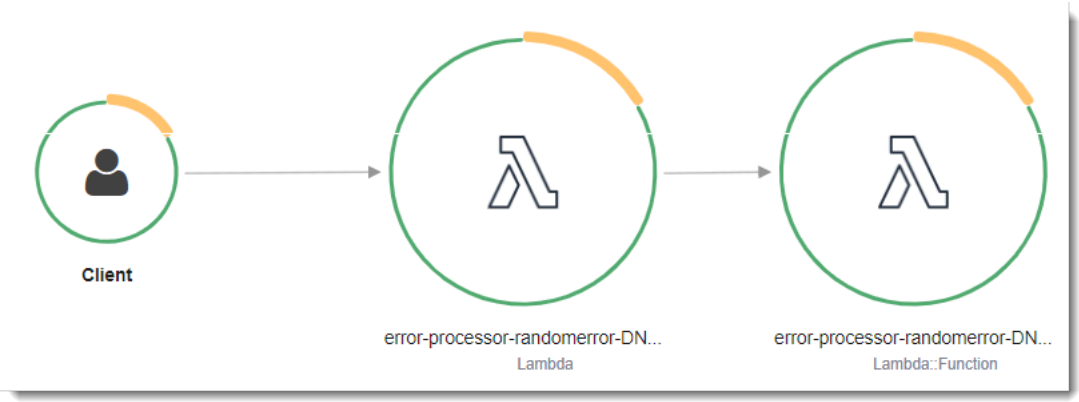
Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você ativa o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à função de execução da função. Caso contrário, adicione a política AWSXRayDaemonWriteAccess à função de execução.

Após configurar o rastreamento ativo, você pode observar solicitações específicas por meio da aplicação. O grafo de serviço do X-Ray exibe informações sobre sua aplicação e todos os componentes. A imagem a seguir demonstra uma aplicação com duas funções. A função principal processa eventos e, às vezes, retorna erros. A segunda função de cima para baixo processa erros que aparecem no primeiro grupo de logs e usa o AWS SDK para chamar o X-Ray, o Amazon Simple Storage Service (Amazon S3) e o Amazon CloudWatch Logs.

O X-Ray não rastreia todas as solicitações para sua aplicação. O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento seja eficiente, enquanto ainda fornece uma amostra representativa das solicitações. A taxa de amostragem é uma solicitação por segundo e 5% de solicitações adicionais.

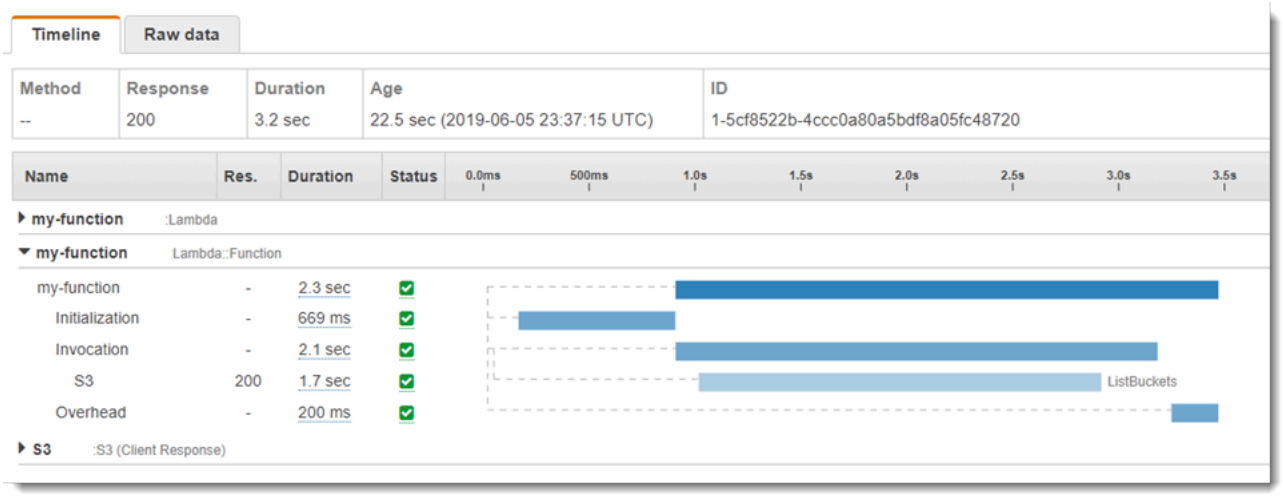
> **Note**
>
> Não é possível configurar a taxa de amostragem do X-Ray para suas funções.

No X-Ray, um rastreamento registra informações sobre uma solicitação que é processada por um ou mais serviços. O Lambda registra dois segmentos por rastreamento, o que cria dois nós no gráfico de serviços. A imagem a seguir destaca esses dois nós:



O primeiro nó à esquerda representa o serviço do Lambda, que recebe a solicitação de invocação. O segundo nó representa a sua função do Lambda específica. O exemplo a seguir mostra um

rastreamento com esses dois segmentos. Ambos têm o nome my-function, mas um tem a origem `AWS::Lambda` e o outro, a origem `AWS::Lambda::Function`. Se o segmento `AWS::Lambda` mostrar um erro, o serviço Lambda teve um problema. Se o segmento `AWS::Lambda::Function` mostrar um erro, sua função teve um problema.



Este exemplo expande o segmento `AWS::Lambda::Function` para mostrar seus três subsegmentos:

- Inicialização: representa o tempo gasto carregando a função e executando o código de inicialização. Esse subsegmento aparece somente para o primeiro evento que cada instância da função processa.
- Invocação: representa o tempo gasto na execução do código do manipulador.
- Sobrecarga: representa o tempo gasto pelo runtime do Lambda preparando-se para lidar com o próximo evento.

Você também pode instrumentar clientes HTTP, registrar consultas SQL e criar subsegmentos personalizados com anotações e metadados. Para obter mais informações, consulte AWS X-Ray SDK para Go, no Guia do desenvolvedor do AWS X-Ray.

> **Definição de preço**
>
> Você pode usar o rastreamento do X-Ray gratuitamente todos os meses até determinado limite como parte do nível gratuito da AWS. Além do limite, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter mais informações, consulte Preços do AWS X-Ray.

Usar variáveis de ambiente do

Para acessar variáveis de ambiente em Go, use a função Getenv .

O seguinte explica como fazer isso. Observe que a função importa o pacote fmt para formatar os resultados impressos e o pacote do os, uma interface do sistema independente de plataforma, que permite acessar variáveis de 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"))

}
```

Para obter uma lista de variáveis de ambiente definidas pelo runtime do Lambda, consulte Variáveis de ambiente com runtime definido.

Construir funções do Lambda com C#

É possível executar a aplicação do .NET no Lambda usando os runtimes do .NET 6 ou .NET 8 gerenciados, um runtime personalizado ou uma imagem de contêiner. Depois que o código da aplicação for compilado, você poderá implantá-lo no Lambda como um arquivo .zip ou uma imagem de contêiner. O Lambda fornece os seguintes runtimes para linguagens .NET:

.NET

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
.NET 8	`dotnet8`	Amazon Linux 2023			
.NET 6	`dotnet6`	Amazon Linux 2	12 de novembro de 2024	28 de fevereiro de 2025	31 de março de 2025

Configurar seu ambiente de desenvolvimento .NET

Para desenvolver e compilar funções do Lambda, você pode usar qualquer um dos ambientes de desenvolvimento integrados (IDEs) .NET comumente disponíveis, como Microsoft Visual Studio, Visual Studio Code e JetBrains Rider. Para simplificar sua experiência de desenvolvimento, a AWS fornece um conjunto de modelos de projeto do .NET, bem como a interface de linha de comando (CLI) do `Amazon.Lambda.Tools`.

Execute os seguintes comandos da CLI do .NET para instalar esses modelos de projeto e ferramentas de linha de comando.

Instalar os modelos de projeto do .NET

Para instalar os modelos de projeto (.NET 8):

```
dotnet new install Amazon.Lambda.Templates
```

Para instalar os modelos de projeto (.NET 6):

```
dotnet new --install Amazon.Lambda.Templates
```

> **Note**
>
> Se você estiver usando o runtime do Lambda gerenciado do .NET 6, recomendamos atualizá-lo para usar o .NET 8. Para saber mais, consulte Gerenciando atualizações do runtime do AWS Lambda e Introdução ao runtime do .NET 8 para AWS Lambda no blog AWS Compute.

Instalar e atualizar as ferramentas da CLI

Execute os comandos a seguir para instalar, atualizar e desinstalar a CLI do `Amazon.Lambda.Tools`.

Para instalar as ferramentas da linha de comando:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Para atualizar as ferramentas da linha de comando:

```
dotnet tool update -g Amazon.Lambda.Tools
```

Para desinstalar as ferramentas da linha de comando:

```
dotnet tool uninstall -g Amazon.Lambda.Tools
```

Definir o manipulador de função do Lambda em C#

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. A função é executada até que o manipulador retorne uma resposta, seja encerrado ou atinja o tempo limite.

Quando sua função é invocada e o Lambda executa o método manipulador da função, ele passa dois argumentos para a função. O primeiro argumento é o objeto do `event`. Quando outro AWS service (Serviço da AWS) invoca a função, o objeto `event` contém dados sobre o evento que fez com que a função fosse invocada. Por exemplo, um objeto `event` do API Gateway contém informações sobre o caminho, o método HTTP e os cabeçalhos HTTP. A estrutura exata do evento varia de acordo com a invocação da função pelo AWS service (Serviço da AWS). Consulte Integração com outros serviços para obter mais informações sobre formatos de eventos para serviços individuais.

O Lambda também passa um objeto `context` para a função. Esse objeto contém informações sobre a invocação, a função e o ambiente de execução. Para ter mais informações, consulte the section called “Contexto”.

O formato nativo para todos os eventos do Lambda são streams de bytes representando o evento formatado em JSON. A não ser que os parâmetros de entrada e saída de sua função sejam do tipo `System.IO.Stream`, você precisará serializá-los. Especifique o serializador que você deseja usar definindo o atributo `LambdaSerializer` do assembly. Para ter mais informações, consulte the section called “Serialização em funções do Lambda”.

Tópicos

- Modelos de execução do .NET para Lambda
- Manipuladores de bibliotecas de classes
- Manipuladores de assembly executáveis
- Serialização em funções do Lambda
- Simplificar o código da função com a estrutura Lambda Annotations
- Restrições do manipulador de função do Lambda

Modelos de execução do .NET para Lambda

Há dois modelos de execução diferentes para executar funções do Lambda no .NET: a abordagem de biblioteca de classes e a abordagem de assembly executável.

Na abordagem biblioteca de classes, você fornece ao Lambda uma string indicando o `AssemblyName`, `ClassName` e `Method` da função a ser invocada. Para obter mais informações sobre o formato de dessa string, consulte the section called “Manipuladores de bibliotecas de classes”. Durante a fase de inicialização da função, a classe da função é inicializada e qualquer código no construtor é executado.

Na abordagem de assembly executável, você usa o recurso de instruções de nível superior do C# 9. Essa abordagem gera um assembly executável que o Lambda executa sempre que recebe um comando invocar para a função. Você fornece ao Lambda somente o nome do assembly executável a ser executado.

As seções a seguir fornecem exemplos de código de função para essas duas abordagens.

Manipuladores de bibliotecas de classes

O código da função do Lambda a seguir mostra um exemplo de um método manipulador (`FunctionHandler`) para uma função do Lambda que usa a abordagem de biblioteca de classes. Neste exemplo de função, o Lambda recebe um evento do API Gateway que invoca a função. A função lê um registro de um banco de dados e retorna o registro como parte da resposta do API Gateway.

```
[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)
        };
    }
}
```

Ao criar uma função do Lambda, você precisa fornecer ao Lambda informações sobre o manipulador da função na forma de uma string do manipulador. Isso informa ao Lambda qual método do código deve ser executado quando sua função é invocada. Em C#, o formato da string do manipulador ao usar a abordagem da biblioteca de classes é o seguinte:

`ASSEMBLY::TYPE::METHOD`, em que:

- `ASSEMBLY` é o nome do arquivo de assembly do .NET para a aplicação. Se você usar a CLI do `Amazon.Lambda.Tools` para criar a aplicação e não definir o nome do assembly usando a propriedade `AssemblyName` no arquivo .csproj, `ASSEMBLY` será simplesmente o nome do arquivo .csproj.
- `TYPE` é o nome completo do tipo de manipulador, que consiste em `Namespace` e o `ClassName`.
- `METHOD` é o nome do método do manipulador da função no código.

Para o código de exemplo mostrado, se o assembly for denominado `GetProductHandler`, a string do manipulador será `GetProductHandler::GetProductHandler.Function::FunctionHandler`.

Manipuladores de assembly executáveis

No exemplo a seguir, a função do Lambda é definida como um assembly executável. O método manipulador nesse código é denominado `Handler`. Ao usar assemblies executáveis, o runtime do Lambda deve ser inicializado. Para fazer isso, use o método `LambdaBootstrapBuilder.Create`. Esse método usa como entradas o método que a função usa como manipulador e o serializador Lambda a ser usado.

Para obter mais informações sobre o uso de instruções de nível superior, consulte Introdução ao runtime do .NET 6 para o AWS Lambda no blog de computação da AWS.

```
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)
    };
};
```

Ao usar assemblies executáveis, a string do manipulador que diz ao Lambda como executar seu código é o nome do assembly. Neste exemplo, isso seria `GetProductHandler`.

Serialização em funções do Lambda

Se a função do Lambda usar tipos de entrada ou de saída diferentes de objeto `Stream`, você deverá adicionar uma biblioteca de serialização à aplicação. Você pode implementar a serialização usando a serialização padrão baseada em reflexão fornecida por `System.Text.Json` e `Newtonsoft.Json` ou usando a serialização gerada pela fonte.

Usar a serialização gerada pela fonte

A serialização gerada pela fonte é um recurso do .NET versão 6 e posteriores que permite que o código de serialização seja gerado durante a compilação. Ele elimina a necessidade de reflexão e pode melhorar a performance da função. Para usar a serialização gerada pela fonte em sua função, faça o seguinte:

- Crie uma nova classe parcial herdada de `JsonSerializerContext`, adicionando atributos `JsonSerializable` para todos os tipos que exigem serialização ou desserialização.

- Configure o `LambdaSerializer` para usar um `SourceGeneratorLambdaJsonSerializer<T>`.
- Atualize as serializações ou desserializações manuais no código da aplicação para usar a classe recém-criada.

Um exemplo de função usando a serialização gerada pela fonte é mostrado no código a seguir.

```
[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 você quiser usar a compilação antecipada (AOT) nativa com o Lambda, você deve usar a serialização gerada pela fonte.

Usar serialização baseada em reflexão

A AWS fornece bibliotecas pré-criadas para permitir que você adicione rapidamente a serialização á aplicação. Você configura isso usando os pacotes NuGet `Amazon.Lambda.Serialization.SystemTextJson` ou `Amazon.Lambda.Serialization.Json`. Nos bastidores, o `Amazon.Lambda.Serialization.SystemTextJson` usa `System.Text.Json` para realizar tarefas de serialização e o `Amazon.Lambda.Serialization.Json` usa o pacote `Newtonsoft.Json`.

Você pode criar sua própria biblioteca de serialização implementando a interface `ILambdaSerializer`, que está disponível como parte da biblioteca `Amazon.Lambda.Core`. Essa interface define dois métodos:

- `T Deserialize<T>(Stream requestStream);`

  Você implementa esse método para desserializar a carga útil da solicitação da API `Invoke` no objeto que é passado para o manipulador da função do Lambda.
- `T Serialize<T>(T response, Stream responseStream);`

  Você implementa esse método para serializar o resultado retornado pelo manipulador da função do Lambda na carga útil da resposta retornada pela operação da API `Invoke`.

Simplificar o código da função com a estrutura Lambda Annotations

O Lambda Annotations é uma estrutura para o .NET 6 e o .NET 8 que simplifica a escrita de funções do Lambda usando C#. Com a estrutura Annotations, você pode substituir grande parte do código em uma função Lambda escrita usando o modelo de programação regular. O código escrito usando a estrutura usa expressões mais simples que permitem que você se concentre em sua lógica de negócios.

O código de exemplo a seguir mostra como o uso da estrutura de anotações pode simplificar a escrita de funções do Lambda. O primeiro exemplo mostra o código escrito usando o modelo regular de programa do Lambda e o segundo mostra o equivalente usando a estrutura 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;
}
```

Para ver outro exemplo de como o uso do Lambda Annotations pode simplificar seu código, consulte este exemplo de aplicação entre serviços no repositório do `awsdocs/aws-doc-sdk-examples` GitHub. A pasta `PamApiAnnotations` usa Lambda Annotations no arquivo `function.cs`

principal. Para comparação, a pasta `PamApi` tem arquivos equivalentes escritos usando o modelo de programação do Lambda normal.

A estrutura Annotations usa geradores de código-fonte para gerar código que é traduzido do modelo de programação do Lambda para o código visto no segundo exemplo.

Para obter mais informações sobre como usar o Lambda Annotations para .NET, consulte os seguintes recursos:

- O repositório do `aws/aws-lambda-dotnet` GitHub.
- Apresentando o.NET Annotations Lambda Framework (versão prévia) no blog de ferramentas para desenvolvedores da AWS
- O pacote NuGet `Amazon.Lambda.Annotations`.

Injeção de dependência com a estrutura Lambda Annotations

Você também pode usar a estrutura do Lambda Annotations para adicionar injeção de dependência nas funções do Lambda usando a sintaxe com a qual você está familiarizado. Quando você adiciona um atributo `[LambdaStartup]` a um arquivo `Startup.cs`, a estrutura do Lambda Annotations gera o código necessário durante a compilação.

```
[LambdaStartup]
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<IDatabaseRepository, DatabaseRepository>();
    }
}
```

A função Lambda pode injetar serviços usando injeção de construtor ou injetando em métodos individuais usando o atributo `[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);
    }
}
```

Restrições do manipulador de função do Lambda

Observe que há algumas restrições na assinatura do manipulador.

- Não pode ser `unsafe` e usar tipos de ponteiro na assinatura do manipulador, embora seja possível usar o contexto `unsafe` dentro do método do manipulador e de suas dependências. Para obter mais informações, consulte não seguro (Referência de C#) no site Microsoft Docs.
- Não pode passar um número variável de parâmetros usando a palavra-chave `params` ou usar `ArgIterator` como um parâmetro de entrada ou retorno que é usado para dar suporte a um número variável de parâmetros.
- O manipulador não pode ser um método genérico por exemplo, IList<T> Sort<T>(entrada IList<T>).
- Manipuladores assíncronos com assinaturas `async void` não são compatíveis.

Criar e implantar funções do Lambda em C# com arquivos .zip

Um pacote de implantação do .NET (arquivo .zip) contém o assembly compilado da função junto com todas as suas dependências de assembly. O pacote também contém um arquivo *proj*`.deps.json`. Isso sinaliza para o runtime do .NET todas as dependências da função e um arquivo *proj*`.runtimeconfig.json`, que é usado para configurar o runtime.

Para implantar funções do Lambda individuais, você pode usar a CLI do .NET Lambda Global `Amazon.Lambda.Tools`. O uso do comando `dotnet lambda deploy-function` cria automaticamente um pacote de implantação .zip e o implanta no Lambda. Porém, recomendamos que você use estruturas como o AWS Serverless Application Model (AWS SAM) ou o AWS Cloud Development Kit (AWS CDK) para implantar as aplicações do .NET na AWS.

As aplicações com tecnologia sem servidor geralmente incluem uma combinação de funções do Lambda e outros Serviços da AWS gerenciados trabalhando juntos para realizar uma tarefa de negócios específica. O AWS SAM e o AWS CDK simplificam a criação e implantação de funções do Lambda com outros Serviços da AWS em escala. A especificação do modelo do AWS SAM fornece uma sintaxe simples e clara para descrever as funções, APIs, permissões, configurações e outros recursos do da AWS que compõem sua aplicação com tecnologia sem servidor. Com o AWS CDK, você define a infraestrutura de nuvem como código para ajudar a criar aplicações confiáveis, escaláveis e econômicas na nuvem, usando linguagens e estruturas de programação modernas, como o .NET. Tanto o AWS CDK quanto o AWS SAM usam a CLI do .NET Lambda Global para empacotar as funções.

Embora seja possível usar as Camadas do Lambda com funções em C# usando a CLI do .NET Core, não recomendamos. Funções em C# que usam camadas carregam manualmente os conjuntos compartilhados na memória durante o Fase de inicialização, o que pode aumentar o tempo de início a frio. Em vez disso, inclua todo o código compartilhado no momento da compilação para aproveitar as otimizações integradas do compilador .NET.

Você pode encontrar instruções para criar e implantar as funções do Lambda do .NET usando o AWS SAM o AWS CDK e a CLI do .NET Lambda Global nas seções a seguir.

Tópicos

- Usando a CLI do .NET Lambda Global
- Implantar funções do Lambda em C# usando o AWS SAM
- Implantar funções do Lambda em C# usando o AWS CDK

- Implementar aplicações ASP.NET

Usando a CLI do .NET Lambda Global

A CLI do .NET e a extensão .NET Lambda Global Tools (`Amazon.Lambda.Tools`) oferecem uma maneira multiplataforma de criar aplicações do Lambda baseadas no .NET, empacotá-las e implantá-las no Lambda. Nesta seção, você aprenderá a criar novos projetos do Lambda do .NET usando a CLI do .NET e os modelos do Amazon Lambda e a empacotá-los e implantá-los usando o `Amazon.Lambda.Tools`

Tópicos

Pré-requisitos

SDK do .NET 8

Se você ainda não o fez, instale o SDK e o runtime do .NET 8.

Modelos de projetos do AWS Amazon.Lambda.Templates do .NET

Para gerar o código da função do Lambda, use o Pacote do NuGet Amazon.Lambda.Templates. Para instalar esse pacote de modelo, execute o comando a seguir:

```
dotnet new install Amazon.Lambda.Templates
```

Ferramentas AWS Amazon.Lambda.Tools da CLI Global do .NET

Para criar as funções do Lambda, você usa a extensão .NET Global Tools do Amazon.Lambda.Tools. Para instalar Amazon.Lambda.Tools, execute o comando a seguir:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Para obter mais informações sobre a extensão Amazon.Lambda.Tools da CLI do .NET, consulte o repositório AWS Extensions for .NET CLI no GitHub.

Criar projetos do .NET usando a CLI do .NET

Na CLI do .NET, você usa o comando `dotnet new` para criar projetos do .NET em uma linha de comando. O Lambda oferece modelos adicionais usando o pacote NuGet `Amazon.Lambda.Templates`.

Depois de instalar o pacote, execute o comando a seguir para ver uma lista dos modelos disponíveis.

```
dotnet new list
```

Para examinar detalhes sobre um modelo, use a opção `help`. Por exemplo, para ver detalhes sobre o modelo de `lambda.EmptyFunction`, execute o comando a seguir.

```
dotnet new lambda.EmptyFunction --help
```

Para criar um modelo básico para uma função do .NET Lambda, use o modelo `lambda.EmptyFunction`. Isso cria uma função simples que pega uma string como entrada e a converte em maiúsculas usando o método `ToUpper`. O modelo é compatível com as opções a seguir:

- `--name`: o nome da função.
- `--region`: a região da AWS na qual criar a função.
- `--profile`: o nome de um perfil no arquivo de credenciais do AWS SDK for .NET. Para saber mais sobre perfis de credenciais no .NET, consulte Configure AWS credentials no AWS SDK for .NET Developer Guide.

Neste exemplo, criamos uma nova função vazia denominada `myDotnetFunction` usando o perfil e as configurações do Região da AWS padrão:

```
dotnet new lambda.EmptyFunction --name myDotnetFunction
```

Esse comando cria os seguintes arquivos e diretórios no diretório do seu projeto.

```
### myDotnetFunction
    ### src
    #   ### myDotnetFunction
    #       ### Function.cs
    #       ### Readme.md
```

```
    #       ### aws-lambda-tools-defaults.json
    #       ### myDotnetFunction.csproj
    ### test
        ### myDotnetFunction.Tests
            ### FunctionTest.cs
            ### myDotnetFunction.Tests.csproj
```

No diretório `src/myDotnetFunction`, examine os seguintes arquivos:

- aws-lambda-tools-defaults.json: este é o local em que você especifica as opções de linha de comando ao implantar sua função do Lambda. Por exemplo:

```
  "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: o código da função do manipulador do Lambda. É um modelo C # que inclui a biblioteca `Amazon.Lambda.Core` padrão e um atributo `LambdaSerializer` padrão. Para obter mais informações sobre os requisitos e as opções de serialização, consulte Serialização em funções do Lambda. Isso também inclui uma função de exemplo que você pode editar para aplicar o código de sua função do 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();
    }
}
```

- myDotnetFunction.csproj: um arquivo MSBuild que lista os arquivos e assemblies que compõem a aplicação.

```
<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: use este arquivo para documentar sua função do Lambda.

No diretório `myfunction/test`, examine os seguintes arquivos:

- myDotnetFunction.Tests.csproj: conforme observado anteriormente, este é um arquivo MSBuild que lista os arquivos e assemblies que compõem seu projeto de teste. Observe também que ele inclui a biblioteca `Amazon.Lambda.Core`, portanto, é possível integrar perfeitamente qualquer modelo do Lambda necessário para testar sua função.

```
<Project Sdk="Microsoft.NET.Sdk">
   ...

    <PackageReference Include="Amazon.Lambda.Core" Version="2.2.0 " />
   ...
```

- FunctionTest.cs: o mesmo arquivo de modelo de código C # que é incluído no diretório `src`. Edite esse arquivo para espelhar o código de produção de sua função e testá-lo antes de carregar sua função do Lambda em um ambiente de produção.

```
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);
        }
    }
}
```

Implantar projetos do .NET usando a CLI do .NET

Para criar o pacote de implantação e implantá-lo no Lambda, você usa as ferramentas da CLI de `Amazon.Lambda.Tools`. Para implantar a função usando os arquivos que você criou nas etapas anteriores, primeiro navegue até a pasta que contém o arquivo `.csproj` da função.

```
cd myDotnetFunction/src/myDotnetFunction
```

Para implantar o código no Lambda como um pacote de implantação .zip, execute o comando a seguir. Escolha o nome da sua função.

```
dotnet lambda deploy-function myDotnetFunction
```

Durante a implantação, o assistente solicita que você selecione um the section called “Perfil de execução (permissões para funções acessarem outros recursos)”. Para esse exemplo, selecione `lambda_basic_role`.

Depois de implantar a função, você poderá testá-la na nuvem usando o comando `dotnet lambda invoke-function`. Para o código de exemplo no modelo da `lambda.EmptyFunction`, você pode testar a função passando uma string usando a opção `--payload`.

```
dotnet lambda invoke-function myDotnetFunction --payload "Just checking if everything
 is OK"
```

Se a função foi implantada com sucesso, você deverá ver um resultado semelhante ao apresentado a seguir.

```
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
```

Usar camadas Lambda com a CLI do .NET

> **Note**
>
> Usar camadas com funções em uma linguagem compilada, como C#, pode não oferecer a mesma quantidade de benefícios que com uma linguagem interpretada, como Python. Como C# é uma linguagem compilada, suas funções ainda precisam carregar manualmente quaisquer montagens compartilhadas na memória durante a fase inicial, o que pode aumentar os tempos de inicialização a frio. Em vez disso, recomendamos incluir qualquer código compartilhado no momento da compilação para aproveitar as otimizações integradas do compilador.

A CLI do .NET é compatível com comandos que ajudam a publicar camadas e implantar funções C# que consomem camadas. Para publicar uma camada em um bucket específico do Amazon S3, execute o seguinte comando no mesmo diretório do seu arquivo `.csproj`:

```
dotnet lambda publish-layer <layer_name> --layer-type runtime-package-store --s3-
bucket <s3_bucket_name>
```

Em seguida, ao implantar sua função usando a CLI do .NET, especifique o ARN da camada a ser consumida no comando a seguir:

```
dotnet lambda deploy-function <function_name> --function-layers arn:aws:lambda:us-
east-1:123456789012:layer:layer-name:1
```

Para obter um exemplo completo de uma função Hello World, consulte a amostra blank-csharp-com-layer.

Implantar funções do Lambda em C# usando o AWS SAM

O AWS Serverless Application Model (AWS SAM) é um kit de ferramentas que ajuda a simplificar o processo de criação e execução de aplicações com tecnologia sem servidor na AWS. Você define os recursos para a aplicação em um modelo YAML ou JSON e usa a interface da linha de comando do AWS SAM (CLI do AWS SAM) para criar, empacotar e implantar aplicações. Quando você cria uma função do Lambda com base em um modelo do AWS SAM, o AWS SAM cria automaticamente um pacote de implantação .zip ou uma imagem de contêiner com o código da função e todas as dependências que você especificar. O AWS SAM então implanta a função usando uma pilha do AWS

CloudFormation Para saber mais sobre como usar o AWS SAM para criar e implantar funções do Lambda, consulte Conceitos básicos do AWS SAM no Guia do desenvolvedor do AWS Serverless Application Model.

As etapas a seguir mostram como baixar, criar e implantar um exemplo de aplicação Hello World do .NET usando o AWS SAM. Essa aplicação de exemplo usa uma função do Lambda e um endpoint do Amazon API Gateway para implementar um back-end básico de API. Quando você envia uma solicitação HTTP GET ao endpoint do API Gateway, o API Gateway invoca a sua função do Lambda. A função retorna uma mensagem "Hello world, juntamente com o endereço IP da instância da função do Lambda que processa a solicitação.

Quando você criar e implanta a aplicação usando o AWS SAM, nos bastidores, a CLI do AWS SAM usa o comando `dotnet lambda package` para empacotar os pacotes individuais de códigos de função do Lambda.

Pré-requisitos

SDK do .NET 8

Instale o SDK e o runtime do .NET 8.

CLI do AWS SAM versão 1.39 ou posterior

Para saber como instalar a versão mais recente da CLI do AWS SAM, consulte Instalar a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS SAM

1. Inicialize a aplicação usando o modelo de Hello world do .NET com o comando a seguir.

```
sam init --app-template hello-world --name sam-app \
--package-type Zip --runtime dotnet8
```

Esse comando cria os seguintes arquivos e diretórios no diretório do seu projeto.

```
### 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. Navegue até o diretório que contém o `template.yaml file`. Esse arquivo é um modelo que define os recursos da AWS da sua aplicação, incluindo a função do Lambda e uma API do API Gateway.

```
cd sam-app
```

3. Para criar o código-fonte da aplicação, execute o comando a seguir.

```
sam build
```

4. Para implantar a aplicação no AWS, execute o comando a seguir.

```
sam deploy --guided
```

Esse comando empacota e implanta a aplicação com a série de prompts a seguir. Para aceitar as opções padrão, pressione Enter.

> **Note**
>
> Para HelloWorldFunction pode não ter autorização definida, tudo bem?, certifique-se de inserir y.

- Nome da pilha: o nome da pilha a implantar para AWS CloudFormation. Esse nome deve ser exclusivo para a sua Conta da AWS e a Região da AWS.
- Região da AWS: a Região da AWS em que você deseja implantar a aplicação.
- Confirmar as alterações antes da implantação: selecione Sim para revisar manualmente qualquer conjunto de alterações antes do AWS SAM implantar as alterações da aplicação.

Se você selecionar Não, a CLI do AWS SAM implantará automaticamente as alterações da aplicação.

- Permitir a criação de perfil do IAM do SAM CLI: muitos modelos do AWS SAM, incluindo o Hello world neste exemplo, criam perfis do AWS Identity and Access Management (IAM) para dar permissão às suas funções do Lambda para acessar outras Serviços da AWS. Selecione Sim para fornecer permissão para implantar uma pilha AWS CloudFormation que cria ou modifica os perfis do IAM.
- Desativar a reversão: por padrão, se o AWS SAM encontrar um erro durante a criação ou implantação da pilha, ela reverte a pilha para a versão anterior. Selecione Não para aceitar esse padrão.
- HelloWorldFunction pode não ter autorização definida, tudo bem?: insira `y`.
- Salvar argumentos em samconfig.toml: selecione Sim para salvar suas opções de configuração. No futuro, você poderá executar o `sam deploy` novamente sem parâmetros para implantar alterações na aplicação.

5. Quando a implantação da aplicação estiver concluída, a CLI retornará o nome do recurso da Amazon (ARN) da função do Lambda Hello World e o perfil do IAM criado para ela. Ela também exibe o endpoint da API do API Gateway. Para testar a aplicação, abra o endpoint em um navegador. Você verá uma resposta semelhante a que se segue.

```
{"message":"hello world","location":"34.244.135.203"}
```

6. Para excluir os seus recursos, execute o comando a seguir. Observe que o endpoint da API que você criou é um endpoint público acessível pela Internet. Recomendamos excluir este endpoint após o teste.

```
sam delete
```

Próximas etapas

Para saber mais sobre o uso do AWS SAM para criar e implantar funções do Lambda usando o .NET, consulte os seguintes recursos:

- O AWS Serverless Application Model (AWS SAM) Developer Guide
- Criar aplicações do .NET com tecnologia sem servidor com o AWS Lambda e com a CLI do SAM

Implantar funções do Lambda em C# usando o AWS CDK

O AWS Cloud Development Kit (AWS CDK) é uma estrutura de desenvolvimento de software de código aberto para definição da infraestrutura em nuvem como código com linguagens de programação e estruturas modernas, como o .NET. Os projetos do AWS CDK são executados para gerar modelos do AWS CloudFormation que são usados para implantar o código.

Para criar e implantar um exemplo de aplicação Hello world do .NET usando o AWS CDK, siga as instruções nas seções a seguir. A aplicação de amostra implementa um back-end de API básica que consiste em um endpoint do API Gateway e uma função do Lambda. O API Gateway invoca a função do Lambda quando você envia uma solicitação HTTP GET ao endpoint. A função retorna uma mensagem Hello world, juntamente com o endereço IP da instância do Lambda que processa a sua solicitação.

Pré-requisitos

SDK do .NET 8

Instale o SDK e o runtime do .NET 8.

AWS CDK versão 2

Para saber como instalar a versão mais recente do AWS CDK, consulte Conceitos básicos do AWS CDK no Guia do desenvolvedor do AWS Cloud Development Kit (AWS CDK) v2.

Implantar uma aplicação de exemplo do AWS CDK

1. Crie um diretório de projeto para a aplicação de amostra e navegue até ele.

```
mkdir hello-world
cd hello-world
```

2. Inicialize uma nova aplicação do AWS CDK executando o comando a seguir.

```
cdk init app --language csharp
```

O comando cria os seguintes arquivos e diretórios no diretório do projeto.

```
### README.md
### cdk.json
```

```
### src
    ### HelloWorld
    #   ### GlobalSuppressions.cs
    #   ### HelloWorld.csproj
    #   ### HelloWorldStack.cs
    #   ### Program.cs
    ### HelloWorld.sln
```

3. Abra o diretório `src` e crie uma nova função do Lambda usando a CLI do .NET. Essa é a função que você implantará usando o AWS CDK. Neste exemplo, você cria uma função Hello world denominada `HelloWorldLambda` usando o modelo `lambda.EmptyFunction`.

```
cd src
dotnet new lambda.EmptyFunction -n HelloWorldLambda
```

Após esta etapa, a estrutura de diretórios dentro do projeto deve ser semelhante à apresentada a seguir.

```
### 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. Abra o arquivo `HelloWorldStack.cs` do diretório `src/HelloWorld`. Substitua o conteúdo do arquivo pelo código a seguir.

```
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
                }),
            });
        }
    }
```

```
}
```

Esse é o código para compilar e empacotar o código da aplicação, bem como a definição da própria função do Lambda. O objeto `BundlingOptions` permite que um arquivo zip seja criado, juntamente com um conjunto de comandos usados para gerar o conteúdo do arquivo zip. Nessa instância, o comando `dotnet lambda package` é usado para compilar e gerar o arquivo zip.

5. Para implantar a aplicação, execute o comando a seguir.

```
cdk deploy
```

6. Invoque sua função do Lambda implantada usando a CLI do .NET Lambda.

```
dotnet lambda invoke-function HelloWorldFunction -p "hello world"
```

7. Depois de finalizados os testes, você poderá excluir os recursos criados, a menos que deseje mantê-los. Execute o comando a seguir para excluir os seus recursos.

```
cdk destroy
```

Próximas etapas

Para saber mais sobre o uso do AWS CDK para criar e implantar funções do Lambda usando o .NET, consulte os seguintes recursos:

- Trabalhar com o AWS CDK em C#
- Criar, empacotar e publicar funções do Lambda no .NET C# com o CDK do AWS

Implementar aplicações ASP.NET

Além de hospedar funções orientadas por eventos, você também pode usar o .NET com o Lambda para hospedar aplicações leves do ASP.NET. Você pode criar e implantar aplicações ASP.NET usando o pacote NuGet `Amazon.Lambda.AspNetCoreServer`. Nesta seção, você aprenderá a implantar uma API da Web do ASP.NET no Lambda usando as ferramentas CLI do .NET Lambda.

Tópicos

- Pré-requisitos
- Implantando uma API Web do ASP.NET no Lambda

- Implantar APIs mínimas do ASP.NET no Lambda

Pré-requisitos

SDK do .NET 8

Instale o SDK do .NET 8 e o Runtime do ASP.NET Core.

Amazon.Lambda.Tools

Para criar as funções do Lambda, você usa a extensão .NET Global Tools do Amazon.Lambda.Tools. Para instalar Amazon.Lambda.Tools, execute o comando a seguir:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Para obter mais informações sobre a extensão Amazon.Lambda.Tools da CLI do .NET, consulte o repositório AWS Extensions for .NET CLI no GitHub.

Amazon.Lambda.Templates

Para gerar o código da função do Lambda, use o Pacote do NuGet Amazon.Lambda.Templates. Para instalar esse pacote de modelo, execute o comando a seguir:

```
dotnet new --install Amazon.Lambda.Templates
```

Implantando uma API Web do ASP.NET no Lambda

Para implantar uma API da Web usando o ASP.NET, você pode usar os modelos do .NET Lambda para criar um novo projeto de API da Web. Use o comando a seguir para inicializar um novo projeto de API da Web do ASP.NET. No comando de exemplo, denominamos o projeto `AspNetOnLambda`.

```
dotnet new serverless.AspNetCoreWebAPI -n AspNetOnLambda
```

Esse comando cria os seguintes arquivos e diretórios no diretório do seu projeto.

```
.
### 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 o Lambda invoca sua função, o ponto de entrada usado é o arquivo `LambdaEntryPoint.cs`. O arquivo criado pelo modelo do .NET Lambda contém o seguinte código de função.

```
namespace AspNetOnLambda;

public class LambdaEntryPoint : Amazon.Lambda.AspNetCoreServer.APIGatewayProxyFunction
{
    protected override void Init(IWebHostBuilder builder)
    {
        builder
            .UseStartup#Startup#();
    }

    protected override void Init(IHostBuilder builder)
    {
    }
}
```

O ponto de entrada usado pelo Lambda deve herdar de uma das três classes básicas do pacote `Amazon.Lambda.AspNetCoreServer`. Essas três classes básicas são:

- `APIGatewayProxyFunction`

- `APIGatewayHttpApiV2ProxyFunction`
- `ApplicationLoadBalancerFunction`

A classe padrão usada quando você cria o arquivo `LambdaEntryPoint.cs` usando o modelo do .NET Lambda fornecido é `APIGatewayProxyFunction`. A classe básica que você usa na função depende de qual camada de API está na frente da função do Lambda.

Cada uma das três classes básicas contém um método público denominado `FunctionHandlerAsync`. O nome desse método fará parte da string do manipulador que o Lambda usa para invocar a função. O método `FunctionHandlerAsync` transforma a carga útil do evento de entrada no formato ASP.NET correto e a resposta do ASP.NET em uma carga útil de resposta do Lambda. Para o projeto de exemplo `AspNetOnLambda` mostrado, a string do manipulador seria como a que se segue.

```
AspNetOnLambda::AspNetOnLambda.LambdaEntryPoint::FunctionHandlerAsync
```

Para implantar a API no Lambda, execute os comandos a seguir para navegar até o diretório que contém o arquivo de código-fonte e implantar a função usando o AWS CloudFormation.

```
cd AspNetOnLambda/src/AspNetOnLambda
dotnet lambda deploy-serverless
```

> **Tip**
>
> Quando você implanta uma API usando o comando **`dotnet lambda deploy-serverless`**, o AWS CloudFormation atribui um nome à função do Lambda com base no nome da pilha que você especificou durante a implantação. Para dar um nome personalizado à função do Lambda, edite o arquivo `serverless.template` para adicionar uma propriedade `FunctionName` ao recurso `AWS::Serverless::Function`. Para saber mais, consulte Tipo de nome no Guia do usuário do AWS CloudFormation.

Implantar APIs mínimas do ASP.NET no Lambda

Para implantar uma API mínima do ASP.NET no Lambda, você pode usar os modelos do .NET Lambda para criar um novo projeto de API mínimo. Use o comando a seguir para inicializar um novo projeto de API mínima. Neste exemplo, denominamos o projeto `MinimalApiOnLambda`.

```
dotnet new serverless.AspNetCoreMinimalAPI -n MinimalApiOnLambda
```

O comando cria os arquivos e diretórios a seguir no diretório do seu projeto.

```
### MinimalApiOnLambda
    ### src
        ### MinimalApiOnLambda
            ### Controllers
            #   ### CalculatorController.cs
            ### MinimalApiOnLambda.csproj
            ### Program.cs
            ### Readme.md
            ### appsettings.Development.json
            ### appsettings.json
            ### aws-lambda-tools-defaults.json
            ### serverless.template
```

O arquivo `Program.cs` contém o código a seguir.

```
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();
```

Para configurar a API mínima para ser executada no Lambda, pode ser necessário editar esse código para que as solicitações e respostas entre o Lambda e o ASP.NET Core sejam traduzidas adequadamente. Por padrão, a função é configurada para uma fonte de eventos API REST. Para uma API HTTP ou um Application Load Balancer, substitua `(LambdaEventSource.RestApi)` por uma das seguintes opções:

- `(LambdaEventSource.HttpAPi)`
- `(LambdaEventSource.ApplicationLoadBalancer)`

Para implantar a API mínima no Lambda, execute os comandos a seguir para navegar até o diretório que contém o arquivo de código-fonte e implantar a função usando o AWS CloudFormation.

```
cd MinimalApiOnLambda/src/MinimalApiOnLambda
dotnet lambda deploy-serverless
```

Implantar funções do Lambda em .NET com imagens de contêiner

Existem três maneiras de criar uma imagem de contêiner para uma função do Lambda em .NET:

- Usar uma imagem base da AWS para .NET

  As imagens base da AWS são pré-carregadas com um runtime de linguagem, um cliente de interface de runtime para gerenciar a interação entre o Lambda e o código da sua função e um emulador de interface de runtime para testes locais.

- Usar uma imagem base somente para sistema operacional da AWS

  As imagens base somente para sistema operacional da AWS contêm uma distribuição do Amazon Linux e o emulador de interface de runtime. Essas imagens são comumente usadas para criar imagens de contêiner para linguagens compiladas, como Go e Rust e para uma linguagem ou versão de linguagem para a qual o Lambda não fornece uma imagem base, como Node.js 19. Você também pode usar imagens base somente para sistema operacional para implementar um runtime personalizado. Para tornar a imagem compatível com o Lambda, você deve incluir o cliente de interface de runtime do .NET na imagem.

- Usar uma imagem base que não é da AWS

  Você também pode usar uma imagem base alternativa de outro registro de contêiner, como Alpine Linux ou Debian. Você também pode usar uma imagem personalizada criada por sua organização. Para tornar a imagem compatível com o Lambda, você deve incluir o cliente de interface de runtime do .NET na imagem.

> **Tip**
>
> Para reduzir o tempo necessário para que as funções do contêiner do Lambda se tornem ativas, consulte Use multi-stage builds na documentação do Docker. Para criar imagens de contêiner eficientes, siga as Melhores práticas para gravar Dockerfiles.

Esta página explica como criar, testar e implantar imagens de contêiner para o Lambda.

Tópicos

- Usar uma imagem base alternativa com o cliente da interface de runtime

Imagens base da AWS para .NET

A AWS oferece as seguintes imagens base para .NET:

Tags	Runtime	Sistema operacional	Dockerfile	Desaprovação
8	.NET 8	Amazon Linux 2023	Dockerfile para .NET 8 no GitHub	
6	.NET 6	Amazon Linux 2	Dockerfile para .NET 6 no GitHub	12 de novembro de 2024

Repositório do Amazon ECR: gallery.ecr.aws/lambda/dotnet

Usar uma imagem base da AWS para .NET

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- .NET SDK: as etapas a seguir usam a imagem base do .NET 8. Certifique-se de que sua versão do .NET corresponda à versão da imagem base que você especifica em seu Dockerfile.
- Docker

Criação e implantação de uma imagem usando uma imagem base

Nas etapas a seguir, você usará Amazon.Lambda.Templates e Amazon.Lambda.Tools para criar um projeto.NET. Em seguida, você construirá uma imagem do Docker, fará upload da imagem no Amazon ECR e a implantará em uma função do Lambda.

1. Instale o pacote NuGet Amazon.Lambda.Templates.

```
dotnet new install Amazon.Lambda.Templates
```

2. Crie um projeto .NET usando o modelo `lambda.image.EmptyFunction`.

```
dotnet new lambda.image.EmptyFunction --name MyFunction --region us-east-1
```

3. Navegue até o diretório *MyFunction*`/src/`*MyFunction*. É aqui o local em que os arquivos do projeto são armazenados. Examine os seguintes arquivos:

   - aws-lambda-tools-defaults.json: este arquivo é onde você especifica as opções de linha de comando ao implantar sua função do Lambda.
   - Function.cs: o código da função do manipulador do Lambda. Ele é um modelo em C# que inclui a biblioteca `Amazon.Lambda.Core` padrão e um atributo `LambdaSerializer` padrão. Para obter mais informações sobre os requisitos e as opções de serialização, consulte Serialização em funções do Lambda. É possível usar o código fornecido para testes ou substituí-lo pelo seu.
   - MyFunction.csproj: um arquivo de projeto .NET que lista os arquivos e conjuntos que compõem sua aplicação.
   - Readme.md: este arquivo contém mais informações sobre a função do Lambda de exemplo.

4. Examine o Dockerfile no diretório `src/`*MyFunction*. É possível usar o Dockerfile fornecido para testes ou substituí-lo pelo seu. Se você usar o seu próprio, certifique-se de:

   - Definir a propriedade `FROM` como o URI da imagem base. Sua versão do .NET corresponda à versão da imagem base.
   - Definir o argumento `CMD` para o manipulador de funções do Lambda. Isso deve corresponder ao `image-command` em `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. Instale a .NET Global Tool do Amazon.Lambda.Tools.

```
dotnet tool install -g Amazon.Lambda.Tools
```

Se o Amazon.Lambda.Tools já estiver instalado, certifique-se de ter a versão mais recente.

```
dotnet tool update -g Amazon.Lambda.Tools
```

6. Altere o diretório para *MyFunction*/src/*MyFunction*, caso você ainda não esteja lá.

```
cd src/MyFunction
```

7. Use o Amazon.Lambda.Tools para construir a imagem do Docker, enviá-la para um novo repositório do Amazon ECR e implantar a função do Lambda.

   Em `--function-role`, especifique o nome do perfil, não o nome do recurso da Amazon (ARN), do perfil de execução da função. Por exemplo, `lambda-role`.

```
dotnet lambda deploy-function MyFunction --function-role lambda-role
```

   Para obter mais informações sobre a ferramenta do Amazon.Lambda.Tools do .NET Global, consulte o repositório AWS Extensions for .NET CLI no GitHub.

8. Invoque a função.

```
dotnet lambda invoke-function MyFunction --payload "Testing the function"
```

   Se tudo for bem-sucedido, você verá o seguinte:

```
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. Exclua a função do Lambda.

```
dotnet lambda delete-function MyFunction
```

Usar uma imagem base alternativa com o cliente da interface de runtime

Se você usar uma imagem base somente para sistema operacional ou uma imagem base alternativa, deverá incluir o cliente de interface de runtime na imagem. O cliente de interface de runtime estende API de tempo de execução do Lambda, que gerencia a interação entre o Lambda e o código da sua função.

O exemplo a seguir demonstra como criar uma imagem de contêiner para .NET usando uma imagem base que não é da AWS e como adicionar o pacote Amazon.Lambda.RuntimeSupport, que é o cliente da interface de runtime do Lambda para .NET. O Dockerfile do exemplo usa a imagem base do Microsoft .NET 8.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- .NET SDK: as etapas a seguir usam uma imagem base do .NET 8. Certifique-se de que sua versão do .NET corresponda à versão da imagem base que você especifica em seu Dockerfile.
- Docker

Criar e implantar uma imagem usando uma imagem base alternativa

1. Instale o pacote NuGet Amazon.Lambda.Templates.

   ```
   dotnet new install Amazon.Lambda.Templates
   ```

2. Crie um projeto .NET usando o modelo `lambda.CustomRuntimeFunction`. Esse modelo inclui o pacote Amazon.Lambda.RuntimeSupport.

   ```
   dotnet new lambda.CustomRuntimeFunction --name MyFunction --region us-east-1
   ```

3. Navegue até o diretório *MyFunction*`/src/`*MyFunction*. É aqui o local em que os arquivos do projeto são armazenados. Examine os seguintes arquivos:

   - aws-lambda-tools-defaults.json: este arquivo é onde você especifica as opções de linha de comando ao implantar sua função do Lambda.
   - Function.cs: o código contém uma classe com um método `Main` que inicializa a biblioteca `Amazon.Lambda.RuntimeSupport` como bootstrap. O método `Main` é o ponto de entrada para o processo da função. O método `Main` ajusta o manipulador da função em um

wrappper com o qual o bootstrap pode trabalhar. Para obter mais informações, consulte Usar Amazon.Lambda.RuntimeSupport como uma biblioteca de classes no repositório do GitHub.

- MyFunction.csproj: um arquivo de projeto .NET que lista os arquivos e conjuntos que compõem sua aplicação.
- Readme.md: este arquivo contém mais informações sobre a função do Lambda de exemplo.

4. Abra o arquivo `aws-lambda-tools-defaults.json` e adicione as seguintes linhas:

```
  "package-type": "image",
  "docker-host-build-output-dir": "./bin/Release/lambda-publish"
```

- package-type: define o pacote de implantação como uma imagem de contêiner.
- docker-host-build-output-dir: define o diretório de saída para o processo de compilação.

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. Crie um Dockerfile no diretório *MyFunction*`/src/`*MyFunction*. O Dockerfile de exemplo a seguir usa uma imagem base do Microsoft.NET em vez de uma imagem base da AWS.

- Defina a propriedade `FROM` como o identificador da imagem base. Sua versão do .NET corresponda à versão da imagem base.
- Use o comando `COPY` para copiar a função para o diretório `/var/task`.
- Defina o `ENTRYPOINT` como o módulo em que você deseja que o contêiner do Docker seja executado quando for iniciado. Nesse caso, o módulo é o bootstrap, que inicializa a biblioteca `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. Instale a extensão .NET Global Tools Amazon.Lambda.Tools.

```
dotnet tool install -g Amazon.Lambda.Tools
```

Se o Amazon.Lambda.Tools já estiver instalado, certifique-se de ter a versão mais recente.

```
dotnet tool update -g Amazon.Lambda.Tools
```

7. Use o Amazon.Lambda.Tools para criar a imagem do Docker, enviá-la para um novo repositório do Amazon ECR e implantar a função do Lambda.

   Em `--function-role`, especifique o nome do perfil, não o nome do recurso da Amazon (ARN), do perfil de execução da função. Por exemplo, `lambda-role`.

```
dotnet lambda deploy-function MyFunction --function-role lambda-role
```

Para obter mais informações sobre a extensão da CLi Amazon.Lambda.Tools .NET, consulte o repositório AWS Extensions for .NET CLI no GitHub.

8. Invoque a função.

```
dotnet lambda invoke-function MyFunction --payload "Testing the function"
```

Se tudo for bem-sucedido, você verá o seguinte:

```
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. Exclua a função do Lambda.

```
dotnet lambda delete-function MyFunction
```

Compilar o código .NET da função do Lambda em um formato de runtime nativo

O .NET 8 oferece suporte à compilação Ahead Of Time (AOT) nativa. Com a AOT nativa, é possível compilar o código da função do Lambda para um formato de runtime nativo, o que elimina a necessidade de compilar o código .NET em runtime. A compilação AOT nativa pode reduzir o tempo de inicialização a frio para funções do Lambda gravadas em .NET. Para obter mais informações, consulte Introdução ao runtime do .NET 8 para AWS Lambda no Blog AWS Compute.

Seções

Runtime do Lambda

Para implantar uma função do Lambda criada com compilação AOT nativa, use o runtime do Lambda do .NET 8 gerenciado. Esse runtime oferece suporte a arquiteturas x86_64 e arm64.

Quando uma função do Lambda .NET é implantada, sua aplicação é compilada em código de linguagem intermediária (IL). Em tempo de execução o compilador just-in-time (JIT) no runtime do Lambda assume o código IL e o compila em código de máquina conforme necessário. Com uma função do Lambda que é compilada antecipadamente com AOT nativa, você compila seu código em código de máquina ao implantar sua função. Assim, você se torna independente do runtime do .NET ou do SDK no runtime do Lambda para compilar seu código antes que ele seja executado.

Uma limitação da AOT é que o código da aplicação deve ser compilado em um ambiente com o mesmo sistema operacional Amazon Linux 2023 (AL2023) usado pelo runtime do .NET 8. A CLI do Lambda .NET fornece funcionalidade para compilar a aplicação em um contêiner do Docker usando uma imagem do AL2023.

Para evitar possíveis problemas de compatibilidade entre arquiteturas, é altamente recomendável compilar o código em um ambiente com a mesma arquitetura de processador configurada para sua

função. Para saber mais sobre as limitações da compilação entre arquiteturas, consulte Compilação cruzada na documentação do Microsoft.NET.

Pré-requisitos

Docker

Para usar a AOT nativa, o código da função deve ser compilado em um ambiente com o mesmo sistema operacional AL2023 que o runtime do .NET 8. Os comandos da CLI do .NET nas seções a seguir usam o Docker para desenvolver e compilar funções do Lambda em um ambiente do AL2023.

SDK do .NET 8

A compilação AOT nativa é um recurso do .NET 8. É necessário instalar o SDK do .NET 8 no computador de compilação, e não somente o runtime.

Amazon.Lambda.Tools

Para criar as funções do Lambda, você usa a extensão .NET Global Tools do Amazon.Lambda.Tools. Para instalar Amazon.Lambda.Tools, execute o comando a seguir:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Para obter mais informações sobre a extensão Amazon.Lambda.Tools da CLI do .NET, consulte o repositório AWS Extensions for .NET CLI no GitHub.

Amazon.Lambda.Templates

Para gerar o código da função do Lambda, use o Pacote do NuGet Amazon.Lambda.Templates. Para instalar esse pacote de modelo, execute o comando a seguir:

```
dotnet new install Amazon.Lambda.Templates
```

Conceitos básicos

Tanto a CLI Global do .NET quanto o AWS Serverless Application Model (AWS SAM) fornecem modelos de introdução para criar aplicações usando a AOT nativa. Para criar sua primeira função do Lambda da AOT nativa, execute as etapas nas seguintes instruções.

Para inicializar e implantar uma função do Lambda compilada na AOT nativa

1. Inicialize um novo projeto usando o modelo da AOT nativa e, em seguida, navegue até o diretório que contém os arquivos `.cs` e `.csproj` criados. Neste exemplo, nossa função tem o nome `NativeAotSample`.

```
dotnet new lambda.NativeAOT -n NativeAotSample
cd ./NativeAotSample/src/NativeAotSample
```

   O arquivo `Function.cs` criado pelo modelo de AOT nativa contém o seguinte código de função.

```
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
```

A AOT nativa compila a aplicaçãp em um único binário nativo. O ponto de entrada desse binário é o método `static Main`. No `static Main`, o runtime do Lambda é inicializado e o método `FunctionHandler` é configurado. Como parte da inicialização do runtime, um serializador gerado pela fonte é configurado usando `new SourceGeneratorLambdaJsonSerializer<LambdaFunctionJsonSerializerContext>()`

2. Para implantar a aplicação no Lambda, certifique-se de que o Docker esteja sendo executado em seu ambiente local e execute o comando a seguir.

```
dotnet lambda deploy-function
```

Nos bastidores, a CLI global do .NET baixa uma imagem do Docker do AL2023 e compila o código da aplicação em um contêiner em execução. O binário compilado é enviado de volta para o sistema de arquivos local antes de ser implantado no Lambda.

3. Teste a função executando o comando a seguir. Substitua `<FUNCTION_NAME>` pelo nome que você escolheu para a função no assistente de implantação.

```
dotnet lambda invoke-function <FUNCTION_NAME> --payload "hello world"
```

A resposta da CLI inclui detalhes de desempenho da inicialização a frio (duração da inicialização) e do tempo total de execução da invocação da função.

4. Para excluir os recursos do AWS que você criou seguindo as etapas anteriores, execute o comando a seguir. Substitua `<FUNCTION_NAME>` pelo nome que você escolheu para a função no assistente de implantação. Excluindo os recursos da AWS que não está mais usando, você evita que encargos desnecessários sejam cobrados em sua Conta da AWS.

```
dotnet lambda delete-function <FUNCTION_NAME>
```

Serialização

Para implantar funções no Lambda usando a AOT nativa, seu código de função deve usar serialização gerada pela fonte. Em vez de usar a reflexão de runtime para reunir os metadados necessários para acessar as propriedades do objeto para serialização, os geradores de fonte geram arquivos fonte em C# que são compilados quando você cria a aplicação. Para configurar corretamente o serializador gerado pela fonte, certifique-se de incluir todos os objetos de entrada e saída que a função usa, bem como todo os tipos personalizados. Por exemplo, uma função do Lambda que recebe eventos de uma API REST do API Gateway e retorna um tipo de `Product` personalizado incluiria um serializador definido da seguinte forma.

```
[JsonSerializable(typeof(APIGatewayProxyRequest))]
[JsonSerializable(typeof(APIGatewayProxyResponse))]
[JsonSerializable(typeof(Product))]
public partial class CustomSerializer : JsonSerializerContext
{
}
```

Remoção

A AOT nativa remove o código da aplicação como parte da compilação para garantir que o binário seja o menor possível. O.NET 8 para Lambda fornece suporte aprimorado à remoção em comparação com as versões anteriores do .NET. Suporte foi adicionado às bibliotecas de runtime do Lambda, ao AWS SDK para .NET, ao .NET Lambda Annotations e ao próprio .NET 8.

Essas melhorias oferecem o potencial de eliminar os avisos de remoção em tempo de compilação, mas nunca será totalmente seguro remover o .NET. Isso significa que partes das bibliotecas das quais a função depende podem ser removidas como parte da etapa de compilação. É possível gerenciar isso definindo `TrimmerRootAssemblies` como parte do seu arquivo `.csproj`, conforme mostrado no exemplo a seguir.

```
<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>
```

Observe que, quando você recebe um aviso de remoção, adicionar a classe que gera o aviso a `TrimmerRootAssembly` pode não resolver o problema. Um aviso de remoção indica que a classe está tentando acessar alguma outra classe que não pode ser determinada até o tempo de execução. Para evitar erros de tempo de execução, adicione essa segunda classe a `TrimmerRootAssembly`.

Para saber mais sobre como gerenciar avisos de remoção, consulte Introdução aos avisos de remoção na documentação do Microsoft .NET.

Solução de problemas

Erro: não há suporte para a compilação nativa entre sistemas operacionais.

> Sua versão da ferramenta global do NET Core para Amazon.Lambda.Tools está desatualizada. Atualize para a versão mais recente e tente novamente.

Docker: o sistema operacional de imagem “linux” não pode ser usado nesta plataforma.

> O Docker está configurado para usar contêineres do Windows em seu sistema. Alterne para contêineres do Linux para executar o ambiente de compilação AOT nativo.

Para obter mais informações sobre erros comuns, consulte o repositório AWS NativeAOT para .NET no GitHub.

Objeto de contexto do AWS Lambda em C#

Quando o Lambda executa a função, ele transmite um objeto de contexto para o handler. Esse objeto fornece propriedades com informações sobre a invocação, a função e o ambiente de execução.

Propriedades de contexto

- `FunctionName`: o nome da função do Lambda.
- `FunctionVersion`: a versão da função.
- `InvokedFunctionArn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `MemoryLimitInMB`: a quantidade de memória alocada para a função.
- `AwsRequestId`: o identificador da solicitação de invocação.
- `LogGroupName`: o grupo de logs da função.
- `LogStreamName`: a transmissão de log para a instância da função.
- `RemainingTime` (`TimeSpan`): o número de milissegundos restantes antes do tempo limite da execução.
- `Identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
- `ClientContext`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.
- `Logger` O objeto logger para a função.

Você pode usar a informação no objeto do `ILambdaContext` para gerar informações sobre a invocação da função para fins de monitoramento. O código a seguir fornece um exemplo de como adicionar informações de contexto a uma estrutura de log estruturada. Neste exemplo, a função adiciona `AwsRequestId` às saídas do log. A função também usa a propriedade `RemainingTime` para cancelar uma tarefa em execução se o tempo limite da função do Lambda estiver prestes a ser atingido.

```
[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)
            };
        }
    }
}
```

Registro em log da função do Lambda em C#

O AWS Lambda monitora automaticamente as funções do Lambda e envia entradas de logs para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente de runtime do Lambda envia detalhes sobre cada invocação e outras saídas do código da função para o fluxo de logs. Para obter mais informações sobre o CloudWatch Logs, consulte Usar logs do Amazon CloudWatch com o AWS Lambda.

Seções

Criar uma função que retorna logs

Para gerar os logs do código de função, você pode usar métodos na classe Console ou qualquer biblioteca de logs que grave no `stdout` ou no `stderr`.

O runtime do .NET registra em log as linhas `START`, `END` e `REPORT` para cada invocação. A linha do relatório fornece os detalhes a seguir.

RELATAR campos de dados de linha

- RequestId: o ID de solicitação exclusivo para a invocação.
- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.

- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o runtime levou para carregar a função e executar o código fora do método do handler.
- XRAY TraceId: para solicitações rastreadas, o ID de rastreamento do AWS X-Ray.
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Ferramentas e bibliotecas

O Powertools para AWS Lambda (.NET) é um kit de ferramentas para desenvolvedores para implementar práticas recomendadas de tecnologia sem servidor e aumentar a velocidade do desenvolvedor. O utilitário de logs fornece um registrador de logs otimizado para Lambda que inclui informações adicionais sobre o contexto da função em todas as suas funções, com saída estruturada como JSON. Use esse utilitário para fazer o seguinte:

- Capturar campos-chave do contexto do Lambda, inicialização a frio e estruturas registrando em log a saída como JSON
- Registrar em log eventos de invocação do Lambda quando instruído (desativado por padrão)
- Imprimir todos os logs somente para uma porcentagem das invocações por meio da amostragem de logs (desativado por padrão)
- Anexar chaves adicionais ao log estruturado a qualquer momento
- Use um formatador de log personalizado (Bring Your Own Formatter) para gerar logs em uma estrutura compatível com o RFC de logs da sua organização

Uso do Powertools para AWS Lambda (.NET) e do AWS SAM para registro em log estruturado

Siga as etapas abaixo para baixar, criar e implantar um exemplo de aplicação C# Hello World com os módulos integrados do Powertools para AWS Lambda (.NET) usando o AWS SAM. Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET ao endpoint do API Gateway, a função do Lambda invoca, envia logs e métricas usando o formato de métricas incorporadas ao CloudWatch e envia rastreamentos ao AWS X-Ray. A função retorna uma mensagem `hello world`.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- .NET 6 ou .NET 8
- AWS CLI versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS SAM

1. Inicialize a aplicação usando o modelo Hello World TypeScript.

```
sam init --app-template hello-world-powertools-dotnet --name sam-app --package-type
 Zip --runtime dotnet6 --no-tracing
```

2. Crie a aplicação.

```
cd sam-app && sam build
```

3. Implante o aplicativo.

```
sam deploy --guided
```

4. Siga as instruções na tela. Para aceitar as opções padrão fornecidas na experiência interativa, pressione `Enter`.

> **Note**
>
> Em HelloWorldFunction pode não ter autorização definida, tudo bem?, certifique-se de inserir `y`.

5. Obtenha o URL da aplicação implantada:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Invoque o endpoint da API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Se tiver êxito, você verá esta resposta:

```
{"message":"hello world"}
```

7. Para obter os logs da função, execute sam logs. Para obter mais informações, consulte Trabalhar com logs no Guia do desenvolvedor do AWS Serverless Application Model.

```
sam logs --stack-name sam-app
```

O resultado de saída do log se parece com:

```
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. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
sam delete
```

Gerenciar a retenção de logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs por tempo indeterminado, exclua o grupo de logs ou configure um período de retenção após o qual o CloudWatch excluirá os logs automaticamente. Para configurar a retenção de logs, adicione o seguinte ao seu modelo do 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
```

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda.

Se seu código puder ser testado no editor de Código incorporado, você encontrará logs nos resultados de execução. Ao usar o recurso de teste do console para invocar uma função, você encontrará Saída de log na seção Detalhes.

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a página Log groups (Grupos de log).
2. Escolha o grupo de logs de sua função (/aws/lambda/***nome-de-sua-função***).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma instância da sua função. Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos intrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- AWS CLI – Configuração rápida com `aws configure`

Você pode usar a AWS CLI para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

A seguinte saída deverá ser mostrada:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `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
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

A seguinte saída deverá ser mostrada:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e Ubuntu no Windows. Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa `sed` para remover as aspas do arquivo de saída e fica inativo por 15 segundos para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 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 macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod -R 755 get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
}
```


Excluir logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ouConfigurar um período de retençãoapós o qual os logs são excluídos automaticamente.

Instrumentar o código C # no AWS Lambda

O Lambda se integra ao AWS X-Ray para ajudar você a rastrear, depurar e otimizar aplicações do Lambda. É possível usar o X-Ray para rastrear uma solicitação enquanto ela atravessa recursos na aplicação, o que pode incluir funções Lambda e outros produtos da AWS.

Para enviar dados de rastreamento ao X-Ray, é possível usar uma das três bibliotecas SDK:

- AWS Distro for OpenTelemetry (ADOT): uma distribuição segura, pronta para produção e com suporte na AWS do SDK OpenTelemetry (OTel).
- AWS X-Ray SDK for .NET: um SDK para geração e envio de dados de rastreamento ao X-Ray.
- Powertools para AWS Lambda (.NET): um kit de ferramentas para desenvolvedores para implementar práticas recomendadas de tecnologia sem servidor e aumentar a velocidade do desenvolvedor.

Cada um dos SDKs oferece maneiras de enviar dados de telemetria ao serviço do X-Ray. Em seguida, é possível usar o X-Ray para visualizar, filtrar e obter insights sobre as métricas de performance da aplicação para identificar problemas e oportunidades de otimização.

> **Important**
>
> Os SDKs do X-Ray e do Powertools para AWS Lambda fazem parte de uma solução de instrumentação totalmente integrada oferecida pela AWS. As camadas do Lambda para ADOT fazem parte de um padrão em todo o setor para instrumentação de rastreamento que coleta mais dados em geral, mas pode não ser adequado para todos os casos de uso. É possível implementar o rastreamento de ponta a ponta no X-Ray usando ambas as soluções. Para saber mais sobre como escolher entre elas, consulte Como escolher entre os SDKs do AWS Distro para OpenTelemetry e do X-Ray.

Seções

- Ativar o rastreamento com o AWS CloudFormation
- Interpretar um rastreamento do X-Ray

Uso do Powertools para AWS Lambda (.NET) e do AWS SAM para rastreamento

Siga as etapas abaixo para baixar, criar e implantar um exemplo de aplicação C# Hello World com os módulos integrados do Powertools para AWS Lambda (.NET) usando o AWS SAM. Esta aplicação implementa um back-end de API básico e usa o Powertools para emitir logs, métricas e rastreamentos. Consiste em um endpoint do Amazon API Gateway e uma função do Lambda. Quando você envia uma solicitação GET ao endpoint do API Gateway, a função do Lambda invoca, envia logs e métricas usando o formato de métricas incorporadas ao CloudWatch e envia rastreamentos ao AWS X-Ray. A função retorna uma mensagem de hello world.

Pré-requisitos

Para executar as etapas desta seção, você deve ter o seguinte:

- .NET 6 ou .NET 8
- AWS CLI versão 2
- AWS SAM CLI versão 1.75 ou posterior. Se você tiver uma versão mais antiga da CLI do AWS SAM, consulte Atualizando a CLI do AWS SAM.

Implantar uma aplicação de exemplo do AWS SAM

1. Inicialize a aplicação usando o modelo Hello World TypeScript.

```
sam init --app-template hello-world-powertools-dotnet --name sam-app --package-type
 Zip --runtime dotnet6 --no-tracing
```

2. Crie a aplicação.

```
cd sam-app && sam build
```

3. Implante o aplicativo.

```
sam deploy --guided
```

4. Siga as instruções na tela. Para aceitar as opções padrão fornecidas na experiência interativa, pressione `Enter`.

> **Note**
>
> Em HelloWorldFunction pode não ter autorização definida, tudo bem?, certifique-se de inserir `y`.

5. Obtenha o URL da aplicação implantada:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Invoque o endpoint da API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Se tiver êxito, você verá esta resposta:

```
{"message":"hello world"}
```

7. Para obter os rastreamentos da função, execute sam traces.

```
sam traces
```

A saída de rastreamento se parece com:

```
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. Este é um endpoint de API pública que é acessado pela Internet. Recomendamos excluir o endpoint após o teste.

```
sam delete
```

O X-Ray não rastreia todas as solicitações para sua aplicação. O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento seja eficiente, enquanto ainda fornece uma amostra representativa das solicitações. A taxa de amostragem é uma solicitação por segundo e 5% de solicitações adicionais.

> **Note**
>
> Não é possível configurar a taxa de amostragem do X-Ray para suas funções.

Uso do SDK do X-Ray para instrumentar suas funções do .NET

É possível instrumentar seu código de função para gravar metadados e rastrear chamadas downstream. Para registrar detalhes sobre as chamadas que a função faz para outros recursos e serviços, use o AWS X-Ray SDK for .NET. Para obter o SDK, adicione os pacotes `AWSXRayRecorder` ao arquivo do projeto.

```
<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>
```

Há uma variedade de pacotes Nuget que fornecem auto-instrumentação automática para os SDKs da AWS, Entity Framework e solicitações HTTP. Para ver o conjunto completo de opções de configuração, consulte AWS X-Ray SDK for .NET no AWS X-Ray Developer Guide.

Depois de adicionar os pacotes Nuget desejados, configure a instrumentação automática. A prática recomendada é realizar essa configuração fora da função de manipulador da função. Isso permite que você aproveite a reutilização do ambiente de execução para melhorar o desempenho da função. No exemplo de código a seguir, o método `RegisterXRayForAllServices` é chamado no construtor da função para adicionar instrumentação a todas as chamadas do SDK da 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)
        };
    }
}
```

Ativar o rastreamento com o console do Lambda

Para alternar o rastreamento ativo na sua função do Lambda usando o console, siga as etapas abaixo:

Para ativar o rastreamento ativo

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.

3. Escolha Configuration (Configuração) e depois Monitoring and operations tools (Ferramentas de monitoramento e operações).
4. Selecione a opção Editar.
5. Em X-Ray, ative a opção Active tracing (Rastreamento ativo).
6. Escolha Salvar.

Ativar o rastreamento com a API do Lambda

Configure o rastreamento na sua função do Lambda com aAWS CLI ou o AWS SDK, usando as seguintes operações de API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration \
--function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Ativar o rastreamento com o AWS CloudFormation

Para ativar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example function-inline.yml: configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
```

```
      ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example template.yml: configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretar um rastreamento do X-Ray

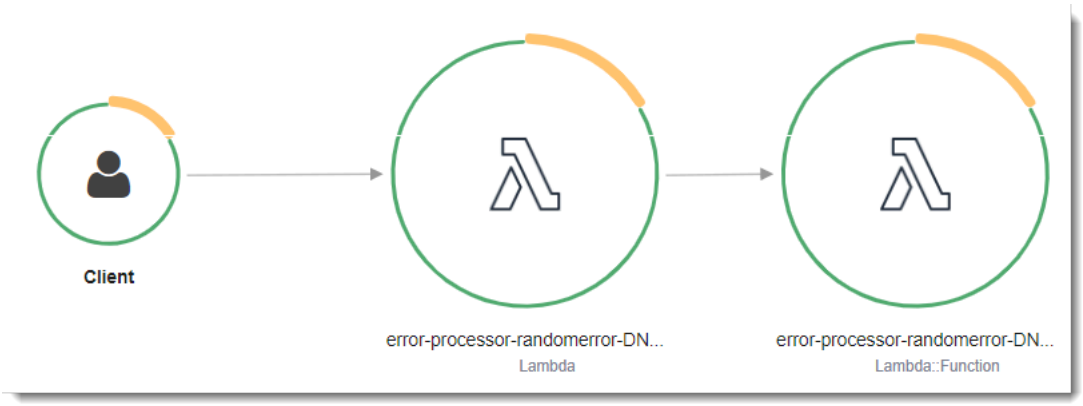
Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você ativa o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à função de execução da função. Caso contrário, adicione a política AWSXRayDaemonWriteAccess à função de execução.

Após configurar o rastreamento ativo, você pode observar solicitações específicas por meio da aplicação. O grafo de serviço do X-Ray exibe informações sobre sua aplicação e todos os componentes. A imagem a seguir demonstra uma aplicação com duas funções. A função principal processa eventos e, às vezes, retorna erros. A segunda função de cima para baixo processa erros que aparecem no primeiro grupo de logs e usa o AWS SDK para chamar o X-Ray, o Amazon Simple Storage Service (Amazon S3) e o Amazon CloudWatch Logs.

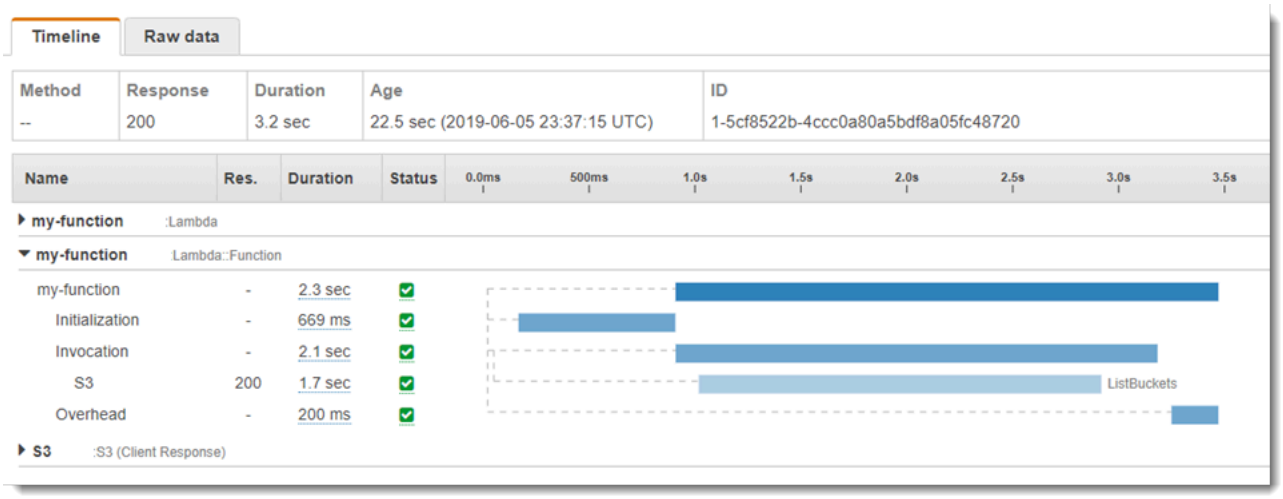
O X-Ray não rastreia todas as solicitações para sua aplicação. O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento seja eficiente, enquanto ainda fornece uma amostra representativa das solicitações. A taxa de amostragem é uma solicitação por segundo e 5% de solicitações adicionais.

> **Note**
>
> Não é possível configurar a taxa de amostragem do X-Ray para suas funções.

No X-Ray, um rastreamento registra informações sobre uma solicitação que é processada por um ou mais serviços. O Lambda registra dois segmentos por rastreamento, o que cria dois nós no gráfico de serviços. A imagem a seguir destaca esses dois nós:



O primeiro nó à esquerda representa o serviço do Lambda, que recebe a solicitação de invocação. O segundo nó representa a sua função do Lambda específica. O exemplo a seguir mostra um rastreamento com esses dois segmentos. Ambos têm o nome my-function, mas um tem a origem `AWS::Lambda` e o outro, a origem `AWS::Lambda::Function`. Se o segmento `AWS::Lambda` mostrar um erro, o serviço Lambda teve um problema. Se o segmento `AWS::Lambda::Function` mostrar um erro, sua função teve um problema.



Este exemplo expande o segmento `AWS::Lambda::Function` para mostrar seus três subsegmentos:

- Inicialização: representa o tempo gasto carregando a função e executando o código de inicialização. Esse subsegmento aparece somente para o primeiro evento que cada instância da função processa.
- Invocação: representa o tempo gasto na execução do código do manipulador.
- Sobrecarga: representa o tempo gasto pelo runtime do Lambda preparando-se para lidar com o próximo evento.

Você também pode instrumentar clientes HTTP, registrar consultas SQL e criar subsegmentos personalizados com anotações e metadados. Para obter mais informações, consulte AWS X-Ray SDK for .NET no Guia do desenvolvedor do AWS X-Ray.

> **Definição de preço**
>
> Você pode usar o rastreamento do X-Ray gratuitamente todos os meses até determinado limite como parte do nível gratuito da AWS. Além do limite, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter mais informações, consulte Preços do AWS X-Ray.

Teste da função do AWS Lambda em C#

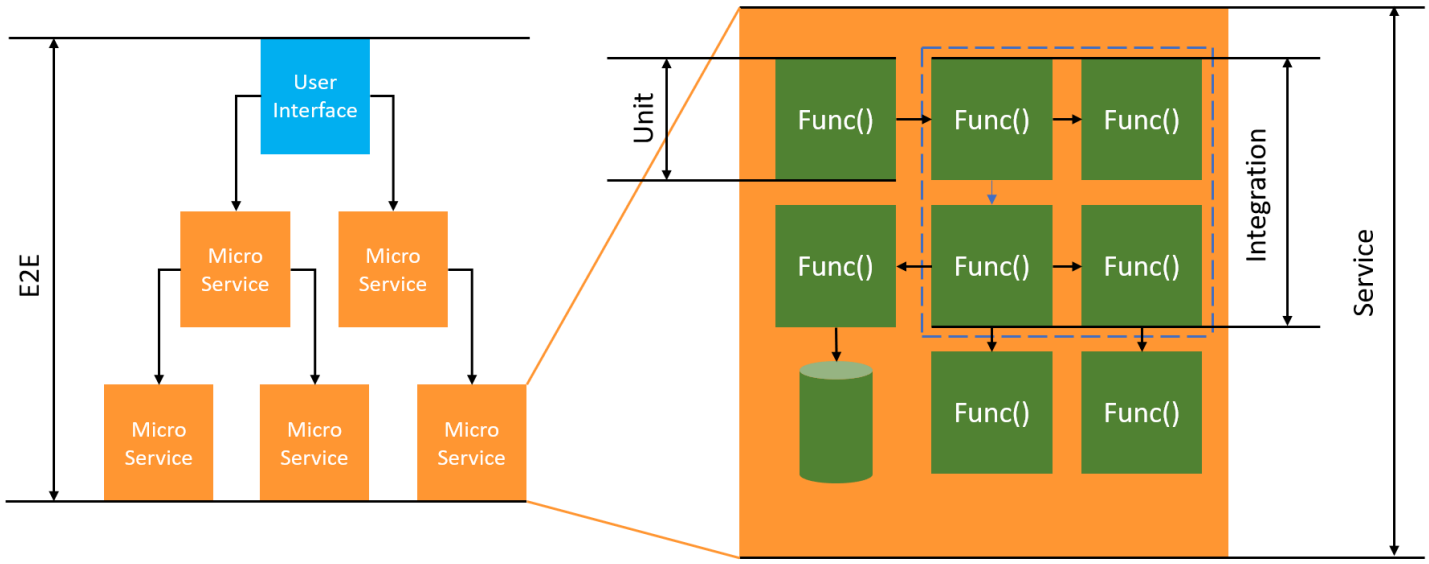
> **Note**
>
> Consulte o capítulo Testar funções para obter uma introdução completa às técnicas e melhores práticas para testar soluções com tecnologia sem servidor.

Os testes de funções com tecnologia sem servidor usam tipos e técnicas de teste tradicionais, mas você também deve considerar testar aplicações com tecnologia sem servidor como um todo. Os testes baseados em nuvem fornecerão a medida mais precisa da qualidade das suas funções e aplicações com tecnologia sem servidor.

Uma arquitetura de aplicações com tecnologia sem servidor inclui serviços gerenciados que fornecem funcionalidade de aplicações críticas por meio de chamadas de API. Por esse motivo, seu ciclo de desenvolvimento deve incluir testes automatizados que verifiquem a funcionalidade quando sua função e seus serviços interagem.

Se você não criar testes baseados em nuvem, poderá encontrar problemas devido às diferenças entre o ambiente local e o ambiente implantado. Seu processo de integração contínua deve executar testes em comparação com um conjunto de recursos provisionados na nuvem antes de promover seu código para o próximo ambiente de implantação, como GQ, Preparação ou Produção.

Continue lendo este breve guia para aprender sobre estratégias de testes para aplicações com tecnologia sem servidor ou acesse o repositório de exemplos de testes com tecnologia sem servidor para conhecer exemplos práticos, específicos da linguagem e do runtime escolhidos.



Para testes sem servidor, você ainda escreverá testes unitários, de integração e end-to-endde testes.

- Testes unitários: testes executados em comparação com um bloco de código isolado. Por exemplo, verificar a lógica de negócios para calcular a taxa de entrega de acordo com um item e um destino determinado.
- Testes de integração: testes envolvendo dois ou mais componentes ou serviços que interagem, normalmente em um ambiente de nuvem. Por exemplo, a verificação de eventos de processos de uma função em uma fila.
- nd-to-end Testes E - Testes que verificam o comportamento em todo o aplicativo. Por exemplo, garantir que a infraestrutura seja configurada corretamente e que os eventos fluam entre os serviços conforme o esperado para registrar o pedido de um cliente.

Testar suas aplicações com tecnologia sem servidor

Geralmente, você usará uma combinação de abordagens para testar o código da aplicação com tecnologia sem servidor, incluindo testes na nuvem, testes com simulações e, ocasionalmente, testes com emuladores.

Testes na nuvem

O teste na nuvem é valioso para todas as fases do teste, incluindo testes unitários, testes de integração e end-to-end testes. Você executa testes com o código implantado na nuvem e interagindo com serviços baseados na nuvem. Essa abordagem fornece a medida mais precisa da qualidade do código.

Uma maneira conveniente de depurar a função do Lambda na nuvem é com o uso de um evento de teste no console. Um evento de teste é uma entrada JSON para sua função. Se a função não necessitar de entrada, o evento poderá ser um documento JSON vazio (`{}`). O console fornece eventos de exemplo para uma variedade de integrações de serviços. Depois de criar um evento no console, você pode compartilhá-lo com sua equipe para tornar os testes mais fáceis e consistentes.

> **Note**
>
> Testar uma função no console é uma maneira rápida de começar a usar, mas automatizar os ciclos de teste garante a qualidade da aplicação e a velocidade de desenvolvimento.

Ferramentas de teste

Para acelerar seu ciclo de desenvolvimento, há várias ferramentas e técnicas que você pode usar ao testar as funções. Por exemplo, o AWS SAM Accelerate e o modo de observação do AWS CDK diminuem o tempo necessário para atualizar os ambientes de nuvem.

A maneira como você define o código da função do Lambda simplifica a adição de testes de unidade. O Lambda exige um construtor público sem parâmetros para inicializar a classe. A introdução de um segundo construtor interno dá a você controle das dependências que a aplicação usa.

```
[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)
        };
    }
}
```

Para escrever um teste para essa função, você pode inicializar uma nova instância da classe da `Function` e passar por uma implementação simulada do `IDatabaseRepository`. Os exemplos abaixo usam `XUnit`, `Moq` e `FluentAssertions` para escrever um teste simples, garantindo que o `FunctionHandler` retorne de um código de status 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);
    }
}
```

Para obter exemplos mais detalhados, incluindo exemplos de testes assíncronos, consulte o repositório de amostras de testes.NET em. GitHub

Construir funções do Lambda com o PowerShell

As seções a seguir explicam como padrões comuns de programação e conceitos fundamentais são aplicados ao criar o código da função do Lambda no PowerShell.

O Lambda fornece as seguintes aplicações de amostra para PowerShell:

- blank-powershell: uma função do PowerShell que mostra o uso do registro em log, as variáveis de ambiente e o AWS SDK.

Antes de começar, você deve primeiro configurar um ambiente de desenvolvimento do PowerShell. Para obter instruções sobre como fazer isso, consulte Configurando um ambiente PowerShell de desenvolvimento.

Para saber mais sobre como usar o módulo AWSLambdaPSCore para fazer download de projetos de amostra do PowerShell usando modelos, criar pacotes de implantação do PowerShell e implantar funções do PowerShell na Nuvem AWS, consulte Implemente funções PowerShell Lambda com arquivos de arquivos.zip.

O Lambda fornece os seguintes runtimes para linguagens .NET:

.NET

Nome	Identificador	Sistema operacional	Data da substituição	Bloquear a criação de funções	Bloquear a atualização de funções
.NET 8	`dotnet8`	Amazon Linux 2023			
.NET 6	`dotnet6`	Amazon Linux 2	12 de novembro de 2024	28 de fevereiro de 2025	31 de março de 2025

Tópicos

- Configurando um ambiente PowerShell de desenvolvimento
- Implemente funções PowerShell Lambda com arquivos de arquivos.zip

- Definir o manipulador de funções do Lambda no PowerShell
- AWS Lambdaobjeto de contexto em PowerShell
- Registro em log da função do AWS Lambda no PowerShell

Configurando um ambiente PowerShell de desenvolvimento

O Lambda fornece um conjunto de ferramentas e bibliotecas para o PowerShell tempo de execução. Para obter instruções de instalação, consulte Ferramentas Lambda para PowerShell saber mais. GitHub

O AWSLambdaPSCore módulo inclui os seguintes cmdlets para ajudar a criar e publicar funções do Lambda PowerShell :

- Get- AWSPowerShellLambdaTemplate — Retorna uma lista de modelos de introdução.
- Novo- AWSPowerShellLambda — Cria um PowerShell script inicial com base em um modelo.
- Publish- AWSPowerShellLambda — Publica um determinado PowerShell script no Lambda.
- Novo- AWSPowerShellLambdaPackage — Cria um pacote de implantação Lambda que você pode usar em um sistema de CI/CD para implantação.

Implemente funções PowerShell Lambda com arquivos de arquivos.zip

Um pacote de implantação para o PowerShell tempo de execução contém seu PowerShell script, os PowerShell módulos necessários para seu PowerShell script e os assemblies necessários para PowerShell hospedar o Core.

Criar a função do Lambda

Para começar a escrever e invocar um PowerShell script com o Lambda, você pode usar `New-AWSPowerShellLambda` o cmdlet para criar um script inicial com base em um modelo. Você pode usar o cmdlet `Publish-AWSPowerShellLambda` para implantar seu script no Lambda. Em seguida, você pode testar seu script na linha de comando ou no console do Lambda.

Para criar um novo PowerShell script, carregá-lo e testá-lo, faça o seguinte:

1. Execute o seguinte comando para exibir a lista de modelos disponíveis:

```
PS C:\> Get-AWSPowerShellLambdaTemplate

Template               Description
--------               -----------
Basic                  Bare bones script
CodeCommitTrigger      Script to process AWS CodeCommit Triggers
...
```

2. Para criar um script de amostra com base no modelo `Basic`, execute o seguinte comando:

```
New-AWSPowerShellLambda -ScriptName MyFirstPSScript -Template Basic
```

   Um novo arquivo chamado `MyFirstPSScript.ps1` é criado em um novo subdiretório do diretório atual. O nome do diretório é baseado no parâmetro `-ScriptName`. Você pode usar o parâmetro `-Directory` para escolher um diretório alternativo.

   É possível ver que o novo arquivo tem o seguinte conteúdo:

```
# 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. Para ver como as mensagens de log do seu PowerShell script são enviadas para o Amazon CloudWatch Logs, remova o comentário da `Write-Host` linha do script de amostra.

   Para demonstrar como você pode retornar dados das suas funções do Lambda, adicione uma nova linha no final do script com `$PSVersionTable`. Isso adiciona o `$PSVersionTable` ao PowerShell pipeline. Depois que o PowerShell script for concluído, o último objeto no PowerShell pipeline são os dados de retorno para a função Lambda. `$PSVersionTable`é uma variável PowerShell global que também fornece informações sobre o ambiente em execução.

   Depois de fazer essas alterações, as duas últimas linhas do script de amostra terão esta aparência:

   ```
   Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
   $PSVersionTable
   ```

4. Depois de editar o arquivo `MyFirstPSScript.ps1`, altere o diretório para o local do script. Em seguida, execute o seguinte comando para publicar o script no Lambda:

   ```
   Publish-AWSPowerShellLambda -ScriptPath .\MyFirstPSScript.ps1 -Name
    MyFirstPSScript -Region us-east-2
   ```

   Observe que o parâmetro `-Name` especifica o nome da função do Lambda, que aparece no console do Lambda. Você pode usar essa função para chamar seu script manualmente.

5. Invoque sua função usando o comando `invoke` da AWS Command Line Interface (AWS CLI).

```
> aws lambda invoke --function-name MyFirstPSScript out
```

Definir o manipulador de funções do Lambda no PowerShell

Quando uma função do Lambda é invocada, o manipulador do Lambda chama o script do PowerShell.

Quando o script do PowerShell é invocado, as seguintes variáveis são predefinidas:

- *$LambdaInput* : um PSObject que contém a entrada para o manipulador. Essa entrada pode ser os dados do evento (publicados por uma origem de evento) ou uma entrada personalizada fornecida por você, tal como uma string ou qualquer objeto de dados personalizado.
- *$LambdaContext*: um objeto Amazon.Lambda.Core.ILambdaContext que você pode usar para acessar informações sobre a invocação atual, como o nome da função atual, limite de memória, tempo de execução restante e registro em log.

Por exemplo, considere o código de exemplo em PowerShell a seguir.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host 'Function Name:' $LambdaContext.FunctionName
```

Esse script retorna a propriedade FunctionName que é obtida a partir da variável $LambdaContext.

> **Note**
>
> É necessário usar a instrução `#Requires` nos seus scripts do PowerShell para indicar os módulos dos quais os seus scripts dependem. Essa declaração executa duas tarefas importantes. 1) Ele comunica a outros desenvolvedores quais módulos o script usa e 2) identifica os módulos dependentes que as ferramentas do AWS PowerShell precisam empacotar com o script como parte da implantação. Para obter mais informações sobre a instrução `#Requires` no PowerShell, consulte About requires. Para obter mais informações sobre os pacotes de implantação do PowerShell, consulte Implemente funções PowerShell Lambda com arquivos de arquivos.zip.
>
> Quando a sua função do Lambda para PowerShell usa cmdlets do AWS PowerShell, certifique-se de definir uma instrução `#Requires` que faça referência ao módulo `AWSPowerShell.NetCore`, que oferece suporte ao PowerShell Core e não ao módulo `AWSPowerShell`, que apenas oferece suporte para o Windows PowerShell. Além disso, certifique-se de usar a versão 3.3.270.0 ou mais recente do `AWSPowerShell.NetCore`, que

otimiza o processo de importação de cmdlets. Se você usar uma versão mais antiga, haverá mais partidas a frio. Para obter mais informações, consulte AWS Tools for PowerShell.

Retorno de dados

Algumas invocações do Lambda são destinadas a retornar dados ao chamador. Por exemplo, se uma invocação tiver ocorrido em resposta a uma solicitação do API Gateway, nossa função do Lambda precisará retornar essa resposta. Para o PowerShell Lambda, o último objeto adicionado ao pipeline do PowerShell são os dados de retorno da invocação do Lambda. Se o objeto for uma string, os dados serão retornados no estado em que se encontram. Caso contrário, o objeto será convertido em JSON usando o cmdlet `ConvertTo-Json`.

Por exemplo, considere a seguinte instrução PowerShell, que adiciona `$PSVersionTable` ao pipeline do PowerShell:

```
$PSVersionTable
```

Depois que o script PowerShell estiver finalizado, o último objeto no pipeline do PowerShell será os dados de retorno da função do Lambda. `$PSVersionTable` é uma variável global do PowerShell que também fornece informações sobre o ambiente de execução.

AWS Lambdaobjeto de contexto em PowerShell

Quando o Lambda executa a função, ele transmite informações de contexto, disponibilizando uma variável `$LambdaContext` para o handler. Essa variável fornece métodos e propriedades com informações sobre a invocação, a função e o ambiente de execução.

Propriedades de contexto

- `FunctionName`: o nome da função do Lambda.
- `FunctionVersion`: a versão da função.
- `InvokedFunctionArn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `MemoryLimitInMB`: a quantidade de memória alocada para a função.
- `AwsRequestId`: o identificador da solicitação de invocação.
- `LogGroupName`: o grupo de logs da função.
- `LogStreamName`: a transmissão de log para a instância da função.
- `RemainingTime`: o número de milissegundos restantes antes do tempo limite da execução.
- `Identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
- `ClientContext`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.
- `Logger`: o objeto do logger da função.

O trecho PowerShell de código a seguir mostra uma função de manipulador simples que imprime algumas das informações de contexto.

```
#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
```

Registro em log da função do AWS Lambda no PowerShell

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome e envia logs para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente do runtime do Lambda envia detalhes sobre cada invocação à transmissão de logs e transmite os logs e outras saídas do código de sua função. Para ter mais informações, consulte Usar logs do Amazon CloudWatch com o AWS Lambda.

Esta página descreve como produzir a saída de logs usando o código de sua função do Lambda ou acessar os logs usando a AWS Command Line Interface, o console do Lambda ou o console do CloudWatch.

Seções

Criar uma função que retorna logs

Para os logs de saída do código da função, você pode usar cmdlets no utilitário Microsoft.PowerShell, ou em qualquer módulo de registro em log que grave no `stdout` ou no `stderr`. O exemplo a seguir usa `Write-Host`.

Example function/Handler.ps1: registro em log

```
#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 do 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
```

O runtime do .NET registra em log as linhas START, END e REPORT para cada invocação. A linha do relatório fornece os detalhes a seguir.

RELATAR campos de dados de linha

- RequestId: o ID de solicitação exclusivo para a invocação.

- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.
- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o runtime levou para carregar a função e executar o código fora do método do handler.
- XRAY TraceId: para solicitações rastreadas, o ID de rastreamento do AWS X-Ray.
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda.

Se seu código puder ser testado no editor de Código incorporado, você encontrará logs nos resultados de execução. Ao usar o recurso de teste do console para invocar uma função, você encontrará Saída de log na seção Detalhes.

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a página Log groups (Grupos de log).
2. Escolha o grupo de logs de sua função (/aws/lambda/***nome-de-sua-função***).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma instância da sua função. Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos intrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- AWS CLI – Configuração rápida com `aws configure`

Você pode usar a AWS CLI para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

A seguinte saída deverá ser mostrada:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `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
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-`

`in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

A seguinte saída deverá ser mostrada:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e Ubuntu no Windows. Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa `sed` para remover as aspas do arquivo de saída e fica inativo por 15 segundos para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 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 macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod -R 755 get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
}
```


Excluir logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ouConfigurar um período de retençãoapós o qual os logs são excluídos automaticamente.

Construção de funções do Lambda com Rust

Como o Rust compila em código nativo, você não precisa de um runtime dedicado para executar o código Rust no Lambda. Em vez disso, use o cliente runtime do Rust para criar seu projeto no local e, depois, implante-o no Lambda usando o runtime `provided.al2023` ou `provided.al2`. Ao usar o `provided.al2023` ou o `provided.al2`, o Lambda mantém automaticamente o sistema operacional atualizado com os patches mais recentes.

> **Note**
>
> O cliente runtime do Rust é um pacote experimental. Ele está sujeito a alterações, e é destinado apenas a fins de avaliação.

Ferramentas e bibliotecas para Rust

- AWS SDK para Rust: o AWS SDK para Rust fornece APIs do Rust para interagir com os serviços de infraestrutura da Amazon Web Services.
- Cliente runtime do Rust para Lambda: o cliente runtime do Rust é um pacote experimental. Ele está sujeito a alterações significativas e não é recomendado para uso em produção.
- Cargo Lambda: esta biblioteca fornece uma aplicação de linha de comando para trabalhar com funções do Lambda criadas com o Rust.
- Lambda HTTP: esta biblioteca fornece um wrapper para trabalhar com eventos de HTTP.
- Extensão do Lambda: esta biblioteca fornece suporte para escrever extensões Lambda com o Rust.
- Eventos do AWS Lambda: esta biblioteca fornece definições de tipos para integrações comuns de origens de eventos.

Exemplo de aplicações do Lambda para Rust

- Função do Lambda básica: uma função do Rust que mostra como processar eventos básicos.
- Função do Lambda com tratamento de erros: uma função do Rust que mostra como lidar com erros personalizados do Rust no Lambda.
- Função do Lambda com recursos compartilhados: um projeto do Rust que inicializa recursos compartilhados antes de criar a função do Lambda.

- Eventos de HTTP do Lambda: uma função do Rust que manipula eventos HTTP.
- Eventos de HTTP do Lambda com cabeçalhos CORS: uma função do Rust que usa o Tower para injetar cabeçalhos CORS.
- API REST do Lambda: uma API REST que usa Axum e Diesel para se conectar a um banco de dados PostgreSQL.
- Demonstração do Rust com tecnologia sem servidor: um projeto do Rust que mostra o uso das bibliotecas do Rust do Lambda, registro em log, variáveis e do AWS SDK.
- Extensão do Lambda básica: uma extensão do Rust que mostra como processar eventos básicos de extensão.
- Extensão Lambda Logs do Amazon Data Firehose: uma extensão do Rust que mostra como enviar logs do Lambda para o Firehose.

Tópicos

Definir o manipulador de função do Lambda em Rust

> **Note**
>
> O cliente runtime do Rust é um pacote experimental. Ele está sujeito a alterações, e é destinado apenas a fins de avaliação.

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. A função é executada até que o manipulador retorne uma resposta, seja encerrado ou atinja o tempo limite.

Escreva seu código de função do Lambda como um executável Rust. Implemente o código da função do manipulador e uma função principal e inclua o seguinte:

- A caixa lambda_runtime de crates.io, que implementa o modelo de programação Lambda para Rust.
- Inclua Tokio em suas dependências. O cliente de runtime Rust para Lambda usa o Tokio para lidar com chamadas assíncronas.

Example — Manipulador do Rust que processa eventos JSON

O exemplo a seguir usa a caixa serde_json para processar eventos JSON básicos:

```
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
}
```

Observe o seguinte:

- `use`: importa as bibliotecas que sua função do Lambda exige.
- `async fn main`: o ponto de entrada que executa o código da função do Lambda. O cliente de runtime Rust usa o Tokio como um runtime assíncrono, de modo que você precisa anotar a função principal com `#[tokio::main]`.
- `async fn handler(event: LambdaEvent<Value>) -> Result<Value,Error>`: esta é a assinatura do manipulador do Lambda. Ele inclui o código que será executado quando a função for chamada.
  - `LambdaEvent<Value>`: este é um tipo genérico que descreve o evento recebido pelo runtime do Lambda, bem como o contexto da função Lambda.
  - `Result<Value, Error>`: a função retorna um tipo `Result`. Se a função tiver êxito, o resultado será um valor JSON. Se a função não tiver êxito, o resultado será um erro.

Uso de estado compartilhado

É possível declarar e modificar variáveis globais que são independentes do código do manipulador da função do Lambda. Essas variáveis podem ajudá-lo a carregar informações de estado durante o Fase de inicialização, antes que sua função receba qualquer evento.

Example — Compartilhe o cliente Amazon S3 em todas as instâncias da função

Observe o seguinte:

- `use aws_sdk_s3::Client`: este exemplo exige que você adicione `aws-sdk-s3 = "0.26.0"` à lista de dependências em seu arquivo `Cargo.toml`.
- `aws_config::from_env`: este exemplo exige que você adicione `aws-config = "0.55.1"` à lista de dependências em seu arquivo `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
}
```

Objeto de contexto do Lambda no Rust

> **Note**
>
> O cliente runtime do Rust é um pacote experimental. Ele está sujeito a alterações, e é destinado apenas a fins de avaliação.

Quando o Lambda executa sua função, ele adiciona um objeto de contexto ao LambdaEvent que o manipulador recebe. Esse objeto fornece propriedades com informações sobre a invocação, a função e o ambiente de execução.

Propriedades de contexto

- `request_id`: o ID da solicitação AWS gerado pelo serviço Lambda.
- `deadline`: o prazo de execução da invocação atual, em milissegundos.
- `invoked_function_arn`: o nome do recurso da Amazon (ARN) da função do Lambda sendo invocada.
- `xray_trace_id`: o ID de rastreamento do AWS X-Ray da invocação atual.
- `client_content`: o objeto de contexto do cliente enviado pelo SDK móvel da AWS. Este campo está vazio, a menos que a função seja invocada usando um SDK móvel da AWS.
- `identity`: a identidade do Amazon Cognito que invocou a função. Este campo está vazio, a menos que a solicitação de invocação às APIs do Lambda tenha sido feita usando credenciais da AWS emitidas pelos grupos de identidade do Amazon Cognito.
- `env_config`: a configuração da função do Lambda a partir das variáveis de ambiente locais. Essa propriedade inclui informações como nome da função, alocação de memória, versão e fluxos de log.

Acessar informações do contexto de invocação

As funções do Lambda têm acesso aos metadados sobre seu ambiente e a solicitação da invocação. O objeto `LambaEvent` que seu manipulador de funções recebe inclui os metadados `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
}
```

Processamento de eventos de HTTP com o Rust

> **Note**
>
> O cliente runtime do Rust é um pacote experimental. Ele está sujeito a alterações, e é destinado apenas a fins de avaliação.

APIs do Amazon API Gateway, Application Load Balancers e URLs de função do Lambda podem enviar eventos de HTTP para o Lambda. É possível usar a caixa aws_lambda_events do crates.io para processar eventos dessas origens.

Example — Gerencia a solicitação de proxy da API Gateway

Observe o seguinte:

- `use aws_lambda_events::apigw::{ApiGatewayProxyRequest, ApiGatewayProxyResponse}`: a caixa aws_lambda_events inclui muitos eventos do Lambda. Para reduzir o tempo de compilação, use sinalizadores de recursos para ativar os eventos de que você precisa. Exemplo: `aws_lambda_events = { version = "0.8.3", default-features = false, features = ["apigw"] }`.
- `use http::HeaderMap`: essa importação exige que você adicione a caixa http às suas dependências.

```
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
}
```

O cliente de runtime Rust para Lambda também fornece uma abstração sobre esses tipos de eventos que permite trabalhar com tipos de HTTP nativos, independentemente de qual serviço envia os eventos. O código a seguir é equivalente ao exemplo anterior e funciona imediatamente com URLs de funções do Lambda, Application Load Balancers e API Gateway.

> **Note**
>
> A caixa lambda_http usa a caixa lambda_runtime logo abaixo. Não é necessário importar `lambda_runtime` separadamente.

Example — Trata solicitações de 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
}
```

Para ver outro exemplo de como usar `lambda_http`, consulte o exemplo de código http-axum no repositório do GitHub do AWS Labs.

Exemplos de eventos de HTTP Lambda para Rust

- Eventos de HTTP do Lambda: uma função do Rust que manipula eventos HTTP.
- Eventos de HTTP do Lambda com cabeçalhos CORS: uma função do Rust que usa o Tower para injetar cabeçalhos CORS.
- Eventos de HTTP do Lambda com recursos compartilhados: uma função do Rust que usa recursos compartilhados inicializados antes do manipulador da função ser criado.

Implantar funções do Lambda em Rust com arquivos .zip

> **Note**
>
> O cliente runtime do Rust é um pacote experimental. Ele está sujeito a alterações, e é destinado apenas a fins de avaliação.

Esta página descreve como compilar sua função do Rust e, em seguida, implantar o binário compilado no AWS Lambda usando o Cargo Lambda. Ela também mostra como implantar o binário compilado com a AWS Command Line Interface e a CLI do AWS Serverless Application Model.

Seções

Pré-requisitos

- Rust
- AWS Command Line Interface (AWS CLI) versão 2

Criação de funções do Rust no macOS, Windows ou Linux

As etapas a seguir demonstram como criar o projeto para sua primeira função do Lambda com Rust e como compilá-lo com o Cargo Lambda.

1. Instale o Cargo Lambda, um subcomando do Cargo, que compila as funções do Rust para o Lambda no macOS, Windows e Linux.

   Para instalar o Cargo Lambda em qualquer sistema que tenha o Python 3 instalado, use o pip:

   ```
   pip3 install cargo-lambda
   ```

   Para instalar o Cargo Lambda no macOS ou no Linux, use o Homebrew:

```
brew tap cargo-lambda/cargo-lambda
brew install cargo-lambda
```

Para instalar o Cargo Lambda no Windows, use o Scoop:

```
scoop bucket add cargo-lambda
scoop install cargo-lambda/cargo-lambda
```

Para outras opções, consulte Instalação na documentação do Cargo Lambda.

2. Crie a estrutura do pacote. Esse comando cria um código de função básico em `src/main.rs`. É possível usar esse código para testes ou substituí-lo pelo seu.

```
cargo lambda new my-function
```

3. Dentro do diretório raiz do pacote, execute o subcomando build para compilar o código na sua função.

```
cargo lambda build --release
```

(Opcional) Se você quiser usar o AWS Graviton2 no Lambda, adicione o sinalizador `--arm64` para compilar seu código para CPUs ARM.

```
cargo lambda build --release --arm64
```

4. Antes de implantar sua função do Rust, configure as credenciais da AWS em sua máquina.

```
aws configure
```

Implantação do binário da função do Rust com o Cargo Lambda

Use o subcomando deploy para implantar o binário compilado no Lambda. Esse comando cria um perfil de execução e, em seguida, cria a função do Lambda. Para especificar um perfil de execução existente, use a sinalização --iam-role.

```
cargo lambda deploy my-function
```

Implantação do binário da sua função do Rust com a AWS CLI

Você também pode implantar seu binário com a AWS CLI.

1. Use o subcomando build para compilar o pacote de implantação .zip.

```
cargo lambda build --release --output-format zip
```

2. Implante o pacote .zip no Lambda. Em `--role`, especifique o ARN do perfil de execução.

```
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
```

Implantação do binário da sua função do Rust com a CLI do AWS SAM

Você também pode implantar seu binário com a CLI do AWS SAM.

1. Crie um modelo do AWS SAM com a definição do recurso e da propriedade. Para obter mais informações, consulte AWS::Serverless::Function no AWS Serverless Application ModelGuia do desenvolvedor.

   Example Definição de recursos e propriedades do SAM para um binário do 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. Use o subcomando build para compilar a função.

```
cargo lambda build --release
```

3. Use o comando sam deploy para implantar a função no Lambda.

```
sam deploy --guided
```

Para obter mais informações sobre como construir funções do Rust com a CLI do AWS SAM, consulte Construção de funções do Lambda do Rust com o Cargo Lambda no Guia do desenvolvedor do AWS Serverless Application Model.

Invocação da sua função do Rust com o Cargo Lambda

Use o subcomando invoke para testar sua função com uma carga.

```
cargo lambda invoke --remote --data-ascii '{"command": "Hello world"}' my-function
```

Invocação da sua função Rust com a AWS CLI

Você também pode usar a AWS CLI para invocar a função.

```
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"command": "Hello world"}' /tmp/out.txt
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

Registro em log da função do Lambda no Rust

> **Note**
>
> O cliente runtime do Rust é um pacote experimental. Ele está sujeito a alterações, e é destinado apenas a fins de avaliação.

AWS Lambdamonitora automaticamente as funções do Lambda em seu nome e envia registros para a Amazon. CloudWatch Sua função Lambda vem com um grupo de CloudWatch registros de registros e um fluxo de registros para cada instância de sua função. O ambiente do runtime do Lambda envia detalhes sobre cada invocação à transmissão de logs e transmite os logs e outras saídas do código de sua função. Para ter mais informações, consulte Usar logs do Amazon CloudWatch com o AWS Lambda. Esta página descreve como produzir a saída de logs usando o código de sua função do Lambda.

Criação de uma função que grava logs

Para gerar os logs do código da sua função, é possível usar qualquer função de log que grave em o `stdout` ou `stderr`, como a macro `println!`. O exemplo a seguir usa `println!` para imprimir uma mensagem quando o manipulador da função é iniciado e antes de ele terminar.

```
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
}
```

Registro em log avançado com a caixa Tracing

O Tracing é uma estrutura para instrumentar programas do Rust para coletar informações de diagnóstico estruturadas e baseadas em eventos. Essa estrutura fornece utilitários para personalizar os níveis e formatos de saída de registro, como a criação de mensagens de log JSON estruturadas. Para usar essa estrutura, é necessário inicializar um `subscriber` antes de implementar o manipulador de funções. Em seguida, é possível usar macros de rastreamento como `debug`, `info` e `error` para especificar o nível de log desejado para cada cenário.

Example — Uso da caixa Tracing

Observe o seguinte:

- `tracing_subscriber::fmt().json()`: quando esta opção é incluída, os registros são formatados em JSON. Para usar esta opção, é necessário incluir o recurso `json` na dependência `tracing-subscriber` (por exemplo, `tracing-subscriber = { version = "0.3.11", features = ["json"] }`).
- `#[tracing::instrument(skip(event), fields(req_id = %event.context.request_id))]`: esta anotação gera um intervalo toda vez que o manipulador é invocado. O intervalo adiciona o ID da solicitação a cada linha do log.
- `{ %first_name }`: esta estrutura adiciona o campo `first_name` à linha do log em que ele é usado. O valor desse campo corresponde à variável com o mesmo 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 essa função do Rust é invocada, ela imprime duas linhas de log semelhantes às seguintes:

```
{"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"}]}
```

Invocando o Lambda com eventos de outros serviços da AWS

Alguns serviços da AWS podem invocar diretamente as funções do Lambda usando acionadores. Esses serviços enviam eventos para o Lambda, e a função é invocada imediatamente quando o evento especificado ocorre. Os acionadores são adequados para eventos discretos e processamento em tempo real. Quando você cria um acionador usando o console do Lambda, o console interage com o serviço da AWS correspondente para configurar a notificação de eventos nesse serviço. Na verdade, o acionador é armazenado e gerenciado pelo serviço que gera os eventos, não pelo Lambda.

Os eventos são dados estruturados em formato JSON. A estrutura JSON varia dependendo do serviço que a gera e do tipo de evento, mas todos eles contêm os dados que a função precisa para processar o evento.

Uma função pode ter vários gatilhos. Cada acionador atua como um cliente ao invocar a função de forma independente, e cada evento que o Lambda transmite para a função tem dados de somente um acionador. O Lambda converte o documento do evento em um objeto e o transmite ao manipulador da função.

Dependendo do serviço, a invocação conduzida por eventos pode ser síncrona ou assíncrona.

- Para a invocação síncrona, o serviço que gera o evento aguarda a resposta da sua função. Esse serviço define os dados que a função precisa retornar na resposta. O serviço controla a estratégia de erro, como se deve tentar novamente em erros.
- Para a invocação assíncrona, o Lambda coloca o evento em filas antes de transmiti-lo para a função. Quando o Lambda coloca o evento em fila, ele envia imediatamente uma resposta bem-sucedida para o serviço que gerou o evento. Depois que a função processa o evento, o Lambda não retorna uma resposta ao serviço de geração de eventos.

Criar um acionador

A maneira mais fácil de criar um acionador é usar o console do Lambda. Quando você cria um acionador usando o console, o Lambda adiciona automaticamente as permissões necessárias à política baseada em recursos da função.

Para criar um acionador usando o console do Lambda

1. Abra a página Funções do console do Lambda.
2. Selecione a função para a qual deseja criar um acionador.
3. No painel Visão geral da função, escolha Adicionar gatilho.
4. Selecione o serviço da AWS no qual deseja invocar a função.
5. Preencha as opções no painel Configuração de acionador e escolha Adicionar. Dependendo do AWS service (Serviço da AWS) que você escolher para invocar a função, as opções de configuração do acionador serão diferentes.

Serviços que podem invocar funções do Lambda

A tabela a seguir lista os serviços que podem invocar funções do Lambda.

Serviço	Método de invocação
Amazon Alexa	Conduzido por eventos; invocação síncrona
Amazon Managed Streaming para Apache Kafka	Mapeamento de origem de evento
Apache Kafka autogerenciado	Mapeamento de origem de evento
Amazon API Gateway	Conduzido por eventos; invocação síncrona
AWS CloudFormation	Conduzido por eventos; invocação assíncrona
Amazon CloudFront (Lambda@Edge)	Conduzido por eventos; invocação síncrona
Amazon CloudWatch Logs	Conduzido por eventos; invocação assíncrona
AWS CodeCommit	Conduzido por eventos; invocação assíncrona
AWS CodePipeline	Conduzido por eventos; invocação assíncrona
Amazon Cognito	Conduzido por eventos; invocação síncrona
AWS Config	Conduzido por eventos; invocação assíncrona

Serviço	Método de invocação
Amazon Connect	Conduzido por eventos; invocação síncrona
Amazon DynamoDB	Mapeamento de origem de evento
Amazon Elastic File System	Integração especial
Elastic Load Balancing (Application Load Balancer)	Conduzido por eventos; invocação síncrona
AWS IoT	Conduzido por eventos; invocação assíncrona
Amazon Kinesis	Mapeamento de origem de evento
Amazon Data Firehose	Conduzido por eventos; invocação síncrona
Amazon Lex	Conduzido por eventos; invocação síncrona
Amazon MQ	Mapeamento de origem de evento
Amazon Simple Email Service	Conduzido por eventos; invocação assíncrona
Amazon Simple Notification Service	Conduzido por eventos; invocação assíncrona
Amazon Simple Queue Service	Mapeamento de origem de evento
Amazon Simple Storage Service (Amazon S3)	Conduzido por eventos; invocação assíncrona
Amazon Simple Storage Service Batch	Conduzido por eventos; invocação síncrona
Secrets Manager	Conduzido por eventos; invocação síncrona
Amazon VPC Lattice	Conduzido por eventos; invocação síncrona
AWS X-Ray	Integração especial

Tipos de aplicações e casos de uso comuns do Lambda

Acionadores e funções do Lambda são os componentes principais ao criar aplicações no AWS Lambda. Uma função do Lambda é o código e tempo de execução que processam eventos, e um acionador é o serviço ou aplicação da AWS que invoca a função. Para ilustrar, considere os seguintes cenários:

- Processamento de arquivos: suponha que você tenha um aplicação de compartilhamento de fotos. As pessoas usam o seu aplicação para fazer upload de fotos e o aplicação armazena essas fotos dos usuários no bucket do Amazon S3. Em seguida, o aplicativo cria uma versão em miniatura de cada foto e as exibe na página de perfil do usuário. Neste cenário, você pode optar por criar uma função do Lambda que cria uma miniatura automaticamente. O Amazon S3 é uma das fontes de eventos da AWS compatíveis que pode publicar eventos criados por objetos e invocar a função do Lambda. Seu código de função do Lambda pode ler o objeto da foto do bucket do S3, criar uma versão em miniatura e salvá-la em outro bucket do S3.
- Dados e análise: suponha que você está criando um aplicação de análise e armazenando dados brutos em uma tabela do DynamoDB. Quando você gravar, atualizar ou excluir itens em uma tabela, os DynamoDB Streams poderão publicar eventos de atualização do item para um fluxo associado à tabela. Neste caso, os dados de eventos fornecem a chave de item, o nome do evento (como, inserir, atualizar e excluir) e outros detalhes relevantes. Você pode escrever uma função do Lambda para gerar métricas personalizadas, agregando dados brutos.
- Sites: suponha que você esteja criando um site e deseja hospedar a lógica de backend no Lambda. Você pode invocar sua função do Lambda pelo HTTP usando o Amazon API Gateway como o endpoint HTTP. Agora, o cliente Web pode invocar a API e o API Gateway pode encaminhar a solicitação para o Lambda.
- Aplicações móveis: suponha que você tem uma aplicação móvel personalizada que produz eventos. Você pode criar uma função do Lambda para processar eventos publicados pela sua aplicação personalizada. Por exemplo, você pode configurar uma função do Lambda para processar os cliques na aplicação móvel personalizada.

O AWS Lambda oferece suporte a muitos produtos da AWS, como fontes de eventos. Para ter mais informações, consulte Invocando o Lambda com eventos de outros serviços da AWS. Quando você configura essas fontes de eventos para acionar uma função do Lambda, essa função do Lambda é invocada automaticamente quando ocorrem eventos. Você define o mapeamento de fontes de eventos, que é a maneira como você identifica quais eventos são rastreados e qual função do Lambda é invocada.

Veja a seguir exemplos introdutórios de fontes de eventos e de como a end-to-end experiência funciona.

Exemplo 1: o Amazon S3 envia eventos por push e invoca uma função do Lambda

O Amazon S3 pode publicar eventos de diferentes tipos, como eventos de objeto PUT, POST, COPY e DELETE, em um bucket. Usando o recurso de notificação do bucket, você pode configurar um mapeamento de fonte de evento que direciona o Amazon S3 para invocar uma função do Lambda quando um tipo específico de evento ocorre.

Veja a seguir uma sequência usual:

1. O usuário cria um objeto em um bucket.
2. O Amazon S3 detecta o evento criado por objeto.
3. O Amazon S3 invoca sua função do Lambda usando as permissões fornecidas pela função de execução.
4. O AWS Lambda executa a função do Lambda especificando o evento como um parâmetro.

Você configura o Amazon S3 para invocar sua função como uma ação de notificação do bucket. Para conceder ao Amazon S3 permissão para invocar a função, atualize a política baseada em recursos da função.

Exemplo 2: o AWS Lambda extrai eventos de uma transmissão do Kinesis e invoca uma função do Lambda

Para fontes de eventos baseadas em sondagem, o AWS Lambda sonda a fonte e, em seguida, invoca a função do Lambda quando são detectados registros naquela fonte.

- CreateEventSourceMapping
- UpdateEventSourceMapping

As etapas a seguir descrevem como um aplicação personalizada grava os registros em uma transmissão do Kinesis:

1. A aplicação personalizada grava os registros em uma transmissão do Kinesis.

2. O AWS Lambda sonda continuamente a transmissão e invoca a função do Lambda no momento em que o serviço detecta novos registros. O AWS Lambda sabe qual transmissão pesquisar e qual função do Lambda invocar com base no mapeamento de fontes de eventos criado no Lambda.
3. A função do Lambda é invocada com o evento de entrada.

Ao trabalhar com fontes de eventos com base em fluxo, você cria mapeamentos da fonte do evento noAWS Lambda. O Lambda lê itens da transmissão e invoca a função de forma síncrona. Você não precisa conceder ao Lambda permissão para invocar a função, mas ele precisa de permissão para ler a transmissão.

Usar o AWS Lambda com o Alexa

Você pode usar funções do Lambda para criar serviços que oferecem novas habilidades para a Alexa, o assistente de voz no Amazon Echo. O Alexa Skills Kit fornece as APIs, as ferramentas e a documentação para criar essas novas habilidades, com base em seus próprios serviços em execução como funções do Lambda. Os usuários do Amazon Echo podem acessar essas novas habilidades fazendo perguntas ao Alexa ou fazendo solicitações.

O Alexa Skills Kit está disponível em. GitHub

- SDK do Alexa Skills Kit para Java
- SDK do Alexa Skills Kit para Node.js
- SDK for Python do Alexa Skills Kit

Example Evento de casa inteligente da Alexa

```
{
  "header": {
    "payloadVersion": "1",
    "namespace": "Control",
    "name": "SwitchOnOffRequest"
  },
  "payload": {
    "switchControlAction": "TURN_ON",
    "appliance": {
      "additionalApplianceDetails": {
        "key2": "value2",
        "key1": "value1"
      },
      "applianceId": "sampleId"
    },
    "accessToken": "sampleAccessToken"
  }
}
```

Para obter mais informações, consulte Hospedar uma skill personalizada como função do AWS Lambda no guia Crie skills com a documentação do desenvolvedor do Alexa Skills Kit.

Invocar uma função do Lambda usando um endpoint do Amazon API Gateway

Você pode criar uma API da Web com um endpoint HTTP para a função do Lambda usando o Amazon API Gateway. O API Gateway fornece ferramentas para criar e documentar APIs da Web que direcionam as solicitações HTTP para as funções do Lambda. Você pode proteger o acesso à sua API com controles de autenticação e autorização. Suas APIs podem veicular o tráfego pela Internet ou podem ser acessadas somente dentro da VPC.

Os recursos em sua API definem um ou mais métodos, como GET ou POST. Os métodos têm uma integração que direciona solicitações para uma função do Lambda ou outro tipo de integração. Você pode definir cada recurso e método individualmente ou usar tipos especiais de recurso e método para corresponder todas as solicitações que se enquadram em um padrão. Um recurso proxy captura todos os caminhos abaixo de um recurso. O método `ANY` captura todos os métodos HTTP.

Seções

Escolher um tipo de API

O API Gateway é compatível com três tipos de API que invocam as funções do Lambda:

- API HTTP: uma API RESTful leve e de baixa latência.
- API REST: uma API RESTful personalizável e com muitos recursos.
- API de WebSocket: uma API da Web que mantém conexões persistentes com clientes para uma comunicação duplex completa.

As APIs HTTP e REST são ambas APIs RESTful que processam solicitações de HTTP e retornam respostas. As APIs HTTP são mais recentes e são criadas com a versão 2 da API do API Gateway. Os recursos abaixo são novos para APIs HTTP:

Recursos da API HTTP

- Implantações automáticas: quando você modifica rotas ou integrações, as alterações são implantadas automaticamente em estágios com implantação automática habilitada.
- Estágio padrão: você pode criar um estágio padrão (`$default`) para veicular solicitações no caminho raiz do URL da API. Para estágios nomeados, você deve incluir o nome do estágio no início do caminho.
- Configuração de CORS: você pode configurar sua API para adicionar cabeçalhos CORS às respostas de saída em vez de adicioná-las manualmente no código da função.

As APIs REST são APIs RESTful clássicas às quais o API Gateway oferece suporte desde o lançamento. As APIs REST têm atualmente mais recursos de personalização, integração e gerenciamento.

Recursos da API REST

- Tipos de integração: APIs REST são compatíveis com integrações personalizadas do Lambda. Com uma integração personalizada, você pode enviar apenas o corpo da solicitação para a função ou aplicar um modelo de transformação ao corpo da solicitação antes de enviá-la para a função.
- Controle de acesso: as APIs REST oferecem suporte a mais opções de autenticação e autorização.
- Monitoramento e rastreamento: as APIs REST oferecem suporte ao monitoramento do AWS X-Ray e a outras opções de log.

Para obter uma comparação detalhada, consulte Escolher entre APIs HTTP e REST no Guia do desenvolvedor do API Gateway.

As APIs WebSocket também usam a API do API Gateway versão 2 e oferecem suporte a um conjunto de recursos semelhante. Use uma API WebSocket para aplicativos que se beneficiam de uma conexão persistente entre o cliente e a API. As APIs WebSocket oferecem uma comunicação duplex completa, o que significa que tanto o cliente quanto a API podem enviar mensagens continuamente sem esperar por uma resposta.

As APIs HTTP oferecem suporte a um formato de evento simplificado (versão 2.0). O exemplo abaixo mostra um evento de uma API HTTP.

Example event-v2.json: evento de proxy do API Gateway (API HTTP)

```
{
    "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
}
```

Para obter mais informações, consulte Integrações do AWS Lambda no Guia do desenvolvedor do API Gateway.

Adicionar um endpoint público à sua função do Lambda

Como adicionar um endpoint público à sua função do Lambda

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Em Visão geral da função, escolha Adicionar gatilho.
4. Selecione API Gateway.
5. Escolher Create an API (Criar uma API) ou Use an existing API (Usar uma API existente).
    a. New API (Nova API): para API type (Tipo de API), escolha HTTP API. Para saber mais, consulte Tipos de APIs.
    b. Existing API (API existente): selecione a API no menu suspenso ou insira seu ID (como r3pmxmplak).
6. Em Security (Segurança), escolha Open (Abrir).
7. Escolha Adicionar.

Integração de proxy

As APIs do API Gateway são compostas por estágios, recursos, métodos e integrações. O estágio e o recurso determinam o caminho do endpoint:

Formato do caminho da API

- `/prod/`: o estágio e o recurso raiz de `prod`.
- `/prod/user`: o estágio de `prod` e o recurso de `user`.
- `/dev/{proxy+}`: qualquer rota no estágio de `dev`.
- `/`: (APIs HTTP) o estágio padrão e o recurso raiz.

Uma integração do Lambda mapeia uma combinação de caminho e método de HTTP para uma função do Lambda. Você pode configurar o API Gateway para transmitir o corpo da solicitação de HTTP no estado em que se encontra (integração personalizada) ou para encapsular o corpo da

solicitação em um documento que inclui todas as informações de solicitação, inclusive cabeçalhos, recursos, caminho e método.

Para obter mais informações, consulte Configurar integrações de proxy do Lambda no API Gateway.

Formato de eventos

O Amazon API Gateway invoca sua função de modo síncrono com um evento que contém uma representação JSON da solicitação de HTTP. Para uma integração personalizada, o evento é o corpo da solicitação. Para uma integração de proxy, o evento tem uma estrutura definida. O exemplo abaixo mostra um evento de proxy de uma API REST do API Gateway.

Example Evento de proxy event.json do API Gateway (API REST)

```
{
      "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 de resposta

O API Gateway aguarda uma resposta de sua função e retransmite o resultado para o chamador. Para uma integração personalizada, defina uma resposta de integração e uma resposta de método para converter a saída da função em uma resposta de HTTP. Para uma integração de proxy, a função deve responder com uma representação da resposta em um formato específico.

O exemplo abaixo mostra um objeto de resposta de uma função Node.js. O objeto de resposta representa uma resposta de HTTP bem-sucedida que contém um documento JSON.

Example index.mjs: objeto de resposta de integração de 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}"
    }
```

O runtime do Lambda serializa o objeto de resposta em JSON e o envia para a API. A API analisa a resposta e a utiliza para criar uma resposta de HTTP que, então, é enviada para o cliente que fez a solicitação original.

Example Resposta 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
}
```

Permissões

O Amazon API Gateway recebe permissão para invocar sua função por meio da política baseada em recursos da função. Você pode conceder permissão de invocar a uma API inteira ou conceder acesso limitado a um estágio, recurso ou método.

Ao adicionar uma API à sua função usando o console do Lambda, o console do API Gateway ou em um modelo de AWS SAM, a política baseada em recursos da função é atualizada automaticamente. Veja abaixo um exemplo de política de função.

Example política de função

```
{
  "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/"
        }
      }
    }
  ]
}
```

Você pode gerenciar manualmente as permissões da política de função com as seguintes operações da API:

- AddPermission
- RemovePermission
- GetPolicy

Para conceder permissão de invocação a uma API existente, use o 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/"
```

A seguinte saída deverá ser mostrada:

```
{
    "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 a sua função e a API estiverem em Regiões da AWS diferentes, o identificador de região no ARN de origem deverá corresponder à região da função, e não à região da API. Quando o API Gateway invoca uma função, ele usa um ARN de recurso baseado no ARN da API, mas modificado para corresponder à região da função.

O ARN de origem no exemplo concede permissão a uma integração no método GET do recurso raiz no estágio padrão de uma API com o ID `mnh1xmpli7`. Você pode usar um asterisco no ARN de origem para conceder permissões a vários estágios, métodos ou recursos.

Padrões de recursos

- `mnh1xmpli7/*/GET/*`: método GET em todos os recursos, em todos os estágios.
- `mnh1xmpli7/prod/ANY/user`: método ANY no recurso `user` no estágio `prod`.
- `mnh1xmpli7/*/*/*`: qualquer método em todos os recursos, em todos os estágios.

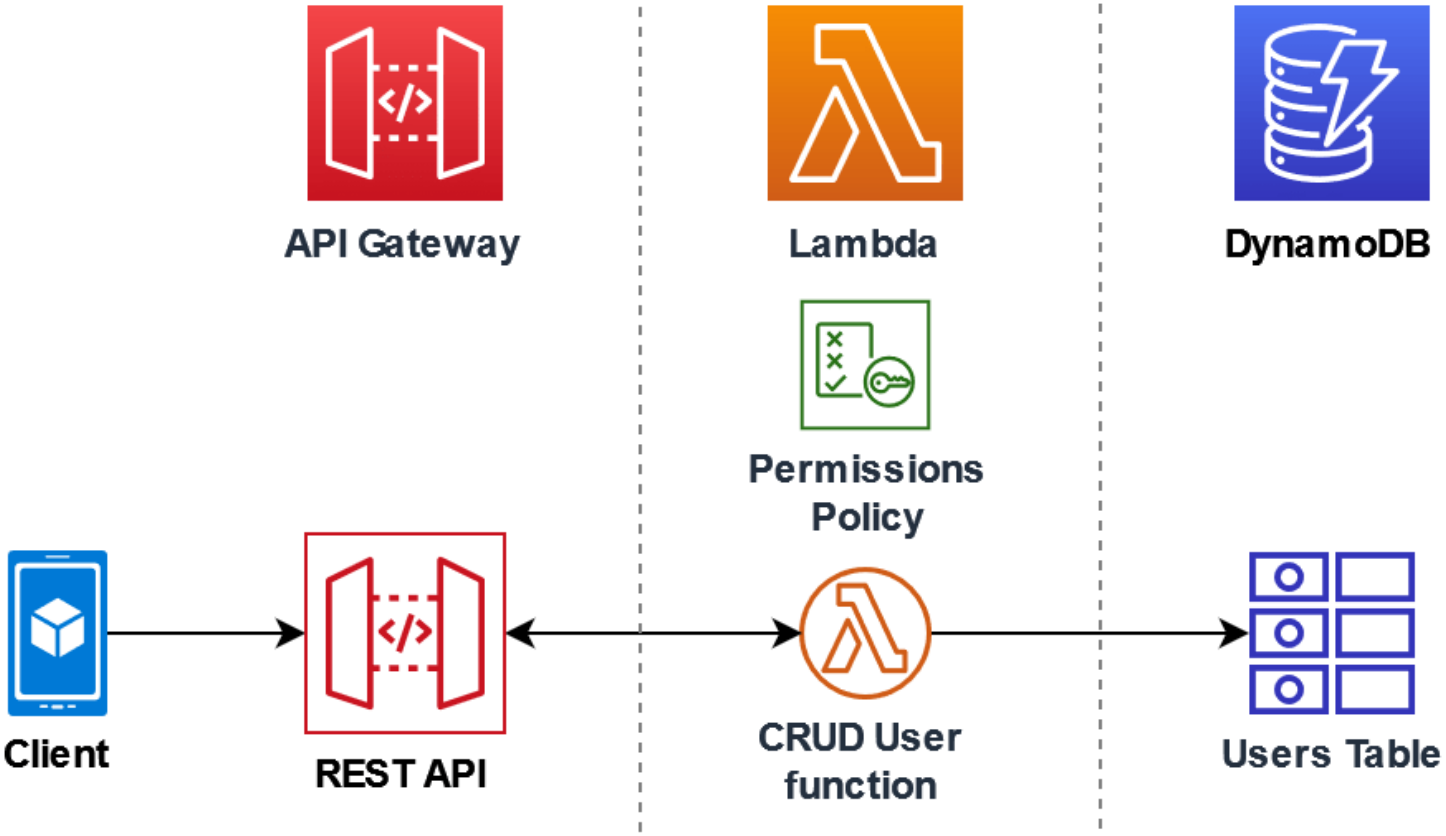
Para obter detalhes sobre como exibir a política e remover instruções, consulte Limpar políticas baseadas em recursos.

Aplicação de exemplo

O aplicativo de exemplo de API Gateway com Node.js inclui uma função com um modelo do AWS SAM que cria uma API REST com o rastreamento do AWS X-Ray habilitado. Ele também inclui scripts para implantar, invocar a função, testar a API e fazer a limpeza.

Tutorial: Uso do Lambda com API Gateway

Neste tutorial, você criará uma API REST que será usada para invocar uma função do Lambda usando uma solicitação HTTP. A função do Lambda executará operações de criação, leitura, atualização e exclusão (CRUD) em uma tabela do DynamoDB. Essa função será fornecida aqui para fins de demonstração, mas você aprenderá a configurar uma API REST do API Gateway que pode invocar qualquer função do Lambda.



O uso do API Gateway fornece aos usuários um endpoint HTTP seguro para invocar a função do Lambda e pode ajudar a gerenciar grandes volumes de chamadas para a função, ao realizar o controle de utilização do tráfego e validar e autorizar as chamadas de API automaticamente. O API Gateway também oferece controles de segurança flexíveis usando o AWS Identity and Access Management (IAM) e o Amazon Cognito. Isso é útil para casos de uso em que é necessária uma autorização prévia para as chamadas em sua aplicação.

Para concluir este tutorial, você passará pelos estágios a seguir:

1. Criação e configuração de uma função do Lambda em Python ou em Node.js para a execução de operações em uma tabela do DynamoDB.
2. Criação de uma API REST no API Gateway para conexão com a função do Lambda.
3. Criação de uma tabela do DynamoDB e realização de teste com a função do Lambda no console.
4. Implantação da API e realização de teste da configuração completa usando curl em um terminal.

Ao concluir esses estágios, você aprenderá a usar o API Gateway para criar um endpoint HTTP que pode invocar uma função do Lambda com segurança em qualquer escala. Você também aprenderá

como implantar a API e testá-la no console, além de como enviar uma solicitação HTTP usando um terminal.

Seções

Pré-requisitos

Cadastre-se em uma Conta da AWS

Se você ainda não tem Conta da AWS, siga as etapas a seguir para criar uma.

Para se cadastrar em uma Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

   Quando você se cadastra em uma Conta da AWS, um Usuário raiz da conta da AWS é criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como uma prática recomendada de segurança, atribua o acesso administrativo para um usuário e use somente o usuário-raiz para executar tarefas que requerem o acesso de usuário-raiz.

A AWS envia um e-mail de confirmação depois que o processo de cadastramento é concluído. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se cadastrar em uma Conta da AWS, proteja seu Usuário raiz da conta da AWS, habilite o AWS IAM Identity Center e crie um usuário administrativo para não usar o usuário raiz em tarefas cotidianas.

Proteger seu Usuário raiz da conta da AWS

1. Faça login no AWS Management Console como o proprietário da conta ao escolher a opção Usuário raiz e inserir o endereço de e-mail da Conta da AWS. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Signing in as the root user (Fazer login como usuário-raiz) no Guia do usuário do Início de Sessão da AWS.
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo MFA virtual para o usuário-raiz de sua conta da Conta da AWS (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center.
2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para obter um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso dos usuários com o Diretório do Centro de Identidade do IAM padrão no Guia do usuário do AWS IAM Identity Center.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use a URL de login que foi enviada ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

Para obter ajuda com o login utilizando um usuário do Centro de Identidade do IAM, consulte Fazer login no portal de acesso da AWS, no Guia do usuário do Início de Sessão da AWS.

Atribuir acesso para usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center.

2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center.

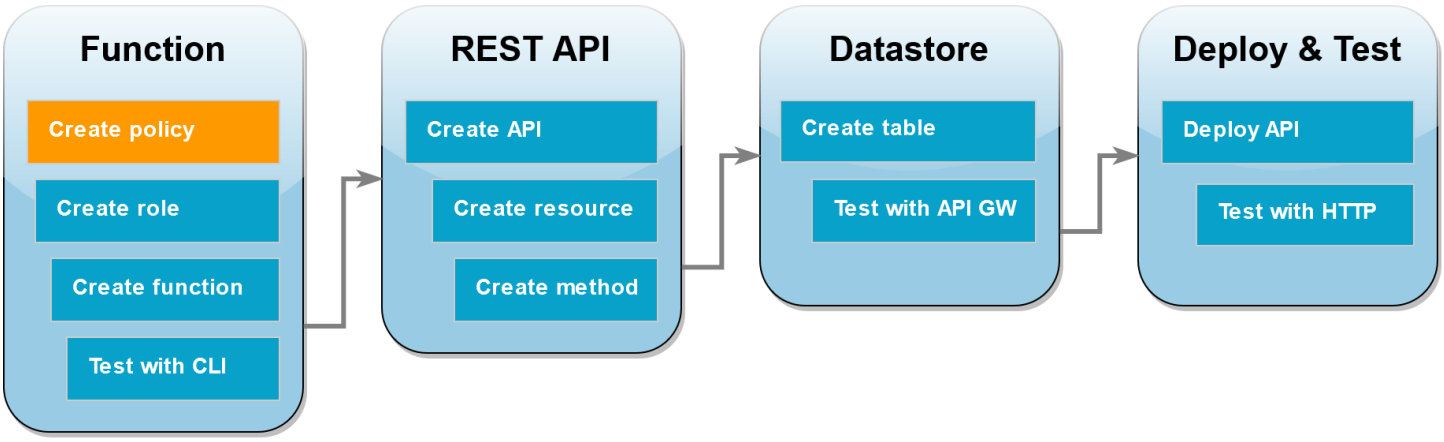
Instalar a AWS Command Line Interface

Se você ainda não instalou a AWS Command Line Interface, siga as etapas em Instalar ou atualizar a versão mais recente da AWS CLI para instalá-la.

O tutorial requer um terminal de linha de comando ou um shell para executar os comandos. No Linux e no macOS, use o gerenciador de pacotes e de shell de sua preferência.

> **Note**
>
> No Windows, alguns comandos da CLI do Bash que você costuma usar com o Lambda (como `zip`) não são compatíveis com os terminais integrados do sistema operacional. Para obter uma versão do Ubuntu com o Bash integrada no Windows, instale o Subsistema do Windows para Linux.

Criação de uma política de permissões



Antes de ser possível criar uma função de execução para a função do Lambda, primeiro é necessário criar uma política de permissões para fornecer à sua função permissão para acessar os recursos da AWS necessários. Para este tutorial, a política permitirá que o Lambda execute operações de CRUD em uma tabela do DynamoDB e grave no Amazon CloudWatch Logs.

Para criar a política

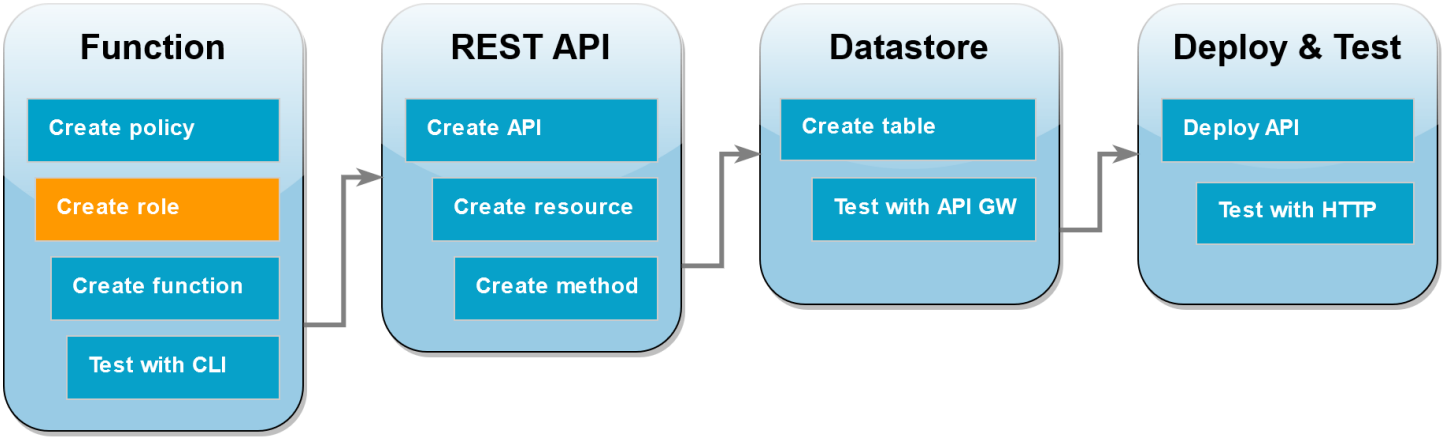
1. Abra a página Policies (Políticas) do console do IAM.
2. Escolha Criar política.
3. Escolha a guia JSON e cole a política personalizada a seguir no editor JSON.

```
{
  "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. Escolha Próximo: etiquetas.
5. Selecione Next: Review (Próximo: revisar).
6. No campo Política de revisão, em Nome da política, insira **`lambda-apigateway-policy`**.
7. Escolha Criar política.

Criar uma função de execução



Um perfil de execução é um perfil do AWS Identity and Access Management (IAM) que concede a uma função do Lambda permissão para acessar serviços e recursos da AWS. Para permitir que a função execute operações em uma tabela do DynamoDB, vincule a política de permissões criada na etapa anterior.

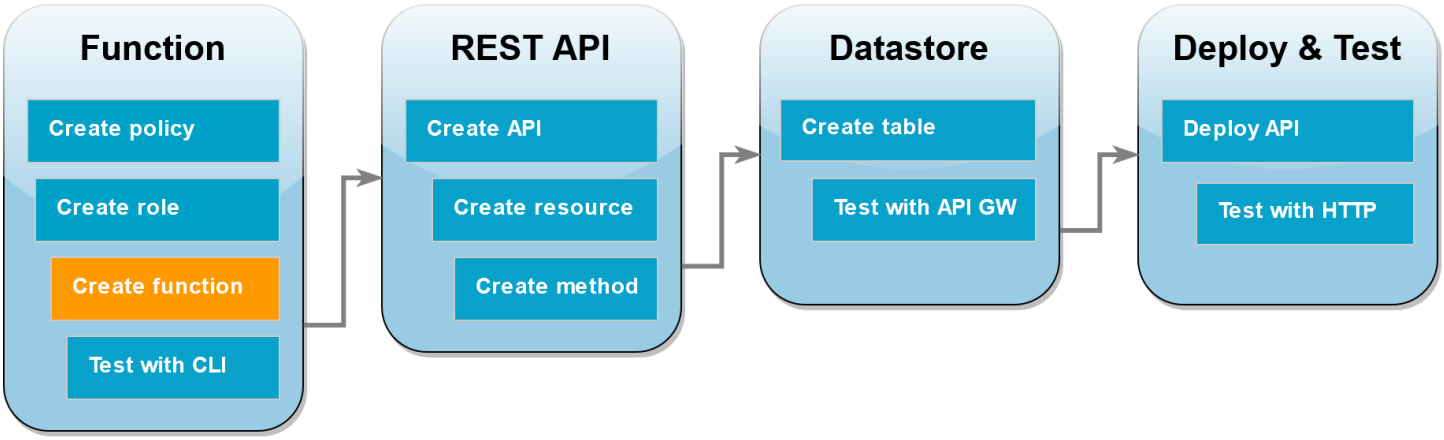
Para criar um perfil de execução e vincular a política de permissões personalizada

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione Criar função.

3. Para o tipo de entidade confiável, escolha Serviço da AWS e, em seguida, para o caso de uso, selecione Lambda.
4. Escolha Próximo.
5. Na caixa de pesquisa de política, insira **`lambda-apigateway-policy`**.
6. Nos resultados da pesquisa, selecione a política que você criou (`lambda-apigateway-policy`) e, depois, escolha Next (Avançar).
7. Em Role details (Detalhes do perfil), para Role name (Nome do perfil), insira **`lambda-apigateway-role`** e, em seguida, escolha Create role (Criar perfil).

Posteriormente no tutorial, você precisará do nome do recurso da Amazon (ARN) do perfil que acabou de criar. Na página Roles (Perfis) do console do IAM, escolha o nome do seu perfil (`lambda-apigateway-role`) e copie o Role ARN (ARN do perfil) exibido na página Summary (Resumo).

Criar a função



O exemplo de código a seguir recebe uma entrada de evento do API Gateway especificando uma operação a ser executada na tabela do DynamoDB que você criará e alguns dados de carga útil. Se os parâmetros recebidos pela função forem válidos, ela executará a operação solicitada na tabela.

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**
>
> Neste exemplo, o nome da tabela do DynamoDB está definido como uma variável em seu código da função. Em uma aplicação real, a prática recomendada é transferir esse parâmetro como uma variável de ambiente e evitar codificar o nome da tabela. Para obter mais informações, consulte Usar variáveis de ambiente do AWS Lambda.

Para criar a função

1. Salve o exemplo de código como um arquivo denominado `index.mjs` e, se necessário, edite a região da AWS especificada no código. A região especificada no código deve ser semelhante à região que você criará a tabela do DynamoDB, posteriormente, no tutorial.
2. Crie um pacote de implantação usando o comando `zip` a seguir.

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

3. Crie uma função do Lambda com o comando `create-function` da AWS CLI. Para o parâmetro `role`, insira o nome do recurso da Amazon (ARN) do perfil de execução que você copiou anteriormente.

```
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**
>
> Neste exemplo, o nome da tabela do DynamoDB está definido como uma variável em seu código da função. Em uma aplicação real, a prática recomendada é transferir esse parâmetro como uma variável de ambiente e evitar codificar o nome da tabela. Para obter mais informações, consulte Usar variáveis de ambiente do AWS Lambda.

Para criar a função

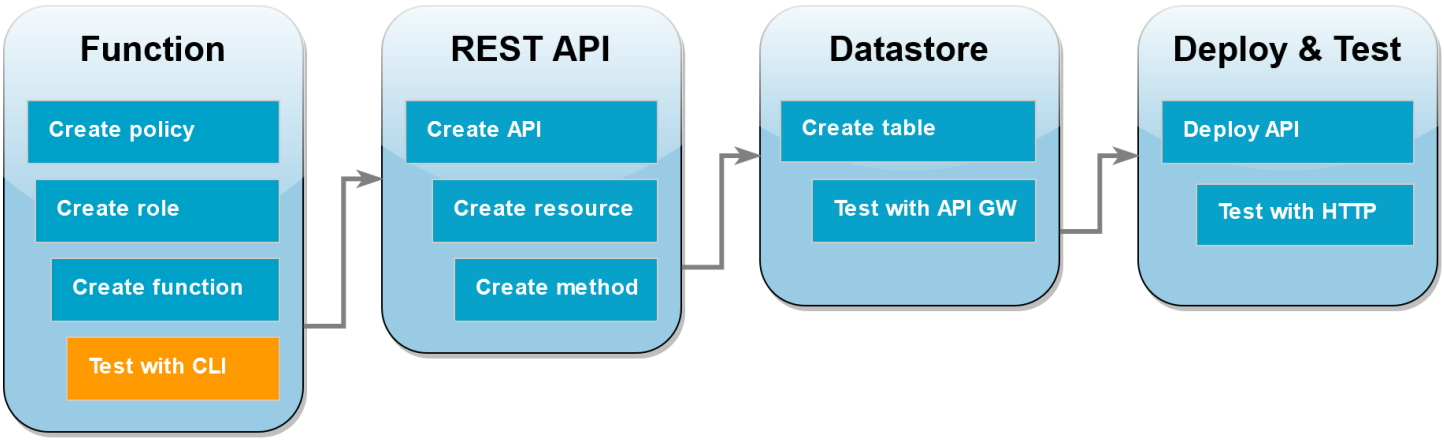
1. Salve o exemplo de código como um arquivo denominado `LambdaFunctionOverHttps.py`.
2. Crie um pacote de implantação usando o comando `zip` a seguir.

```
zip function.zip LambdaFunctionOverHttps.py
```

3. Crie uma função do Lambda com o comando `create-function` da AWS CLI. Para o parâmetro `role`, insira o nome do recurso da Amazon (ARN) da função de execução que você copiou anteriormente.

```
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
```

Invocação da função usando a AWS CLI



Antes de integrar a função com o API Gateway, confirme se a implantação da função ocorreu com êxito. Crie um evento de teste contendo os parâmetros que a API do API Gateway enviará para o Lambda e use o comando `invoke` da AWS CLI para executar a função.

Para invocar a função do Lambda com a AWS CLI

1. Salve o JSON a seguir como um arquivo denominado `input.txt`.

```
{
    "operation": "echo",
    "payload": {
        "somekey1": "somevalue1",
        "somekey2": "somevalue2"
    }
}
```

2. Execute o comando `invoke` a seguir da AWS CLI.

```
aws lambda invoke \
--function-name LambdaFunctionOverHttps \
--payload file://input.txt outputfile.txt \
--cli-binary-format raw-in-base64-out
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

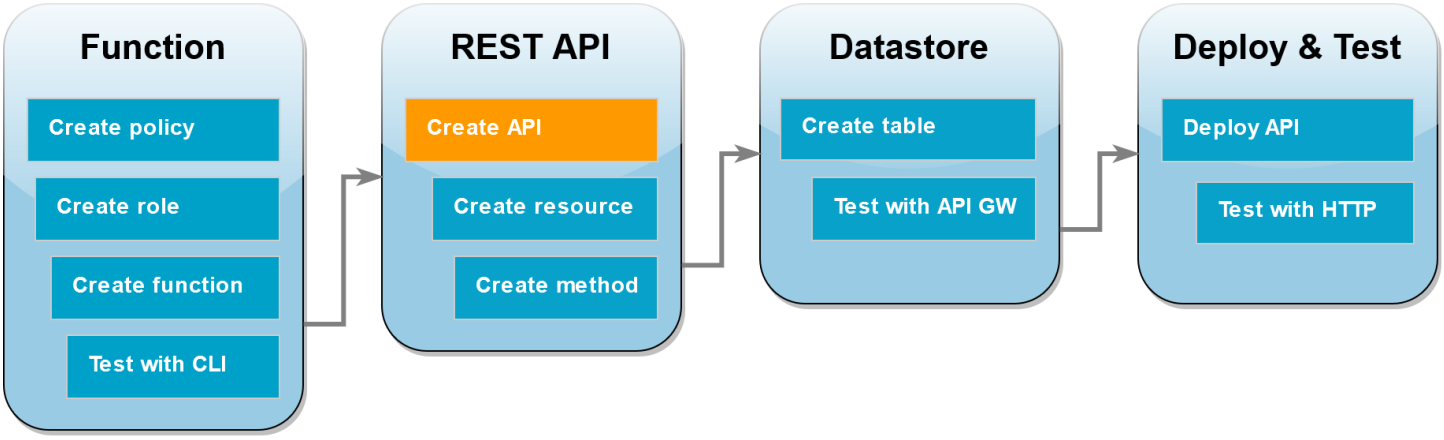
Você deverá ver a seguinte resposta:

```
{
"StatusCode": 200,
"ExecutedVersion": "LATEST"
}
```

3. Confirme se a função executou a operação `echo` especificada no evento de teste JSON. Inspecione o arquivo `outputfile.txt` e verifique se ele contém o seguinte:

```
{"somekey1": "somevalue1", "somekey2": "somevalue2"}
```

Criar uma API REST usando o API Gateway

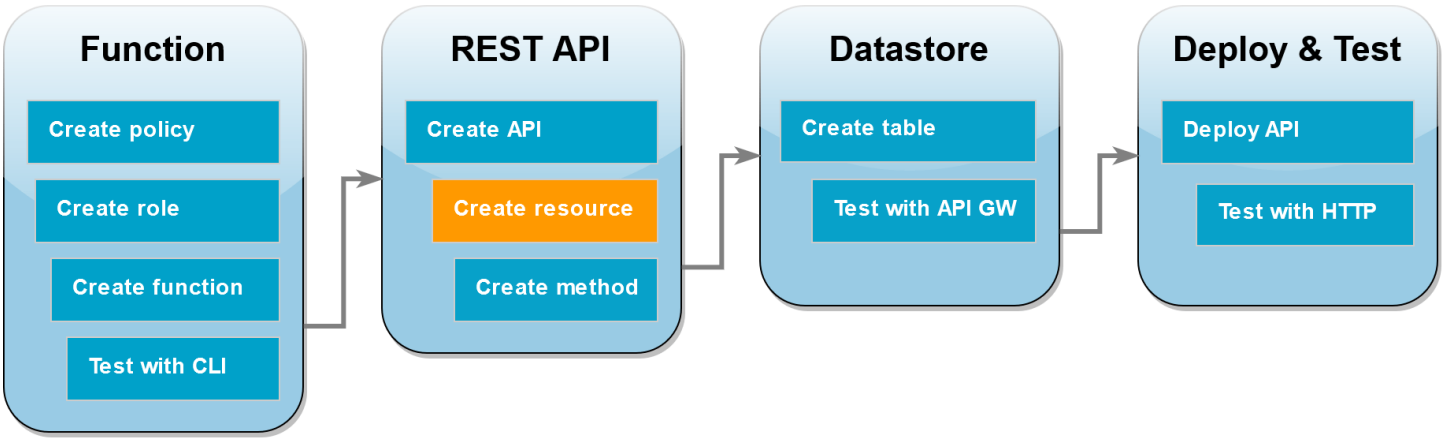


Nesta etapa, você cria a API REST do API Gateway que usará para invocar a função do Lambda.

Para criar a API

1. Abra o console do API Gateway.
2. Selecione Create API (Criar API).
3. Na caixa do API REST, escolha Construir.
4. Em Detalhes da API, deixe a opção Nova API selecionada e, em Nome da API, insira**`DynamoDBOperations`**.
5. Selecione Criar API.

Criação de um recurso na API REST

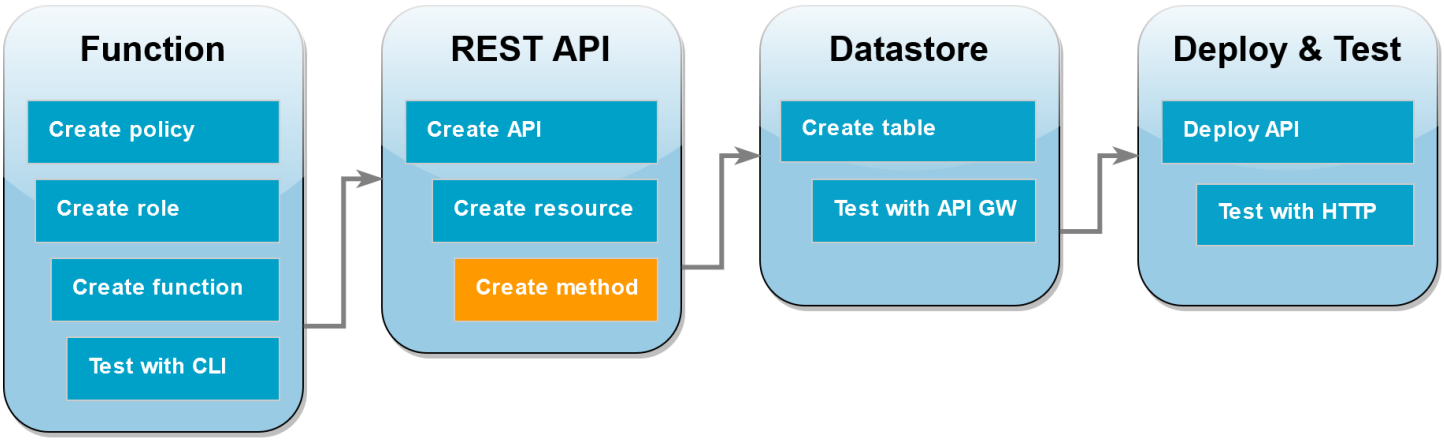


Para adicionar um método HTTP à API, primeiro é necessário criar um recurso no qual esse método possa operar. Aqui você cria o recurso para gerenciar a tabela do DynamoDB.

Para criar o recurso

1. No Console do API Gateway, na página Recursos da sua API, escolha Criar recurso.
2. Em Detalhes do recurso, em Nome do recurso, insira **`DynamoDBManager`**.
3. Escolha Create Resource (Criar recurso).

Criação de um método HTTP POST



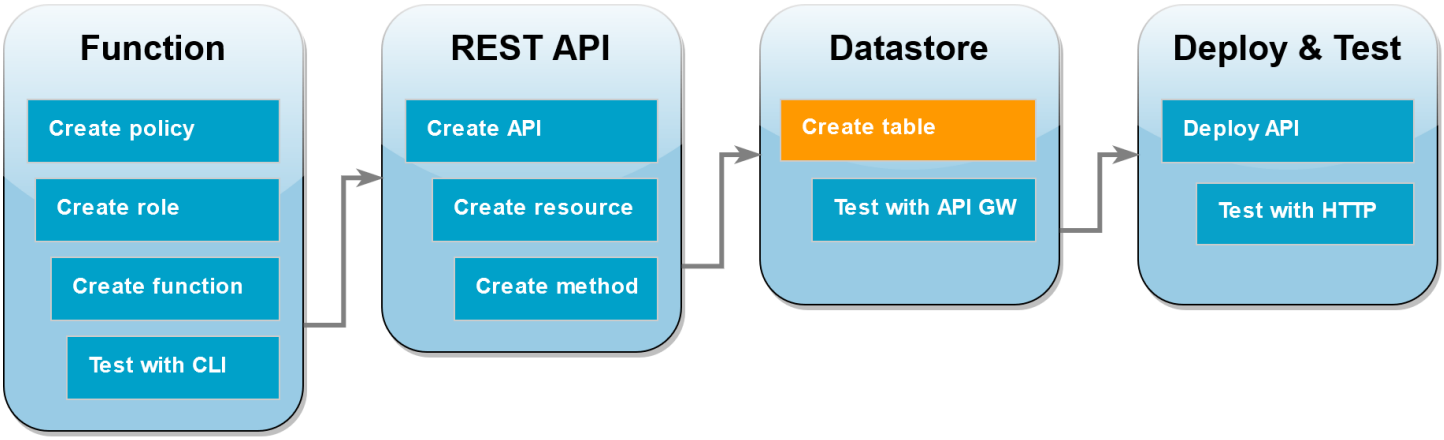
Nesta etapa, você cria um método (`POST`) para o recurso `DynamoDBManager`. Você vincula esse método `POST` à função do Lambda para que, quando o método receber uma solicitação HTTP, o API Gateway invoque sua função do Lambda.

> **Note**
>
> Para a finalidade deste tutorial, um método HTTP (`POST`) será usado para invocar uma única função do Lambda que executa todas as operações em sua tabela do DynamoDB. Em uma aplicação real, a prática recomendada é usar diferentes funções do Lambda e métodos HTTP para cada operação. Para obter mais informações, consulte The Lambda monolith no Serverless Land.

Para criar o método POST

1. Na página Recursos da API, certifique-se de que o recurso `/DynamoDBManager` esteja realçado. Em seguida, no painel Métodos, escolha Criar método.
2. Em Tipo de método, escolha POST.
3. Em Tipo de integração, mantenha a opção Função do Lambda selecionada.
4. Em Função do Lambda, escolha o nome do recurso da Amazon (ARN) para sua função (`LambdaFunctionOverHttps`).
5. Escolha Criar método.

Criar uma tabela do DynamoDB



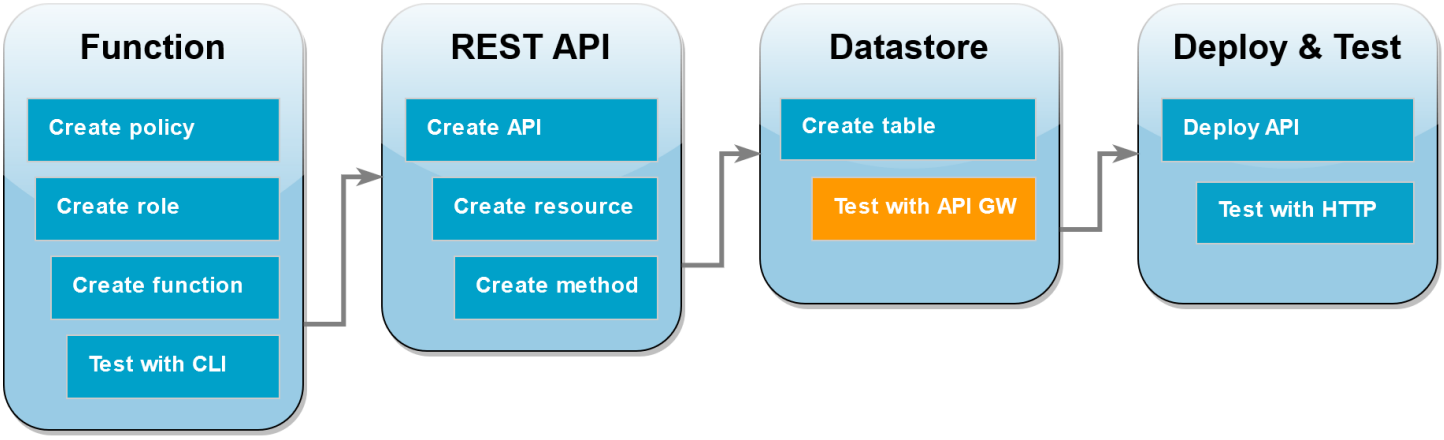
Crie uma tabela do DynamoDB em branco na qual a função do Lambda executará as operações de CRUD.

Para criar uma tabela do DynamoDB

1. Abra a página Tables (Tabelas) no console do DynamoDB.

2. Escolha Create table.
3. Em Detalhes da tabela, faça o seguinte:
    1. Em Table name (Nome da tabela), insira **`lambda-apigateway`**.
    2. Para a Chave de partição, insira **`id`** e mantenha o tipo de dados definido como String.
4. Em Table settings (Configurações da tabela), mantenha Default settings (Configurações padrões).
5. Escolha Create table.

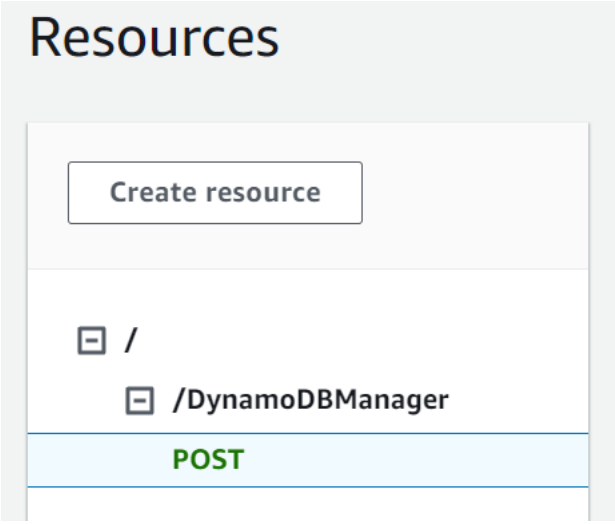
Teste da integração do API Gateway, do Lambda e do DynamoDB



Agora está tudo pronto para que ocorra o teste da integração do método de API do API Gateway com a função do Lambda e a tabela do DynamoDB. Ao usar o console do API Gateway, você envia solicitações diretamente para o método POST usando a função de teste do console. Nesta etapa, primeiro você usa uma operação `create` para adicionar um novo item à tabela do DynamoDB e, em seguida, usa uma operação `update` para modificar o item.

Teste 1: para criar um novo item na tabela do DynamoDB

1. No API Gateway console (Console do API Gateway), escolha a sua API (`DynamoDBOperations`).
2. No recurso `DynamoDBManager`, escolha o método POST.



3. Selecione a guia Testar. Talvez seja necessário selecionar o botão de seta para a direita para mostrar a guia.
4. Em Método de teste, mantenha as opções Strings de consulta e Cabeçalhos vazias. Em Corpo da solicitação, cole o seguinte JSON:

```
{
  "operation": "create",
  "payload": {
    "Item": {
      "id": "1234ABCD",
      "number": 5
    }
  }
}
```

5. Escolha Testar.

   Os resultados que são exibidos quando o teste é concluído devem mostrar status `200`. Esse código de status indica que a operação `create` ocorreu com êxito.

   Para confirmar, verifique se a tabela do DynamoDB passou a conter o novo item.
6. Abra a página Tables (Tabelas) do console do DynamoDB e escolha a tabela `lambda-apigateway`.
7. Escolha Explore table items (Explorar itens da tabela). No painel Itens retornados, você verá um item com o id `1234ABCD` e o número `5`.

Teste 2: para atualizar o item na tabela do DynamoDB

1. No Console do API Gateway, retorne para a guia Teste do seu método POST.
2. Em Método de teste, mantenha as opções Strings de consulta e Cabeçalhos vazias. Em Corpo da solicitação, cole o seguinte JSON:

```
{
    "operation": "update",
    "payload": {
        "Key": {
            "id": "1234ABCD"
        },
        "AttributeUpdates": {
            "number": {
                "Value": 10
            }
        }
    }
}
```

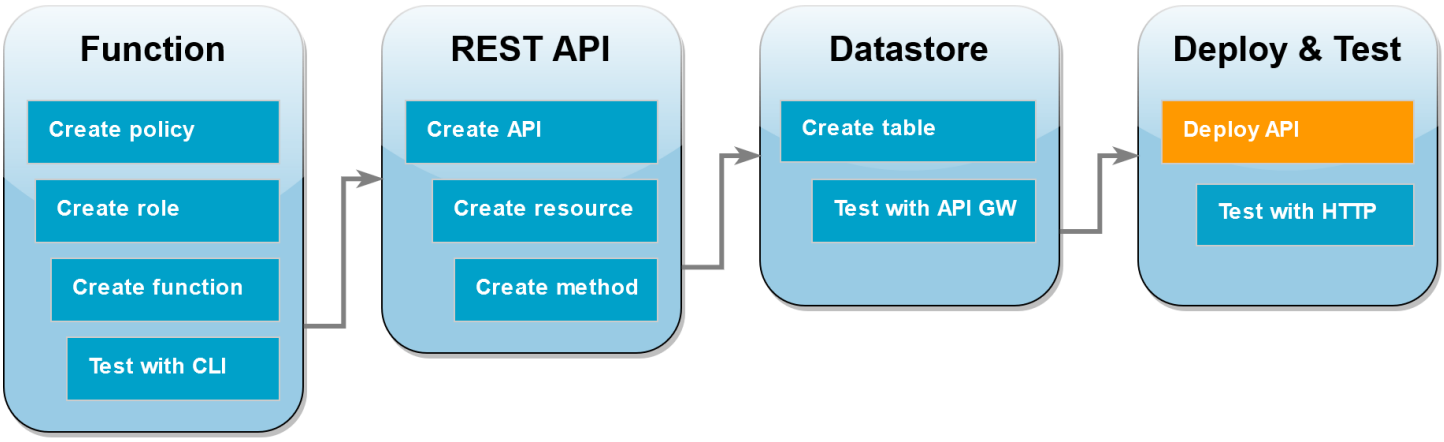
3. Escolha Testar.

   Os resultados que são exibidos quando o teste é concluído devem mostrar status `200`. Esse código de status indica que a operação `update` ocorreu com êxito.

   Para confirmar, verifique se o item na tabela do DynamoDB foi modificado.

4. Abra a página Tables (Tabelas) do console do DynamoDB e escolha a tabela `lambda-apigateway`.
5. Escolha Explore table items (Explorar itens da tabela). No painel Itens retornados, você verá um item com o id `1234ABCD` e o número `10`.

Implantar a API

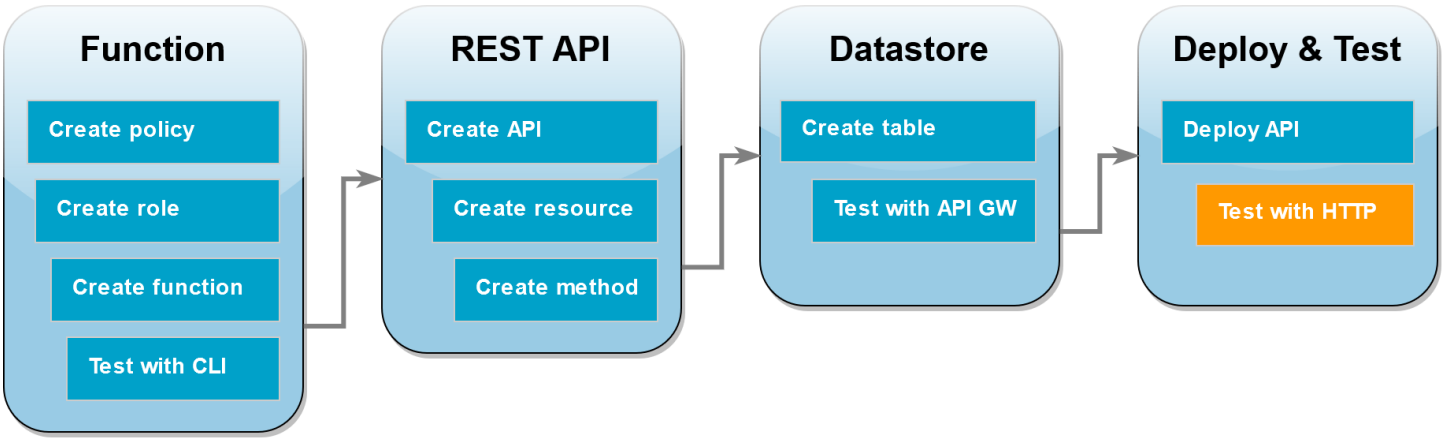


Para que um cliente chame a API, você deve criar uma implantação e um estágio associado. Um estágio representa um snapshot da API, incluindo seus métodos e integrações.

Para implantar a API

1. Abra a página APIs do API Gateway console (Console do API Gateway) e escolha a API `DynamoDBOperations`.
2. Na página Recursos da sua API, escolha Implantar API.
3. Em Estágio, escolha *Novo estágio* e, em seguida, em Nome do estágio, insira **`test`**.
4. Escolha Implantar.
5. No painel Detalhes do estágio copie o URL de invocação. Você usará isso na próxima etapa para invocar a função usando uma solicitação HTTP.

Uso de curl para invocar a função usando solicitações HTTP



Agora é possível invocar a função do Lambda ao emitir uma solicitação HTTP para a API. Nesta etapa, você criará um novo item na tabela do DynamoDB e o excluirá.

Para invocar a função do Lambda usando curl

1. Execute o comando `curl` usando o URL de invocação que você copiou na etapa anterior. Quando você usa curl com a opção `-d` (dados), o método HTTP POST é usado automaticamente.

   ```
   curl https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/DynamoDBManager \
   -d '{"operation": "create", "payload": {"Item": {"id": "5678EFGH", "number": 15}}}'
   ```

2. Para verificar se a operação de criação ocorreu com êxito, faça o seguinte:

   1. Abra a página Tables (Tabelas) do console do DynamoDB e escolha a tabela `lambda-apigateway`.
   2. Escolha Explore table items (Explorar itens da tabela). No painel Itens retornados, você verá um item com o id `5678EFGH` e o número `15`.

3. Execute o comando `curl` a seguir para excluir o item que você acabou de criar. Use seu próprio URL de invocação.

   ```
   curl https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/DynamoDBManager \
   -d '{"operation": "delete", "payload": {"Key": {"id": "5678EFGH"}}}'
   ```

4. Confirme se a operação de exclusão ocorreu com êxito. No painel Items returned (Itens retornados) da página Explore items (Explorar itens) do console do DynamoDB, verifique se o item com o id `5678EFGH` não está mais na tabela.

Limpeza dos recursos (opcional)

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.

4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Para excluir a função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Escolha Excluir.
4. Insira o nome do perfil no campo de entrada de texto e escolha Delete (Excluir).

Para excluir a API

1. Abra a página APIs do console do API Gateway.
2. Selecione a API que você criou.
3. Escolha Ações, Excluir.
4. Escolha Excluir.

Para excluir uma tabela do DynamoDB

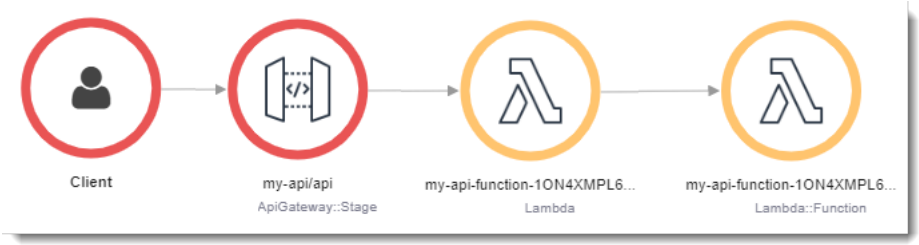
1. Abra a página Tables (Tabelas) no console do DynamoDB.
2. Selecione a tabela que você criou.
3. Escolha Excluir.
4. Digite **`delete`** na caixa de texto.
5. Selecione Delete table (Excluir tabela).

Tratamento de erros do Lambda com uma API do API Gateway

O API Gateway trata todos os erros de invocação e função como erros internos. Se a API do Lambda rejeitar a solicitação de invocação, o API Gateway retornará um código de erro 500. Se a função é executada, mas retorna um erro ou retorna uma resposta no formato errado, o API Gateway retorna um código de erro 502. Em ambos os casos, o corpo da resposta do API Gateway é `{"message": "Internal server error"}`.

> **Note**
>
> O API Gateway não tenta novamente nenhuma invocação do Lambda. Se o Lambda retornar um erro, o API Gateway retornará uma resposta de erro ao cliente.

O exemplo a seguir mostra um mapa de rastreamento do X-Ray para uma solicitação que resultou em um erro de função e um erro 502 do API Gateway. O cliente recebe a mensagem de erro genérica.

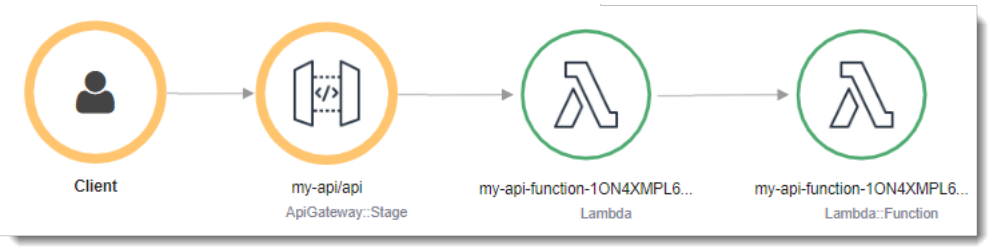


Para personalizar a resposta de erro, é necessário detectar erros no código e formatar uma resposta no formato necessário.

Example index.mjs: erro de formatação

```
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
}
```

O API Gateway converte essa resposta em um erro HTTP com um código de status e um corpo personalizado. No mapa de rastreamento, o nó da função é verde porque ele tratou o erro.



Usar o AWS Lambda com o AWS Application Composer

AWS Application Composer é uma ferramenta de criação visual para projetar aplicações modernas na AWS. Para projetar a arquitetura das aplicações, basta arrastar, agrupar e conectar Serviços da AWS em uma tela visual. O Application Composer cria modelos de infraestrutura como código (IaC) com base no seu design que podem ser implantados usando o AWS SAM ou o AWS CloudFormation.

Exportar a função do lambda para o Application Composer

Você pode começar a usar o Application Composer criando um projeto com base na configuração de uma função do Lambda existente usando o console do Lambda. Para exportar a configuração e o código da função para o Application Composer a fim de criar um projeto, faça o seguinte:

1. Abra a página Funções do console do Lambda.
2. Selecione a função que você deseja usar como base para o projeto no Application Composer.
3. No painel Visão geral da função, escolha Exportar para o Application Composer.

   Para exportar a configuração e o código da função para o Application Composer, o Lambda cria um bucket do Amazon S3 na sua conta para armazenar esses dados temporariamente.
4. Na caixa de diálogo, escolha Confirmar e criar projeto para aceitar o nome padrão desse bucket e exportar a configuração e o código da função para o Application Composer.
5. (Opcional) Para escolher outro nome para o bucket do Amazon S3 criado pelo Lambda, insira um novo nome e escolha Confirmar e criar projeto. Os nomes de buckets do Amazon S3 devem ser exclusivos no mundo todo e seguir as regras de nomenclatura de buckets.
6. Para salvar seus arquivos de projeto e função no Application Composer, ative o modo de sincronização local.

> **Note**
>
> Se você já usou o recurso Exportar para o Application Composer antes e criou um bucket do Amazon S3 usando o nome padrão, o Lambda pode reutilizar esse bucket, caso ele ainda exista. Aceite o nome padrão do bucket na caixa de diálogo para reutilizar o bucket existente.

Configuração de transferência do bucket do Amazon S3

O bucket do Amazon S3 que o Lambda cria para transferir a configuração da função criptografa automaticamente os objetos usando o padrão de criptografia AES 256. O Lambda também configura o bucket para usar a condição de proprietário do bucket a fim de garantir que apenas sua Conta da AWS possa adicionar objetos ao bucket.

O Lambda configura o bucket para excluir automaticamente os objetos dez dias após o upload. No entanto, o Lambda não exclui automaticamente o bucket em si. Para excluir o bucket da sua Conta da AWS, siga as instruções em Excluir um bucket. O nome padrão do bucket usa o prefixo `lambdasam`, uma sequência alfanumérica de dez dígitos, e a Região da AWS em que você criou sua função:

```
lambdasam-06f22da95b-us-east-1
```

Para evitar que cobranças adicionais sejam incluídas na sua Conta da AWS, recomendamos que você exclua o bucket do Amazon S3 assim que terminar de exportar sua função para o Application Composer.

Os preços padrão do Amazon S3 se aplicam.

Permissões obrigatórias

Para usar a integração do Lambda com o recurso Application Composer, você precisa de certas permissões para baixar um modelo do AWS SAM e gravar a configuração da sua função no Amazon S3.

Para baixar um modelo do AWS SAM, você deve ter permissão para usar as seguintes ações de API:

- GetPolicy
- iam:GetPolicyVersion

- iam:GetRole
- iam:GetRolePolicy
- iam:ListAttachedRolePolicies
- iam:ListRolePolicies
- iam:ListRoles

Você pode conceder permissão para usar todas essas ações adicionando a política `AWSLambda_ReadOnlyAccess` gerenciada pela AWS ao seu perfil de usuário do IAM.

Para que o Lambda grave a configuração da sua função no Amazon S3, você precisa ter permissão para usar as seguintes ações de API:

- S3:PutObject
- S3:CreateBucket
- S3:PutBucketEncryption
- S3:PutBucketLifecycleConfiguration

Se não puder exportar a configuração da função para o Application Composer, verifique se sua conta tem as permissões necessárias para essas operações. Se você tiver as permissões necessárias, mas ainda assim não conseguir exportar a configuração da função, verifique se há políticas baseadas em recursos que talvez limitem o acesso ao Amazon S3.

Outros recursos

Para obter um tutorial mais detalhado sobre como criar uma aplicação sem servidor no Application Composer com base em uma função do Lambda existente, consulte the section called “Infraestrutura como código (IaC)”.

Para usar o Application Composer e o AWS SAM para projetar e implantar uma aplicação completa sem servidor usando o Lambda, você também pode seguir o tutorial do AWS Application Composer em AWS Serverless Patterns Workshop.

Usar o Lambda com CloudWatch Logs

Você pode usar uma função do Lambda para monitorar e analisar logs de uma transmissão de log do Amazon CloudWatch Logs. Crie inscrições para um ou mais fluxos de log para invocar uma função quando os logs forem criados ou corresponderem a um padrão opcional. Use a função para enviar uma notificação ou manter o log em um banco de dados ou armazenamento.

O CloudWatch Logs invoca a função de forma assíncrona com um evento que contém dados de log. O valor do campo de dados é um arquivo .gzip codificado em base64.

Example Evento de mensagem do CloudWatch Logs

```
{
  "awslogs": {
    "data":
 "ewogICAgIm1lc3NhZ2VUeXBlIjogIkRBVEFfTUVTU0FHRSIsCiAgICAib3duZXIiOiAiMTIzNDU2Nzg5MDEyIiwKICAgI
  }
}
```

Quando decodificados e descompactados, os dados de log compõem um documento JSON com a seguinte estrutura:

Example Dados de mensagem do CloudWatch Logs (decodificados)

```
{
    "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"
        }
    ]
```

```
}
```

Usar o AWS Lambda com o AWS CloudFormation

Em um modelo do AWS CloudFormation, você pode especificar uma função do Lambda como o destino de um recurso personalizado. Use recursos personalizados para processar parâmetros, recuperar valores de configurações ou chamar outros serviços da AWS durante eventos de ciclo de vida da pilha.

O exemplo a seguir invoca uma função que foi definida em outro lugar do modelo.

Example – Definição de recurso personalizado

```
Resources:
  primerinvoke:
    Type: AWS::CloudFormation::CustomResource
    Version: "1.0"
    Properties:
      ServiceToken: !GetAtt primer.Arn
      FunctionName: !Ref randomerror
```

O token do serviço é o nome de recurso da Amazon (ARN) da função que o AWS CloudFormation invoca quando você cria, atualiza ou exclui a pilha. Você também pode incluir propriedades, como `FunctionName`, que o AWS CloudFormation transmite à sua função como está.

O AWS CloudFormation invoca a função do Lambda de forma assíncrona com um evento que inclui um URL de retorno de chamada.

Example – Evento de mensagem do AWS CloudFormation

```
{
    "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"
    }
}
```

A função é responsável por retornar uma resposta ao URL de retorno que indica êxito ou falha. Para sintaxe de resposta completa, consulte Objetos de resposta de recursos personalizados.

Example – Resposta de recursos personalizados do AWS CloudFormation

```
{
    "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"
}
```

O AWS CloudFormation fornece uma biblioteca chamada `cfn-response` que lida com o envio da resposta. Se definir a função dentro de um modelo, você poderá exigir a biblioteca pelo nome. O AWS CloudFormation adiciona a biblioteca ao pacote de implantação que cria para a função.

Se sua função usada por um recurso personalizado tiver uma interface de rede elástica vinculada a ela, adicione os recursos a seguir à política da VPC, onde **`region`** é a região em que a função está, sem os traços. Por exemplo, `us-east-1` é `useast1`. Isso permitirá que o recurso personalizado responda ao URL de retorno de chamada que envia um sinal de volta para a pilha do AWS CloudFormation.

```
arn:aws:s3:::cloudformation-custom-resource-response-region",
"arn:aws:s3:::cloudformation-custom-resource-response-region/*",
```

O exemplo a seguir invoca uma segunda função. Se a chamada tiver êxito, a função enviará uma resposta de êxito ao AWS CloudFormation e a atualização da pilha continuará. O modelo usa o tipo de recurso AWS::Serverless::Function fornecido pelo AWS Serverless Application Model.

Example – Função de recurso personalizado

```
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 a função que o recurso personalizado invocar não estiver definida em um modelo, você poderá obter o código fonte para o `cfn-response` no módulo cfn-response no Manual do usuário do AWS CloudFormation.

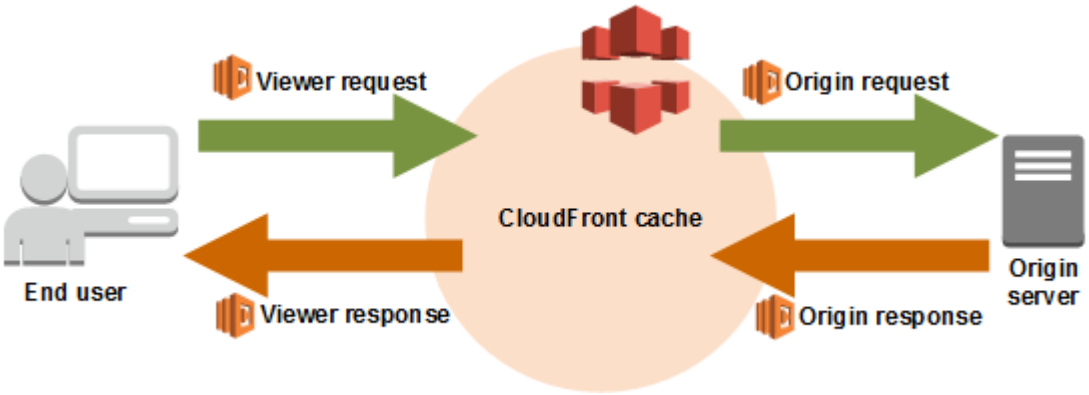
Para obter mais informações sobre recursos personalizados, consulte Recursos personalizadosno Guia do usuário do AWS CloudFormation.

Usando AWS Lambda com o CloudFront Lambda @Edge

O Lambda @Edge é uma extensão AWS Lambda que permite implantar funções do Python e do Node.js nos pontos de presença da Amazon. CloudFront Um caso de uso comum do Lambda @Edge é usar funções para personalizar o conteúdo que sua CloudFront distribuição entrega aos usuários finais. Invocar essas solicitações em locais mais próximos do visualizador, em vez de em servidores de origem, reduz significativamente a latência e melhora a experiência do usuário.

Quando você associa uma CloudFront distribuição a uma função Lambda @Edge, CloudFront intercepta solicitações e respostas em CloudFront pontos de presença. CloudFront em seguida, invoca sua função Lambda enviando um evento. Você pode CloudFront invocar sua função Lambda quando os seguintes eventos ocorrerem:

- Quando CloudFront recebe uma solicitação de um visualizador (solicitação do visualizador)
- Antes de CloudFront encaminhar uma solicitação para a origem (solicitação de origem)
- Quando CloudFront recebe uma resposta da origem (resposta de origem)
- Before CloudFront retorna a resposta ao espectador (resposta do espectador)



> **Note**
>
> O Lambda@Edge oferece suporte a um conjunto limitado de tempos de execução e recursos. Para obter detalhes, consulte Requisitos e restrições sobre funções Lambda no guia do CloudFront desenvolvedor da Amazon.

Veja a seguir um exemplo de um CloudFront evento.

Example CloudFront evento de mensagem

```
{
  "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"
              }
            ]
          }
        }
      }
    }
  ]
}
```

Para obter mais informações sobre o uso do Lambda @Edge, consulte Usando com o CloudFront Lambda @Edge.

Usar o AWS Lambda com o AWS CodeCommit

Você pode criar um acionador para um repositório do AWS CodeCommit de modo que eventos no repositório chamem uma função do Lambda. Por exemplo, você pode invocar uma função do Lambda quando uma ramificação ou etiqueta é criada ou quando um push é feito para uma ramificação existente.

Example Evento de mensagem do 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"
        }
    ]
}
```

Para obter mais informações, consulte Gerenciar triggers para um repositório do AWS CodeCommit.

Usar o AWS Lambda com o Amazon Cognito

O recurso de eventos do Amazon Cognito permite executar funções do Lambda em resposta a eventos no Amazon Cognito. O Amazon Cognito fornece autenticação, autorização e gerenciamento de usuários para seus aplicativos Web e móveis. É possível invocar uma função do Lambda em resposta a eventos importantes no Amazon Cognito. Por exemplo, usando os eventos do acionador de sincronização, é possível invocar uma função do Lambda que está publicada sempre que um conjunto de dados é sincronizado. Para saber mais e obter um exemplo prático, consulte Introducing Amazon Cognito Events: Sync Triggers no blog Mobile Development (Desenvolvimento para dispositivos móveis).

Example Evento de mensagem do 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
}
```

Você configura o mapeamento de fonte do evento usando a configuração de assinatura de eventos do Amazon Cognito. Para obter informações sobre o mapeamento de origem do evento e um evento de exemplo, consulte Amazon Cognito events no Amazon Cognito Developer Guide.

Usar o Lambda com o Amazon Connect

Você pode usar uma função do Lambda para processar solicitações do Amazon Connect. É possível usar o Amazon Connect para criar uma central de atendimento na nuvem.

O Amazon Connect invoca a sua função do Lambda de forma síncrona com um evento que contém o corpo da solicitação e os metadados.

Example Evento de solicitação do 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"
}
```

Para obter informações sobre como usar o Amazon Connect com o Lambda, consulte Invocar funções do Lambda no Guia do administrador do Amazon Connect.

Usar o AWS Lambda com o Amazon EC2

Você pode usar o AWS Lambda para processar eventos de ciclo de vida do Amazon Elastic Compute Cloud e gerenciar recursos do Amazon EC2. O Amazon EC2 envia eventos ao Amazon EventBridge (CloudWatch Events) para eventos de ciclo de vida, como quando uma instância muda de estado, quando um snapshot de volume do Amazon Elastic Block Store é concluído ou quando uma instância spot está programada para ser encerrada. Configure o EventBridge (CloudWatch Events) para encaminhar esses eventos para uma função do Lambda para processamento.

O EventBridge (CloudWatch Events) invoca sua função do Lambda de forma assíncrona com o documento de evento no Amazon EC2.

Example evento do ciclo de vida da instância

```
{
    "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"
    }
}
```

Para obter detalhes sobre como configurar eventos, consulte Usar o Lambda com o Agendador do Amazon EventBridge. Para obter uma função de exemplo que processa notificações de snapshot do Amazon EBS, consulte EventBridge Scheduler para Amazon EBS.

Você também pode usar o AWS SDK para gerenciar instâncias e outros recursos com a API do Amazon EC2.

Permissões

Para processar eventos de ciclo de vida do Amazon EC2, o EventBridge (CloudWatch Events) precisa de permissão para invocar sua função. Essa permissão vem da política baseada em recursos da função. Se você usar o console do EventBridge (CloudWatch Events) para configurar um acionador de evento, o console atualizará a política baseada em recursos em seu nome. Caso contrário, adicione uma instrução como a seguinte:

Example instrução de política baseada em recursos para notificações de ciclo de vida do 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/*"
    }
  }
}
```

Para adicionar uma instrução, use o comando `add-permission` da 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 sua função usar o AWS SDK para gerenciar recursos do Amazon EC2, adicione permissões do Amazon EC2 à função de execução da função.

Tutorial: configurar uma função do Lambda para acessar o Amazon ElastiCache em uma Amazon VPC

Para saber como configurar o Lambda para acessar o Amazon ElastiCache em uma Amazon VPC, consulte o Lambda tutorial no ElastiCache for Redis User Guide.

Processar solicitações do Application Load Balancer com o Lambda

Você pode usar uma função do Lambda para processar solicitações de um Application Load Balancer. O Elastic Load Balancing é compatível com funções do Lambda como destino para um Application Load Balancer. Use regras de load balancer para rotear solicitações HTTP para uma função, com base no caminho ou em valores de cabeçalho. Processe a solicitação e retorne uma resposta HTTP de sua função do Lambda.

O Elastic Load Balancing invoca a função do Lambda de forma síncrona com um evento que contém o corpo da solicitação e os metadados.

Example Evento de solicitação do 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
}
```

Sua função processa o evento e retorna um documento de resposta para o balanceador de carga no JSON. O Elastic Load Balancing converte o documento em uma resposta de sucesso ou de erro de HTTP e o devolve ao usuário.

Example formato do documento de resposta

```
{
    "statusCode": 200,
    "statusDescription": "200 OK",
    "isBase64Encoded": False,
    "headers": {
        "Content-Type": "text/html"
    },
    "body": "<h1>Hello from Lambda!</h1>"
}
```

Para configurar um Application Load Balancer como um acionador de função, conceda ao Elastic Load Balancing permissão para executar a função, crie um grupo de destino que roteie as solicitações para a função e adicione uma regra ao balanceador de carga que envie solicitações para o grupo de destino.

Use o comando `add-permission` para adicionar uma instrução de permissão à política baseada em recursos de sua função.

```
aws lambda add-permission --function-name alb-function \
--statement-id load-balancer --action "lambda:InvokeFunction" \
--principal elasticloadbalancing.amazonaws.com
```

A seguinte saída deverá ser mostrada:

```
{
    "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\"}"
}
```

Para obter instruções sobre como configurar o listener e um grupo de destino do balanceador de carga da aplicação, consulte Funções do Lambda como destino no Guia do usuário para Application Load Balancers.

Uso do Amazon EFS com o Lambda

O Lambda se integra ao Amazon Elastic File System (Amazon EFS) para oferecer acesso seguro e compartilhado ao sistema de arquivos para aplicativos do Lambda. É possível configurar funções para montar um sistema de arquivos durante a inicialização com o protocolo NFS na rede local em uma VPC. O Lambda gerencia a conexão e criptografa todo o tráfego de e para o sistema de arquivos.

O sistema de arquivos e a função do Lambda devem estar na mesma região. Uma função do Lambda em uma conta pode montar um sistema de arquivos em uma conta diferente. Nesse cenário, você configura um emparelhamento de VPCs entre a VPC da função e a VPC do sistema de arquivos.

> **Note**
>
> Para configurar uma função para se conectar a um sistema de arquivos, consulte Configurar o acesso ao sistema de arquivos para funções do Lambda.

O Amazon EFS oferece suporte ao bloqueio de arquivos para evitar a corrupção se várias funções tentarem gravar no mesmo sistema de arquivos ao mesmo tempo. O bloqueio do Amazon EFS segue o protocolo NFS v4.1 para bloqueios recomendados, além de permitir que seus aplicativos usem bloqueios de arquivo inteiro ou de intervalos de bytes.

O Amazon EFS fornece opções para personalizar o sistema de arquivos com base na necessidade da aplicação de manter a alta performance em escala. Há três principais fatores a serem considerados: o número de conexões, o throughput (em MiB por segundo) e as IOPS.

> **Cotas**
>
> Para obter detalhes sobre cotas e limites do sistema de arquivos, consulte Cotas para sistemas de arquivos do Amazon EFS no Manual do usuário do Amazon Elastic File System.

Para evitar problemas com escalabilidade, taxa de transferência e IOPS, monitore as métricas que o Amazon EFS envia para a Amazon. CloudWatch Para obter uma visão geral do monitoramento no Amazon EFS, consulteMonitoramento do EFSnoGuia do Usuário do Amazon Elastic File System.

Seções

Conexões

O Amazon EFS oferece suporte para até 25.000 conexões por sistema de arquivos. Durante a inicialização, cada instância de uma função cria uma única conexão com o sistema de arquivos que persiste entre invocações. Isso significa que você pode alcançar a simultaneidade de 25.000 em uma ou mais funções conectadas a um sistema de arquivos. Para limitar o número de conexões criadas por uma função, use a simultaneidade reservada.

No entanto, quando você faz alterações no código ou na configuração da função em escala, ocorre um aumento temporário no número de instâncias da função além da simultaneidade atual. O Lambda provisiona novas instâncias para lidar com novas solicitações e existe um atraso antes que as instâncias antigas fechem suas conexões com o sistema de arquivos. Para evitar atingir o limite máximo de conexões durante uma implantação, use implantações contínuas. Com implantações contínuas, você gradualmente muda o tráfego para a nova versão sempre que fizer uma alteração.

Se você se conectar ao mesmo sistema de arquivos de outros serviços, como o Amazon EC2, esteja ciente também do comportamento de escalabilidade das conexões no Amazon EFS. Um sistema de arquivos oferece suporte para a criação de até 3.000 conexões em uma intermitência, após a qual oferece suporte para 500 novas conexões por minuto.

Para monitorar e acionar um alarme nas conexões, use a métrica `ClientConnections`.

Throughput

Em escala, também é possível exceder o throughput máximo de um sistema de arquivos. No modo intermitente (padrão), um sistema de arquivos tem um baixo throughput de linha de base que é dimensionado linearmente com o seu tamanho. Para permitir intermitências de atividade, o sistema de arquivos recebe créditos de intermitência que permitem o uso de 100 MiB/s ou mais de throughput. Os créditos são acumulados continuamente e são gastos em cada operação de leitura e gravação. Se o sistema de arquivos ficar sem créditos, ele limitará as operações de leitura e gravação que ultrapassarem o throughput de linha de base, o que pode fazer com que as invocações atinjam o tempo limite.

> **Note**
>
> Se você usar a simultaneidade provisionada, a função poderá consumir créditos de intermitência mesmo quando estiver ociosa. Com a simultaneidade provisionada, o Lambda inicializa as instâncias da função antes dela ser invocada e recicla as instâncias em intervalos regulares. Se você usar arquivos em um sistema de arquivos anexado durante a inicialização, essa atividade poderá usar todos os créditos de intermitência.

Para monitorar e acionar um alarme no throughput, use a métrica `BurstCreditBalance`. Ela deverá aumentar quando a simultaneidade da função estiver baixa e diminuir quando ela estiver alta. Se ela sempre diminuir ou não acumular durante a baixa atividade o suficiente para cobrir o tráfego de pico, talvez seja necessário limitar a simultaneidade da função ou ativar o throughput provisionado.

IOPS

As operações de entrada/saída por segundo (IOPS) é uma medida do número de operações de leitura e gravação processadas pelo sistema de arquivos. No modo de uso geral, as IOPS são limitadas a favor da menor latência, o que é benéfico para a maioria dos aplicativos.

Para monitorar e criar alarmes em IOPS no modo de uso geral, use a métrica `PercentIOLimit`. Se essa métrica atingir 100%, a função poderá atingir o tempo limite aguardando a conclusão das operações de leitura e gravação.

Usar o Lambda com o Agendador do Amazon EventBridge

O Agendador do Amazon EventBridge utiliza a tecnologia sem servidor que permite criar, executar e gerenciar tarefas a partir de um serviço central gerenciado. Com o Agendador do EventBridge, você pode criar programações usando expressões cron e rate para padrões recorrentes ou configurar invocações únicas. Você pode configurar janelas de tempo flexíveis para entrega, definir limites de repetição e definir o tempo máximo de retenção para eventos não processados.

Quando você configura o EventBridge Scheduler com o Lambda, o EventBridge Scheduler invoca a função do Lambda de modo assíncrono. Esta página explica como usar o Agendador do EventBridge para invocar uma função do Lambda em uma programação.

Configurar o perfil de execução

Quando você cria uma programação, o Agendador do EventBridge deve ter permissão para invocar a operação da API de destino em seu nome. Você concede essas permissões ao Agendador do EventBridge usando um perfil de execução. A política de permissão que você anexa ao perfil de execução da programação define as permissões necessárias. As permissões dependem da API de destino que você deseja que o Agendador do EventBridge invoque.

Quando você usa o console do Agendador do EventBridge para criar programações, como no procedimento a seguir, o Agendador do EventBridge configura automaticamente um perfil de execução com base no destino selecionado. Se você quiser criar uma programação usando um dos SDKs do Agendador do EventBridge, a AWS CLI ou o AWS CloudFormation, você deve ter um perfil de execução que conceda as permissões que o Agendador do EventBridge precisa para invocar o destino. Para obter mais informações sobre como configurar manualmente um perfil de execução para programações, consulte Como configurar um perfil de execução no Guia do usuário do Agendador do EventBridge.

Criar uma programação

Para criar uma programação usando o console

1. Abra o console do Agendador do Amazon EventBridge em https://console.aws.amazon.com/scheduler/home.
2. Na página Programações, clique em Criar programação.
3. Na página Especificar detalhes da programação, na seção Nome e descrição da programação, faça o seguinte:

a. Em Nome da programação, insira um nome para a programação. Por exemplo, **`MyTestSchedule`**.

b. (Opcional) Em Descrição, insira a descrição da programação. Por exemplo, **`My first schedule`**.

c. Em Grupo de programação, escolha um grupo de programação na lista suspensa. Se você não tiver um grupo, escolha padrão. Para criar um grupo de programação, escolha criar sua própria programação.

   Para adicionar tags a grupos de programação, você usa os grupos de programação.

4. • Escolha as opções de programação.

Ocorrência	Fazer isso...	
Programação única A programação única invoca o destino somente uma vez na data e hora que você especificar.	Em Data e hora, faça o seguinte: • Insira uma data válida no formato `YYYY/MM/DD` . • Insira um carimbo de data/hora no formato de 24 horas `hh:mm`. • Em Fuso horário, escolha o fuso horário.	
Programação recorrente A programação recorrent e invoca o destino em uma taxa especificada por você usando uma expressão cron ou rate.	a. Em Tipo de programaç ão, siga um dos procedimentos a seguir. • Para usar uma expressão cron para definir a programação, escolha Programaç ão baseada em cron e insira a expressão cron.	

Ocorrência	Fazer isso...	
	• Para usar uma expressão rate para definir a programação, escolha Programação baseada em rate e insira a expressão rate. Para obter mais informações sobre as expressões cron e rate, consulte Tipos de programação no Agendador do EventBridge no Guia do usuário do Agendador do Amazon EventBridge. b. Em Janela de tempo flexível, escolha Desativar para desativar a opção ou escolha uma das janelas de tempo predefinidas. Por exemplo, se você escolher 15 minutos e definir uma programação recorrente para invocar o destino uma vez a cada hora, a programação será executada em até 15 minutos após o início de cada hora.	

5. (Opcional) Se você escolher Programação recorrente na etapa anterior, na seção Período, faça o seguinte:

   a. Em Fuso horário, escolha um fuso horário.

   b. Em Data e hora de início, insira uma data válida no formato `YYYY/MM/DD` e, em seguida, especifique um carimbo de data/hora no formato de 24 horas `hh:mm`.

   c. Para Data e hora de término, insira uma data válida no formato `YYYY/MM/DD` e, em seguida, especifique um carimbo de data/hora no formato 24 horas `hh:mm`.

6. Escolha Próximo.

7. Na página Selecionar destino, escolha a operação de API da AWS que o Agendador do EventBridge invoca:

   a. Escolha Invocar o AWS Lambda.

   b. Na seção Invocar, selecione uma função ou escolha Criar uma nova função do Lambda.

   c. (Opcional) Insira uma carga útil JSON. Se você não inserir uma carga útil, o Agendador do EventBridge usará um evento vazio para invocar a função.

8. Escolha Próximo.

9. Na página Configurações, faça o seguinte:

   a. Para ativar a programação, em Estado da programação, mude para Ativar programação.

   b. Para configurar uma política de novas tentativas para a programação, em Política de novas tentativas e fila de mensagens não entregues (DLQ), faça o seguinte:

      - Mude para Tentar novamente.
      - Em Duração máxima do evento, insira o período máximo de horas e minutos que o Agendador do EventBridge deve manter um evento não processado.
      - O período máximo é de 24 horas.
      - Em Máximo de tentativas, insira o número máximo de vezes que o Agendador do EventBridge tentará executar a programação se o destino retornar um erro.

        O valor máximo é 185 tentativas.

      Com as políticas de novas tentativas, se a programação não conseguir invocar o destino, o Agendador do EventBridge tentará executar novamente a programação. Se configurado, você deve definir o tempo máximo de retenção e as novas tentativas da programação.

c. Escolha onde o Agendador do EventBridge armazena os eventos não entregues.

Opção Fila de mensagens não entregues (DLQ)	Fazer isso...
Não armazene	Selecione Nenhum.
Armazenar o evento na mesma Conta da AWS em que você está criando a programação	a. Escolha Selecionar uma fila do Amazon SQS na minha Conta da AWS como uma DLQ. b. Escolha o nome do recurso da Amazon (ARN) da fila do Amazon SQS.
Armazenar o evento em uma Conta da AWS diferente de onde você está criando a programação	a. Escolha Especificar uma fila do Amazon SQS em outras Contas da AWS como uma DLQ. b. Insira o nome do recurso da Amazon (ARN) da fila do Amazon SQS.

d. Para usar uma chave gerenciada pelo cliente para criptografar a entrada de destino, em Criptografia, escolha Personalizar as configurações de criptografia (avançado).

   Se você escolher essa opção, insira o ARN da chave do KMS existente ou escolha Criar AWS KMS key para navegar até o console do AWS KMS. Para obter mais informações sobre como o Agendador do EventBridge criptografa os dados em repouso, consulte Criptografia em repouso no Guia do usuário do Agendador do Amazon EventBridge.

e. Para que o Agendador do EventBridge crie um novo perfil de execução para você, escolha Criar novo perfil para esta programação. Depois, insira um nome em Nome do perfil. Se você escolher essa opção, o Agendador do EventBridge anexará as permissões necessárias para o destino de exemplo ao perfil.

10. Escolha Próximo.

11. Na página Revisar e criar programação, revise os detalhes da programação. Em cada seção, escolha Editar para voltar a essa etapa e editar seus detalhes.
12. Clique em Criar programação.

    Você pode ver a lista com as programações novas e existentes na página Programações. Na coluna Status, verifique se sua nova programação está Ativada.

Para confirmar que o Agendador do EventBridge invocou a função, verifique os logs do Amazon CloudWatch da função.

Recursos relacionados

Para obter mais informações sobre o Agendador do EventBridge, consulte:

- Guia do usuário do Agendador do EventBridge
- Referência da API do Agendador do EventBridge
- Preços do Agendador do EventBridge

Usar o AWS Lambda com o AWS IoT

O AWS IoT fornece comunicação segura entre dispositivos conectados à Internet (como sensores) e a Nuvem AWS. Isso permite que você colete, armazene e analise dados de telemetria de vários dispositivos.

É possível criar regras de AWS IoT para que seus dispositivos interajam com os serviços da AWS. O Mecanismo de regras do AWS IoT fornece uma linguagem baseada em SQL para selecionar dados de cargas de mensagem e enviar os dados para outros serviços, como o Amazon S3, o Amazon DynamoDB e o AWS Lambda. Defina uma regra para invocar uma função do Lambda quando quiser invocar outro serviço da AWS ou um serviço de terceiros.

Quando uma mensagem de entrada da IoT aciona a regra, o AWS IoT invoca a função do Lambda de forma assíncrona e passa os dados da mensagem da IoT para a função.

O exemplo a seguir mostra uma leitura de umidade do sensor de uma estufa. Os valores de linha e pos identificam a localização do sensor. Esse evento de exemplo é baseado no tipo de estufa nos Tutoriais de regras da AWS IoT.

Example Evento de mensagem do AWS IoT

```
{
    "row" : "10",
    "pos" : "23",
    "moisture" : "75"
}
```

Para a invocação assíncrona, o Lambda coloca em fila a mensagem e tenta novamente caso a sua função retorne um erro. Configure sua função com um destino para manter os eventos que a função não puder processar.

É necessário conceder permissão para que o serviço da AWS IoT invoque sua função do Lambda. Use o comando `add-permission` para adicionar uma instrução de permissão à política baseada em recursos de sua função.

```
aws lambda add-permission --function-name my-function \
--statement-id iot-events --action "lambda:InvokeFunction" --principal
 iot.amazonaws.com
```

A seguinte saída deverá ser mostrada:

```
{
    "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\"}"
}
```

Para obter mais informações sobre como usar o Lambda com o AWS IoT, consulte Criar uma regra do AWS Lambda.

Usar o AWS Lambda com o Amazon Data Firehose

O Amazon Data Firehose captura, transforma e carrega dados de transmissão em serviços downstream, como o Managed Service for Apache Flink ou o Amazon S3. Você pode escrever funções do Lambda para solicitar processamento personalizado adicional dos dados antes que eles sejam enviados downstream.

Example Evento de mensagem do 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": ""
      }
    }
  ]
}
```

Para obter mais informações, consulte Transformação de dados do Amazon Data Firehose no Guia do desenvolvedor do Firehose.

O uso doAWS LambdaCom o Amazon Lex

É possível usar o Amazon Lex para integrar um bot de conversação à sua aplicação. O bot do Amazon Lex fornece uma interface de conversação com seus usuários. O Amazon Lex fornece integração pré-criada com o Lambda, que permite usar uma função do Lambda com o bot do Amazon Lex.

Ao configurar um bot do Amazon Lex, é possível especificar uma função do Lambda para executar validação, atendimento ou ambos. Para validação, o Amazon Lex invoca a função do Lambda após cada resposta do usuário. A função do Lambda pode validar a resposta e fornecer feedback corretivo para o usuário, se necessário. Para atendimento, o Amazon Lex invoca a função do Lambda para atender à solicitação do usuário depois que o bot coleta com êxito todas as informações necessárias e recebe a confirmação do usuário.

É possível gerenciar a simultaneidade da função do Lambda para controlar o número máximo de conversas simultâneas do bot atendidas. A API do Amazon Lex retornará um código de status HTTP 429 — Too Many Requests (Excesso de solicitações) se a função estiver na simultaneidade máxima.

A API retornará um código de status HTTP 424 — Dependency Failed Exception (Exceção com falha de dependência) se a função do Lambda lançar uma exceção.

O bot do Amazon Lex invoca a função do Lambda de forma síncrona. O parâmetro de evento contém informações sobre o bot e o valor de cada slot na caixa de diálogo. Para obter as definições dos campos de evento e de resposta, consulte Lambda event and response format (Formato de evento e resposta do Lambda) no Guia do desenvolvedor do Amazon Lex. O `invocationSource` parâmetro no evento de mensagem do Amazon Lex indica se a função Lambda deve validar as entradas (DialogCodeHook) ou cumprir a intenção (). FulfillmentCodeHook

Para obter um exemplo de tutorial que mostra como usar o Lambda com o Amazon Lex, consulte Exercício 1: criar um bot do Amazon Lex usando um esquema no Guia do desenvolvedor do Amazon Lex.

Funções e permissões

É necessário configurar uma função vinculada ao serviço comoFunção de execução do. O Amazon Lex define a função vinculada ao serviço com permissões predefinidas. Ao criar um bot do Amazon Lex usando o console, a função vinculada ao serviço é criada automaticamente. Para criar um perfil vinculado ao serviço com a AWS CLI, use o comando `create-service-linked-role`.

```
aws iam create-service-linked-role --aws-service-name lex.amazonaws.com
```

Esse comando cria a função a seguir.

```
{
  "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 a função do Lambda usar outros serviços da AWS, será necessário adicionar as permissões correspondentes à função vinculada ao serviço.

Você usa uma política de permissões baseada em recursos para permitir que o bot do Amazon Lex invoque a função do Lambda. Se você usar o console do Amazon Lex, a política de permissões será criada automaticamente. Na AWS CLI, use o comando `add-permission` do Lambda para definir a permissão.

Para o Amazon Lex V2, execute o comando a seguir. No ARN de origem, substitua `us-east-1` pela Região da AWS em que seu bot do Amazon Lex está, e use o número e alias do bit da sua própria Conta da AWS.

```
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"
```

Você também pode usar o Amazon Lex V1 para invocar uma função do Lambda. Para o Amazon Lex V1, execute o comando a seguir. No ARN de origem, substitua `us-east-1` pela Região da AWS em que a intenção do Amazon Lex está e use o número e intenção da sua própria Conta da AWS.

```
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:*"
```

Observe que o Amazon Lex V1 não recebe mais manutenção. Recomendamos que você use o Amazon Lex V2.

Usar o AWS Lambda com o Amazon RDS

Você pode conectar uma função do Lambda a um banco de dados do Amazon Relational Database Service (Amazon RDS) diretamente e por meio de um Amazon RDS Proxy. As conexões diretas são úteis em cenários simples e os proxies são recomendados para produção. Um proxy de banco de dados gerencia um pool de conexões de banco de dados compartilhadas, o que permite que sua função atinja altos níveis de simultaneidade sem esgotar as conexões de banco de dados.

Recomendamos o uso do Amazon RDS Proxy para funções do Lambda que fazem conexões curtas e frequentes com o banco de dados ou que abrem e fecham um grande número de conexões de banco de dados.

Configurar a função

No console do Lambda, você pode provisionar e configurar instâncias de banco de dados e recursos de proxy do Amazon RDS. Para obter mais informações, consulte Bancos de dados do RDS na guia Configuração. Como alternativa, você também pode criar e configurar conexões com funções do Lambda no console do Amazon RDS.

- Para se conectar a um banco de dados, a função deve estar na mesma Amazon VPC em que o banco de dados é executado.
- Você pode usar bancos de dados do Amazon RDS com mecanismos MySQL, MariaDB, PostgreSQL ou Microsoft SQL Server.
- Você também pode usar clusters de banco de dados do Aurora com mecanismos MySQL ou PostgreSQL.
- Você precisa fornecer um segredo do Secrets Manager para a autenticação do banco de dados.
- Um perfil do IAM deve fornecer permissão para usar o segredo e uma política de confiança deve permitir que o Amazon RDS assuma o perfil.
- A entidade principal do IAM que usa o console para configurar o recurso Amazon RDS e conectá-lo à sua função deve ter as seguintes permissões:

> **Note**
>
> Você só precisa das permissões do Amazon RDS Proxy se configurar um Amazon RDS Proxy para gerenciar um pool de suas conexões de banco de dados.

Example política de permissão

```
{
  "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": "*"
    }
  ]
}
```

O Amazon RDS cobra uma taxa horária por proxies com base no tamanho da instância do banco de dados. Consulte Preço do RDS Proxy para obter detalhes. Para obter mais informações sobre conexões de proxy, consulte Usar o Amazon RDS Proxy no Guia do usuário do Amazon RDS.

> **Configuração do Lambda e do Amazon RDS**
>
> Os consoles do Lambda e do Amazon RDS ajudarão você a configurar automaticamente alguns dos recursos necessários para fazer uma conexão entre o Lambda e o Amazon RDS.

Conectar a um banco de dados do Amazon RDS em uma função do Lambda

O exemplo de código a seguir mostra como implementar uma função do Lambda que se conecta a um banco de dados do Amazon RDS. A função faz uma solicitação simples ao banco de dados e exibe o resultado.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Conectar-se a um banco de dados do Amazon RDS em uma função do Lambda usando 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 para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Conectar-se a um banco de dados do Amazon RDS em uma função do Lambda usando 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)
  }
};
```

Processar notificações de eventos do Amazon RDS

Você pode usar o Lambda para processar notificações de eventos em um banco de dados do Amazon RDS. O Amazon RDS envia notificações para um tópico do Amazon Simple Notification Service (Amazon SNS), que você pode configurar para invocar uma função do Lambda. O Amazon SNS envolve a mensagem do Amazon RDS em seu próprio documento de evento e a envia para sua função.

Para obter mais informações sobre como configurar um banco de dados do Amazon RDS para enviar notificações, consulte Trabalhar com a notificação de eventos do Amazon RDS.

Example Mensagem do Amazon RDS em um evento do 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 sobre Lambda e Amazon RDS

- Usar uma função do Lambda para acessar um banco de dados do Amazon RDS: no Guia do usuário do Amazon RDS, saiba como usar uma função do Lambda para gravar dados em um banco de dados do Amazon RDS por meio do Amazon RDS Proxy. A função do Lambda lerá registros de uma fila do Amazon SQS e gravará novos itens em uma tabela no banco de dados sempre que uma mensagem for adicionada.

Processar notificações de eventos do Amazon S3 com o Lambda

Você pode usar o Lambda para processar notificações de eventos do Amazon Simple Storage Service. O Amazon S3 pode enviar um evento para uma função do Lambda quando um objeto é criado ou excluído. Você define as configurações de notificação em um bucket e concede permissão ao Amazon S3 para invocar uma função na política de permissões baseada em recursos da função.

> **Warning**
>
> Se a função do Lambda usar o mesmo bucket que a aciona, isso poderá fazer a função ser executada em um loop. Por exemplo, se o bucket disparar uma função sempre que um objeto for carregado e a função fizer upload de um objeto no bucket, a função vai se disparar indiretamente. Para evitar isso, use dois buckets ou configure o trigger para só se aplicar a um prefixo usado em objetos recebidos.

O Amazon S3 invoca a função de forma assíncrona com um evento que contém detalhes sobre o objeto. O exemplo a seguir mostra um evento que o Amazon S3 enviou quando um pacote de implantação foi carregado no Amazon S3.

Example Evento de notificação do 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"
        }
      }
    }
  ]
}
```

Para invocar a função, o Amazon S3 precisa de permissão da política baseada em recursos da função. Quando você configura um acionador do Amazon S3 no console do Lambda, o console modifica a política baseada em recursos para permitir que o Amazon S3 invoque a função se o nome do bucket e o ID da conta corresponderem. Se você configurar a notificação no Amazon S3, use a API do lambda para atualizar a política. Também é possível usar a API do Lambda para conceder permissão a outra conta ou restringir a permissão a um alias designado.

Se a sua função usa o AWS SDK para gerenciar recursos do Amazon S3, ela também precisa de permissões do Amazon S3 em sua função de execução.

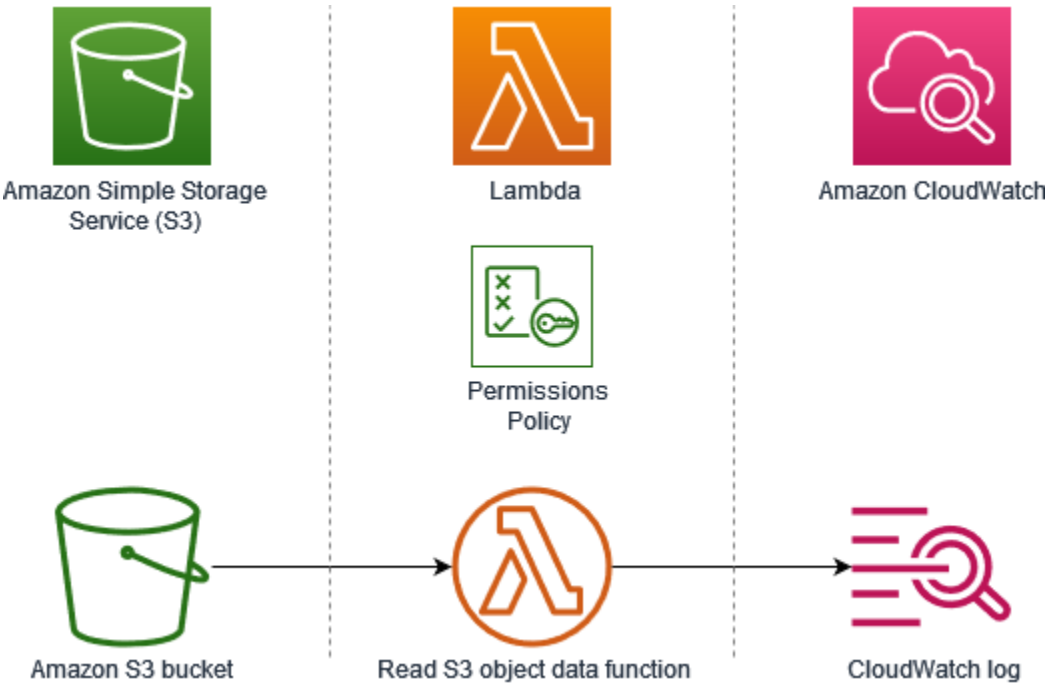
Tópicos

- Tutorial: Usar um acionador do Amazon S3 para invocar uma função do Lambda
- Tutorial: Usar um acionador do Amazon S3 para criar imagens em miniatura

Tutorial: Usar um acionador do Amazon S3 para invocar uma função do Lambda

Neste tutorial, você usará o console para criar uma função do Lambda e configurar um acionador para um bucket do Amazon Simple Storage Service (Amazon S3). Toda vez que você adiciona

um objeto ao bucket do Amazon S3, sua função é executada e envia o tipo de objeto ao Amazon CloudWatch Logs.



Este tutorial demonstra como:

1. Crie um bucket do Amazon S3.
2. Crie uma função do Lambda que retorne o tipo de objeto de um bucket do Amazon S3.
3. Configure um acionador do Lambda que invoque a função quando forem carregados objetos para o bucket.
4. Teste sua função, primeiro com um evento fictício e depois usando o acionador.

Ao concluir essas etapas, você aprenderá a configurar uma função do Lambda a ser executada sempre que forem adicionados ou excluídos objetos de um bucket do Amazon S3. Você pode concluir este tutorial usando somente o AWS Management Console.

Pré-requisitos

Cadastre-se em uma Conta da AWS

Se você ainda não tem Conta da AWS, siga as etapas a seguir para criar uma.

Para cadastrar-se em uma Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.

2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

   Quando você se cadastra em uma Conta da AWS, um Usuário raiz da conta da AWS é criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como prática recomendada de segurança, atribua o acesso administrativo a um usuário e use somente o usuário-raiz para executar tarefas que exigem acesso de usuário-raiz.

AWS envia um e-mail de confirmação depois que o processo de cadastramento é concluído. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se cadastrar em uma Conta da AWS, proteja seu Usuário raiz da conta da AWS, habilite o AWS IAM Identity Center e crie um usuário administrativo para não usar o usuário raiz em tarefas cotidianas.

Proteja seu Usuário raiz da conta da AWS

1. Faça login no AWS Management Console como o proprietário da conta ao selecionar a opção Root user (Usuário raiz) e inserir o endereço de e-mail da Conta da AWS. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário raiz, consulte Signing in as the root user (Fazer login como usuário raiz) no Guia do usuárioInício de Sessão da AWS.

2. Ative a autenticação multifator (MFA) para seu usuário raiz.

   Para obter instruções, consulte Habilitar um dispositivo MFA virtual para o usuário raiz (console)Conta da AWS no Guia do Usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center.

2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para obter um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso dos usuários com o Diretório do Centro de Identidade do IAM padrão no Guia do usuário do AWS IAM Identity Center.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário IAM Identity Center, use a URL de login enviada ao seu endereço de e-mail quando você criou o usuário IAM Identity Center user.

  Para obter ajuda com o login utilizando um usuário do IAM Identity Center, consulte Fazendo login no portal de acesso da AWS, no Guia do UsuárioInício de Sessão da AWS.

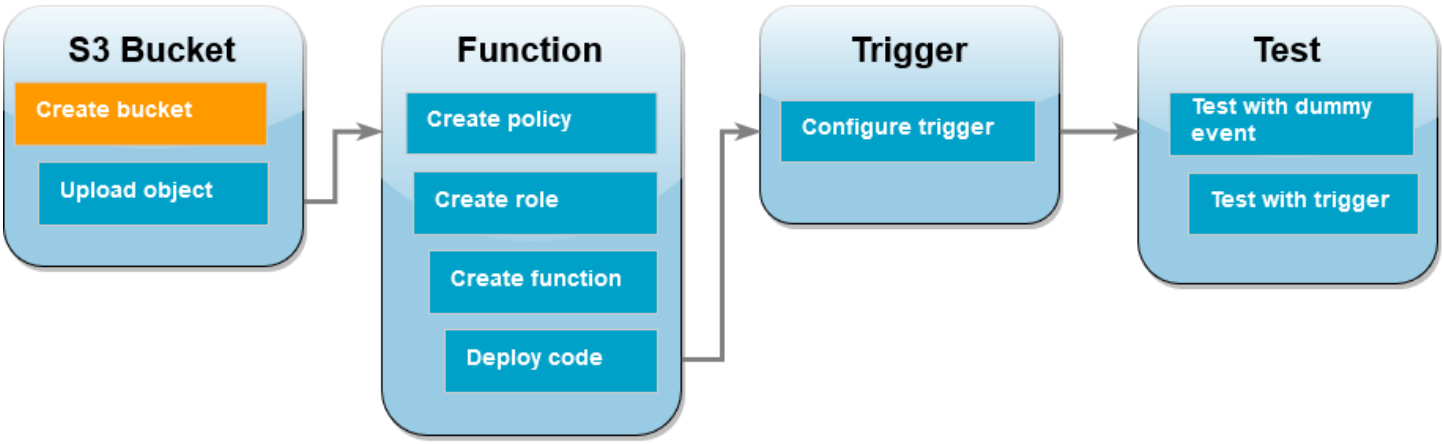
Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center.
2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center.

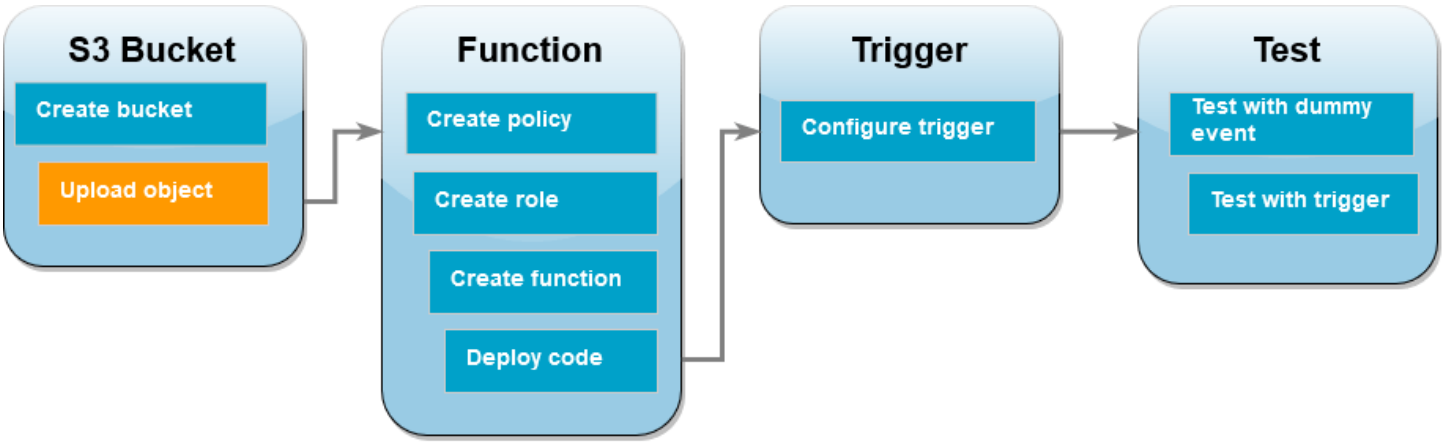
Criar um bucket do Amazon S3



Como criar um bucket do Amazon S3

1. Abra o console do Amazon S3 e selecione a página Buckets.
2. Selecione Criar bucket.
3. Em General configuration (Configuração geral), faça o seguinte:
   a. Em Nome do bucket, insira um nome global exclusivo que atenda às regras de nomenclatura de buckets do Amazon S3. Os nomes dos buckets podem conter apenas letras minúsculas, números, pontos (.) e hifens (-).
   b. Em AWS Region (Região da AWS), escolha uma região. Mais adiante no tutorial, você deverá criar sua função do Lambda na mesma região.
4. Deixe todas as outras opções com seus valores padrão e escolha Criar bucket.

Carregar um objeto de teste em um bucket

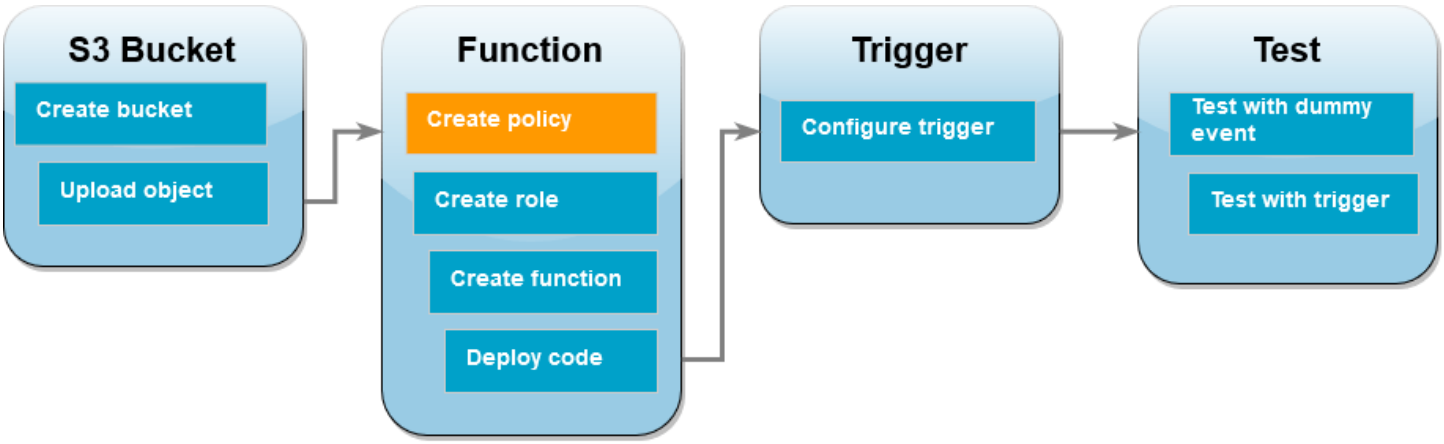


Para carregar um objeto de teste

1. Abra a página Buckets do console do Amazon S3 e escolha o bucket que você criou durante a etapa anterior.
2. Escolha Carregar.
3. Escolha Adicionar arquivos e selecione o objeto que deseja de carregar. Você pode selecionar qualquer arquivo (por exemplo, `HappyFace.jpg`).
4. Selecione Abrir e Carregar.

Posteriormente no tutorial, você testará a função do Lambda usando esse objeto.

Criação de uma política de permissões



Crie uma política de permissões que permita ao Lambda obter objetos de um bucket do Amazon S3 e gravar no Amazon CloudWatch Logs.

Para criar a política

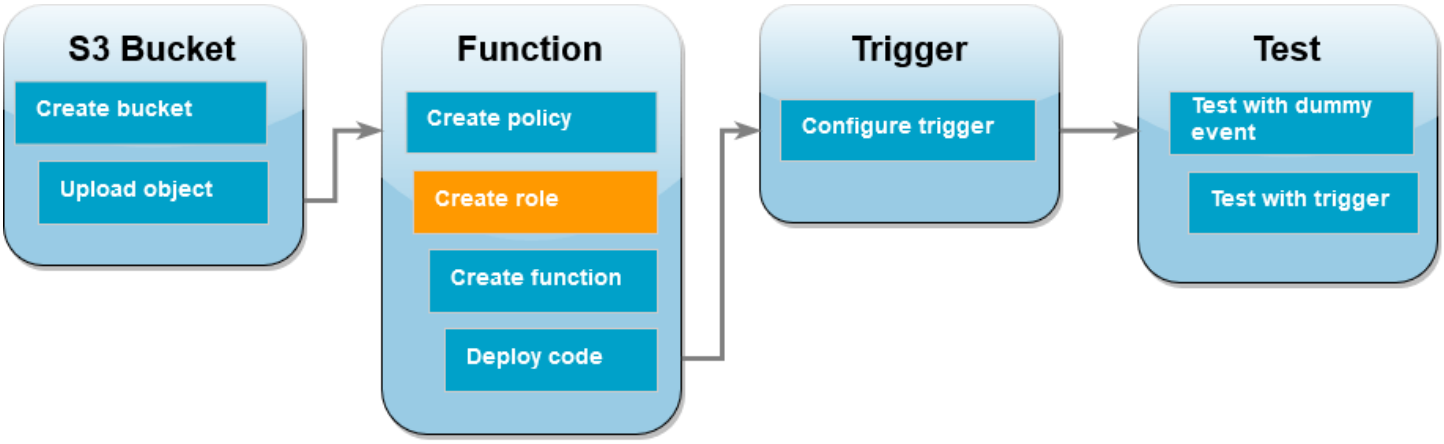
1. Abra a página Policies (Políticas) do console do IAM.
2. Escolha Criar política.
3. Escolha a guia JSON e cole a política personalizada a seguir no editor JSON.

```
{
    "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. Escolha Próximo: etiquetas.
5. Selecione Next: Review (Próximo: revisar).
6. No campo Política de revisão, em Nome da política, insira **`s3-trigger-tutorial`**.
7. Escolha Criar política.

Criar uma função de execução

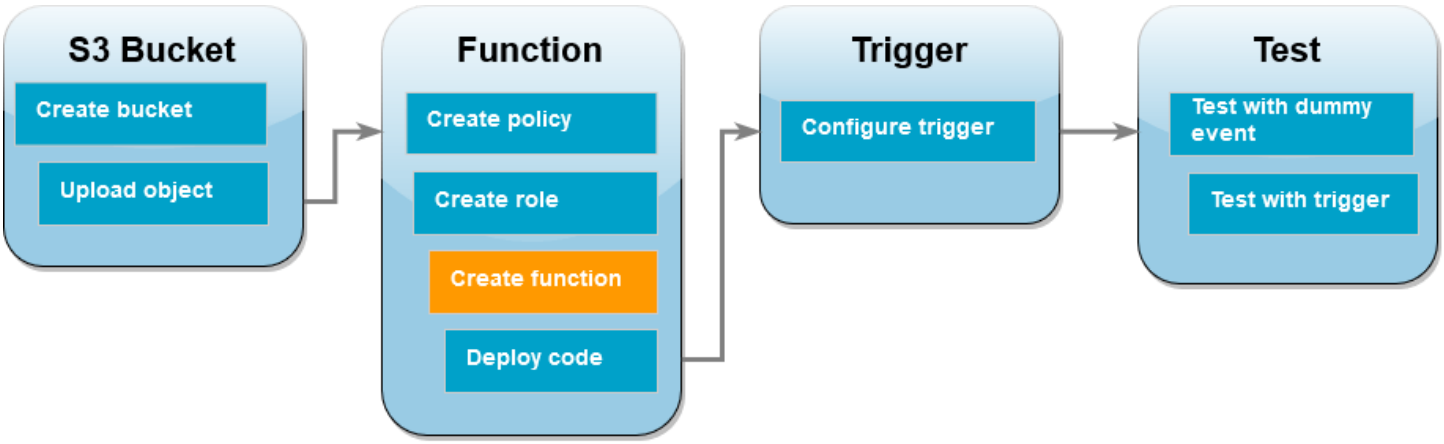


Um perfil de execução é um perfil do AWS Identity and Access Management (IAM) que concede a uma função do Lambda permissão para acessar serviços e recursos da AWS. Nesta etapa, crie um perfil de execução usando a política de permissões que criou na etapa anterior.

Para criar uma função de execução e vincular a política de permissões personalizada

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione Criar função.
3. Para o tipo de entidade confiável, escolha Serviço da AWS e, em seguida, para o caso de uso, selecione Lambda.
4. Escolha Próximo.
5. Na caixa de pesquisa de política, insira **`s3-trigger-tutorial`**.
6. Nos resultados da pesquisa, selecione a política que você criou (`s3-trigger-tutorial`) e, depois, escolha Next (Avançar).
7. Em Role details (Detalhes do perfil), para Role name (Nome do perfil), insira **`lambda-s3-trigger-role`** e, em seguida, escolha Create role (Criar perfil).

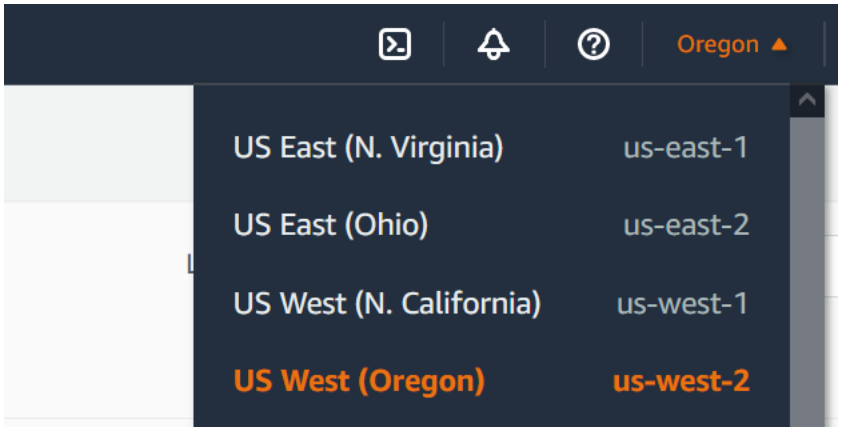
Criar a função do Lambda



Crie uma função do Lambda no console usando o runtime do Python 3.12.

Para criar a função do Lambda

1. Abra a página Funções do console do Lambda.
2. Verifique se você está trabalhando na mesma Região da AWS em que criou o bucket do Amazon S3. Você pode alterar sua região usando a lista suspensa na parte superior da tela.



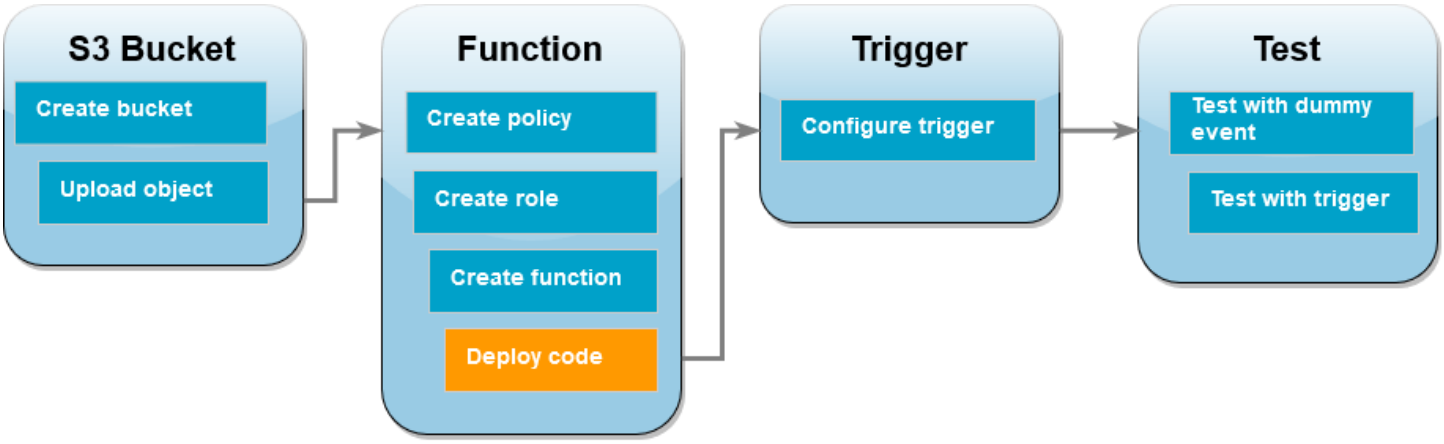
3. Escolha a opção Criar função.
4. Escolher Criar do zero
5. Em Basic information (Informações básicas), faça o seguinte:
   a. Em Nome da função, inserir `s3-trigger-tutorial`
   b. Em Runtime, selecione Python 3.12.
   c. Em Architecture (Arquitetura), escolha x86_64.
6. Na guia Alterar função de execução padrão, faça o seguinte:

a. Expanda a guia e escolha Usar uma função existente.

b. Selecione a `lambda-s3-trigger-role` que você criou anteriormente.

7. Escolha Criar função.

Implantar o código da função



Este tutorial usa o runtime do Python 3.12, mas também fornecemos exemplos de arquivos de código para outros runtimes. Você pode selecionar a guia na caixa a seguir para ver o código do runtime do seu interesse.

A função do Lambda vai recuperar o nome da chave do objeto carregado e o nome do bucket do parâmetro `event` que receberá do Amazon S3. Em seguida, a função usará o método get_object do AWS SDK for Python (Boto3) para recuperar os metadados do objeto, incluindo o tipo de conteúdo (tipo MIME) do objeto carregado.

Para implantar o código da função

1. Escolha a guia Python na caixa a seguir e copie o código.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do S3 com o Lambda usando .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do S3 com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do S3 com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do S3 com o Lambda usando JavaScript.

```
// 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);
    }
};
```

Consumir um evento do S3 com o Lambda usando TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> =>
 {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/
g, ' '));
  const params = {
    Bucket: bucket,
```

```
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make
 sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do S3 com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do S3 com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do S3 com o Lambda usando 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 para Rust

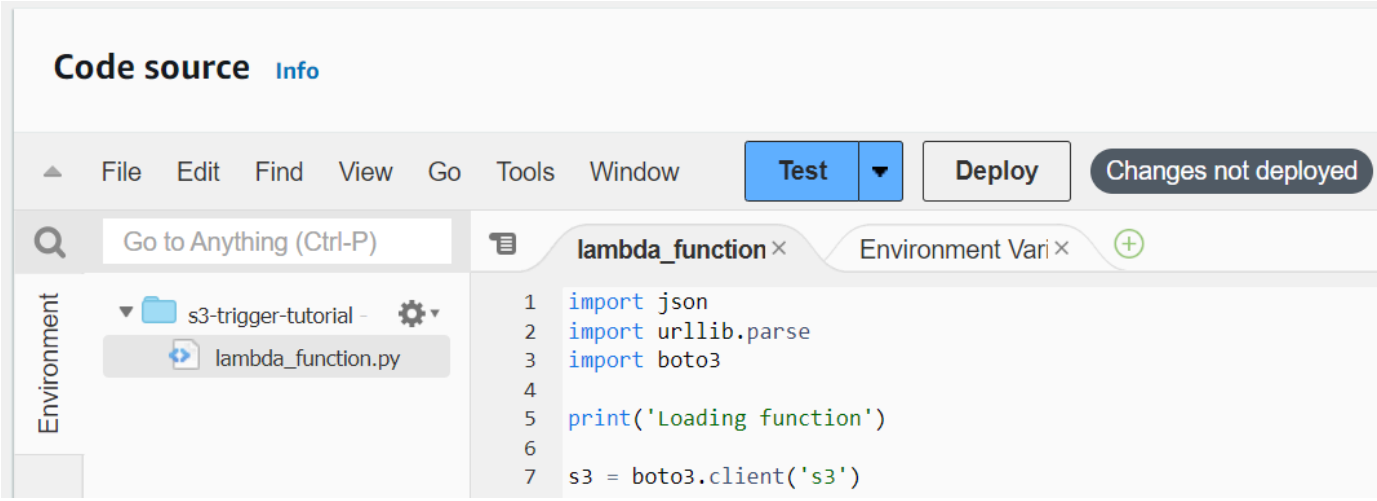
> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do S3 com o Lambda usando 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,
```

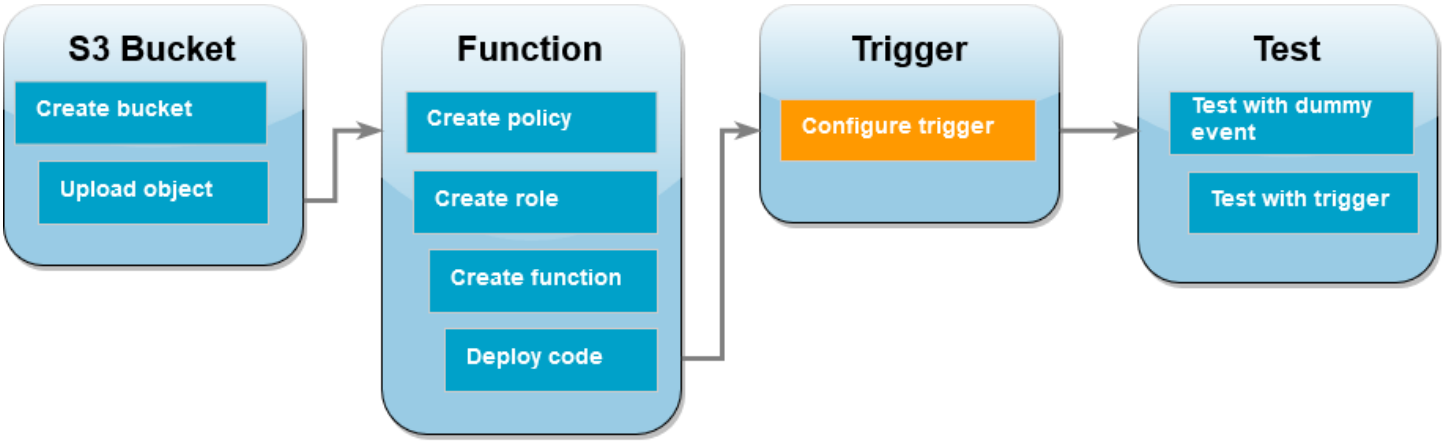
```
    evt: LambdaEvent<S3Event>
) -> Result<(), Error> {
    tracing::info!(records = ?evt.payload.records.len(), "Received request
 from SQS");

    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. No painel Código-fonte no console do Lambda, cole o código no arquivo lambda_function.py.



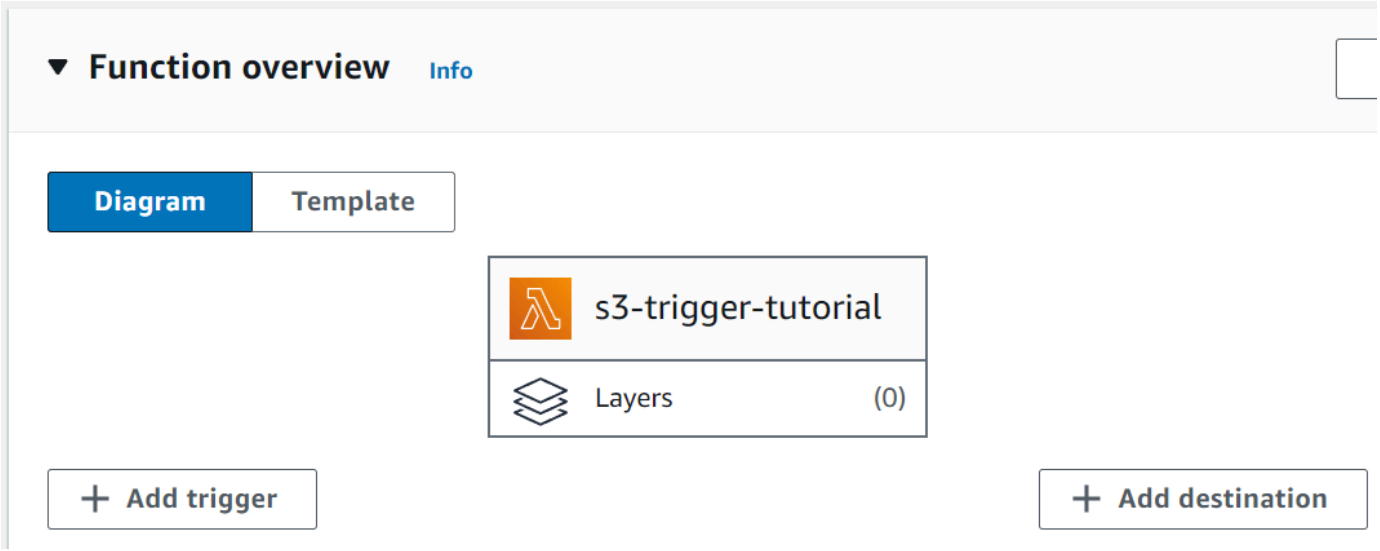
3. Escolha Implantar.

Criar o acionador do Amazon S3



Para criar o acionador do Amazon S3

1. No painel Visão geral da função, escolha Adicionar gatilho.

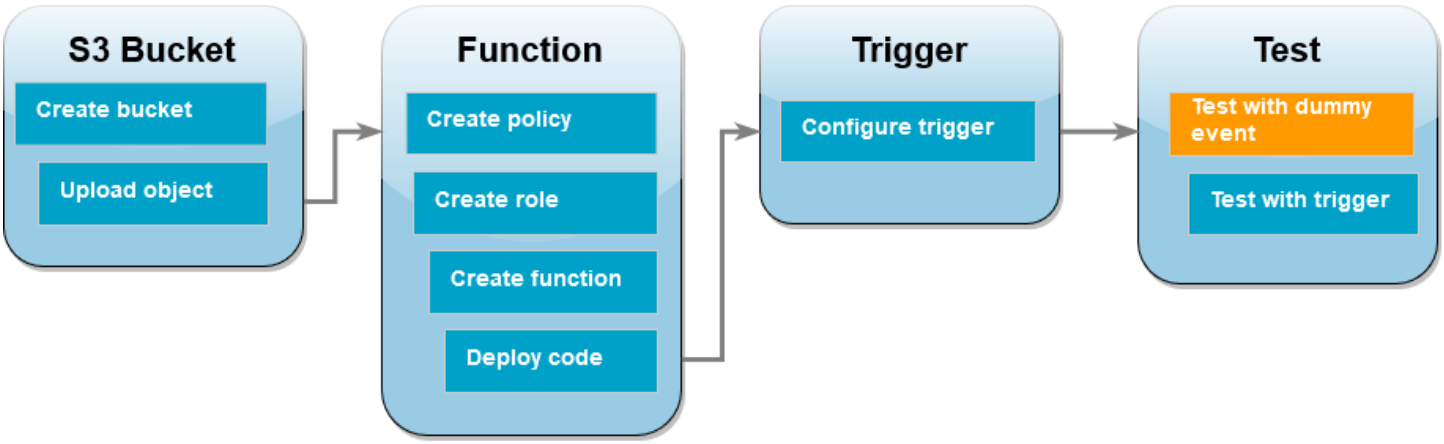


2. Selecione S3.
3. Em Bucket, selecione o bucket que você criou anteriormente no tutorial.
4. Em Tipos de eventos, garanta que a opção Todos os eventos de criação de objetos esteja selecionada.
5. Em Invocação recursiva, marque a caixa de seleção para confirmar que não é recomendável usar o mesmo bucket do Amazon S3 para entrada e saída.
6. Escolha Adicionar.

> **Note**
>
> Quando você cria um acionador do Amazon S3 para uma função do Lambda usando o console do Lambda, o Amazon S3 configura uma notificação de evento no bucket que você especificar. Antes de configurar essa notificação de evento, o Amazon S3 executa uma série de verificações para confirmar que o destino do evento existe e tem as políticas do IAM necessárias. O Amazon S3 também executa esses testes em qualquer outra notificação de evento configurada para esse bucket.
>
> Por causa dessa verificação, se o bucket tiver destinos de eventos previamente configurados para recursos que não existem mais ou para recursos que não têm as políticas de permissões necessárias, o Amazon S3 não poderá criar a nova notificação de evento. Você verá a seguinte mensagem de erro indicando que não foi possível criar seu acionador:
>
> ```
> An error occurred when creating the trigger: Unable to validate the following
>  destination configurations.
> ```

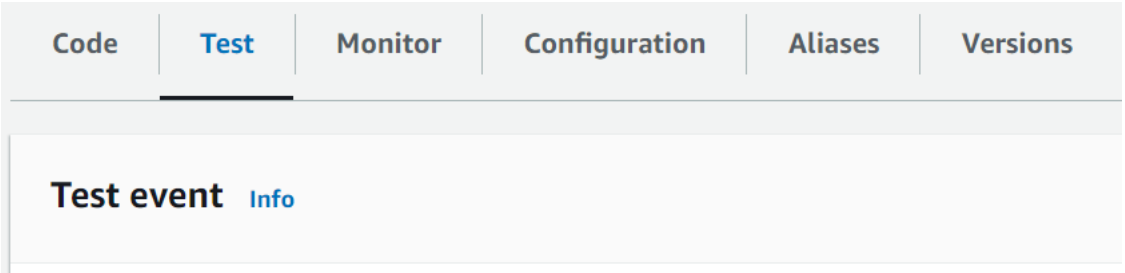
Você poderá ver esse erro se tiver configurado anteriormente um acionador para outra função do Lambda usando o mesmo bucket e, desde então, tiver excluído a função ou modificado suas políticas de permissões.

Testar sua função do Lambda com um evento fictício



Para testar a função do Lambda com um evento fictício

1. Na página de console do Lambda da sua função, escolha a guia Testar.



2. Em Nome do evento, insira `MyTestEvent`.
3. Em Evento JSON, cole o seguinte evento de teste. Não se esqueça de substituir estes valores:

   - Substitua `us-east-1` pela região em que você criou o bucket do Amazon S3.
   - Substitua ambas as instâncias de `DOC-EXAMPLE-BUCKET` pelo nome do seu próprio bucket do Amazon S3.
   - Substitua `test%2FKey` pelo nome do objeto de teste que você carregou anteriormente para o bucket (por exemplo, `HappyFace.jpg`).

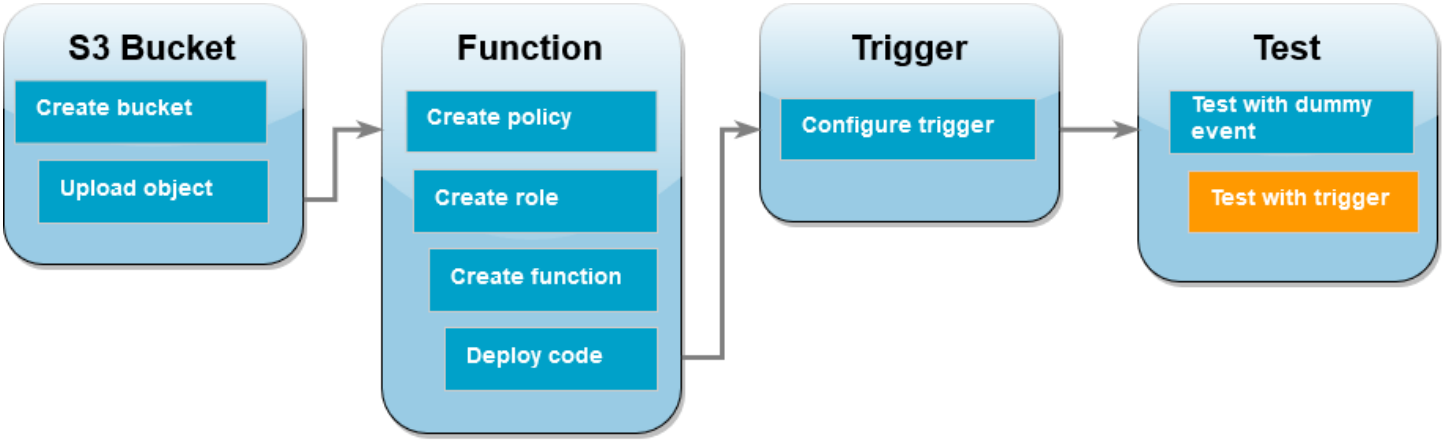
```
{
```

```
  "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. Escolha Salvar.
5. Escolha Test (Testar).
6. Se a função for executada com êxito, você verá uma saída semelhante à seguinte na guia Resultados da execução.

```
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
```

Testar a função do Lambda com o acionador do Amazon S3



Para testar a função com o gatilho configurado, carregue um objeto para o bucket do Amazon S3 usando o console. Para verificar se a função do Lambda foi executada conforme planejado, use o CloudWatch Logs para visualizar a saída da sua função.

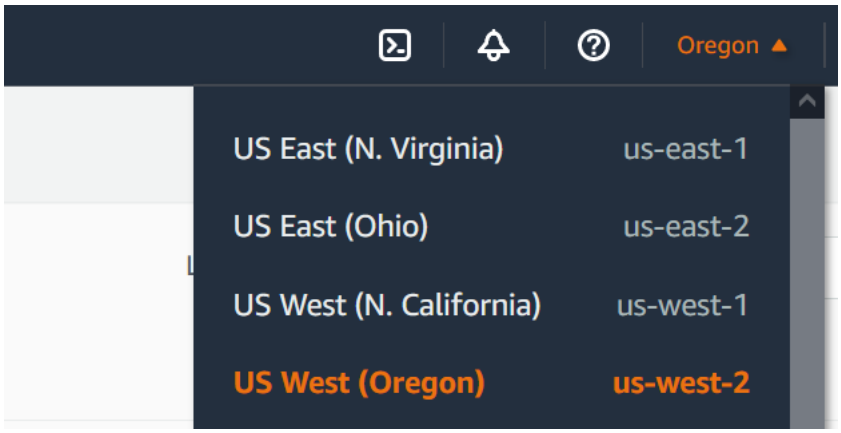
Para carregar um objeto para o bucket do Amazon S3

1. Abra a página Buckets do console do Amazon S3 e escolha o bucket que você criou anteriormente.
2. Escolha Carregar.

3. Escolha Adicionar arquivos e use o seletor de arquivos para escolher um objeto que você deseje carregar. Esse objeto pode ser qualquer arquivo que você escolher.
4. Selecione Abrir e Carregar.

Para verificar a invocação da função usando o CloudWatch Logs

1. Abra o console do CloudWatch.
2. Verifique se você está trabalhando na mesma Região da AWS em que criou a função do Lambda. Você pode alterar sua região usando a lista suspensa na parte superior da tela.



3. Escolha Logs e depois escolha Grupos de logs.
4. Escolha o nome do grupo de logs para sua função (`/aws/lambda/s3-trigger-tutorial`).
5. Em Fluxos de logs, escolha o fluxo de logs mais recente.
6. Se sua função tiver sido invocada corretamente em resposta ao gatilho do Amazon S3, você verá uma saída semelhante à seguinte. O `CONTENT TYPE` que você vê depende do tipo de arquivo que você carregou no bucket.

```
2022-05-09T23:17:28.702Z 0cae7f5a-b0af-4c73-8563-a3430333cc10 INFO CONTENT
 TYPE: image/jpeg
```

Limpe os recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Para excluir a função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Escolha Excluir.
4. Insira o nome do perfil no campo de entrada de texto e escolha Delete (Excluir).

Para excluir o bucket do S3

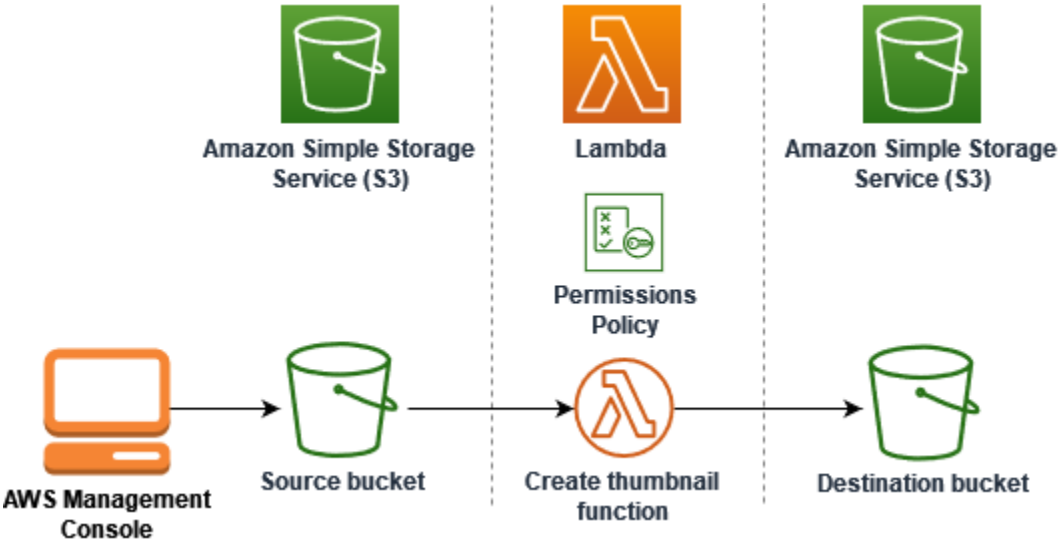
1. Abra o console Amazon S3.
2. Selecione o bucket que você criou.
3. Escolha Excluir.
4. Insira o nome do bucket no campo de entrada de texto.
5. Escolha Excluir bucket.

Próximas etapas

No Tutorial: Usar um acionador do Amazon S3 para criar imagens em miniatura, o gatilho do Amazon S3 invoca uma função para criar uma imagem em miniatura para cada arquivo de imagem que é carregado para um bucket. Este tutorial requer um nível moderado de conhecimento de domínios da AWS e do Lambda. Ele demonstra como criar recursos usando a AWS Command Line Interface (AWS CLI) e como criar um arquivo .zip de pacote de implantação de arquivamento para a sua função e suas dependências.

Tutorial: Usar um acionador do Amazon S3 para criar imagens em miniatura

Neste tutorial, você criará e configurará uma função do Lambda que redimensiona as imagens adicionadas a um bucket do Amazon Simple Storage Service (Amazon S3). Quando você adiciona um arquivo de imagem ao bucket, o Amazon S3 invoca a função do Lambda. Em seguida, a função cria uma versão em miniatura da imagem e a envia para um bucket diferente do Amazon S3.



Para concluir este tutorial, execute as seguintes tarefas:

1. Crie buckets do Amazon S3 de origem e de destino e faça upload de uma imagem de amostra.
2. Crie uma função do Lambda que redimensiona uma imagem e envia uma miniatura para um bucket do Amazon S3.
3. Configure um acionador do Lambda que invoca a função quando os objetos são carregados no bucket de origem.
4. Teste a função com um evento fictício e, em seguida, ao fazer upload de uma imagem para o bucket de origem.

Ao concluir essas etapas, você aprenderá a usar o Lambda para executar uma tarefa de processamento de arquivos em objetos adicionados a um bucket do Amazon S3. Você pode concluir este tutorial usando o AWS Command Line Interface ou a AWS CLI (AWS Management Console).

Caso esteja procurando um exemplo mais simples para aprender como configurar um acionador do Amazon S3 para o Lambda, você pode experimentar o Tutorial: usar um acionador do Amazon S3 para invocar uma função do Lambda.

Tópicos

Pré-requisitos

Cadastre-se em uma Conta da AWS

Se você ainda não tem Conta da AWS, siga as etapas a seguir para criar uma.

Para se cadastrar em uma Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

   Quando você se cadastra em uma Conta da AWS, um Usuário raiz da conta da AWS é criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como uma prática recomendada de segurança, atribua o acesso administrativo para um usuário e use somente o usuário-raiz para executar tarefas que requerem o acesso de usuário-raiz.

A AWS envia um e-mail de confirmação depois que o processo de cadastramento é concluído. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se cadastrar em uma Conta da AWS, proteja seu Usuário raiz da conta da AWS, habilite o AWS IAM Identity Center e crie um usuário administrativo para não usar o usuário raiz em tarefas cotidianas.

Proteger seu Usuário raiz da conta da AWS

1. Faça login no AWS Management Console como o proprietário da conta ao escolher a opção Usuário raiz e inserir o endereço de e-mail da Conta da AWS. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Signing in as the root user (Fazer login como usuário-raiz) no Guia do usuário do Início de Sessão da AWS.
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo MFA virtual para o usuário-raiz de sua conta da Conta da AWS (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center.
2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para obter um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso dos usuários com o Diretório do Centro de Identidade do IAM padrão no Guia do usuário do AWS IAM Identity Center.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use a URL de login que foi enviada ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

  Para obter ajuda com o login utilizando um usuário do Centro de Identidade do IAM, consulte Fazer login no portal de acesso da AWS, no Guia do usuário do Início de Sessão da AWS.

Atribuir acesso para usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

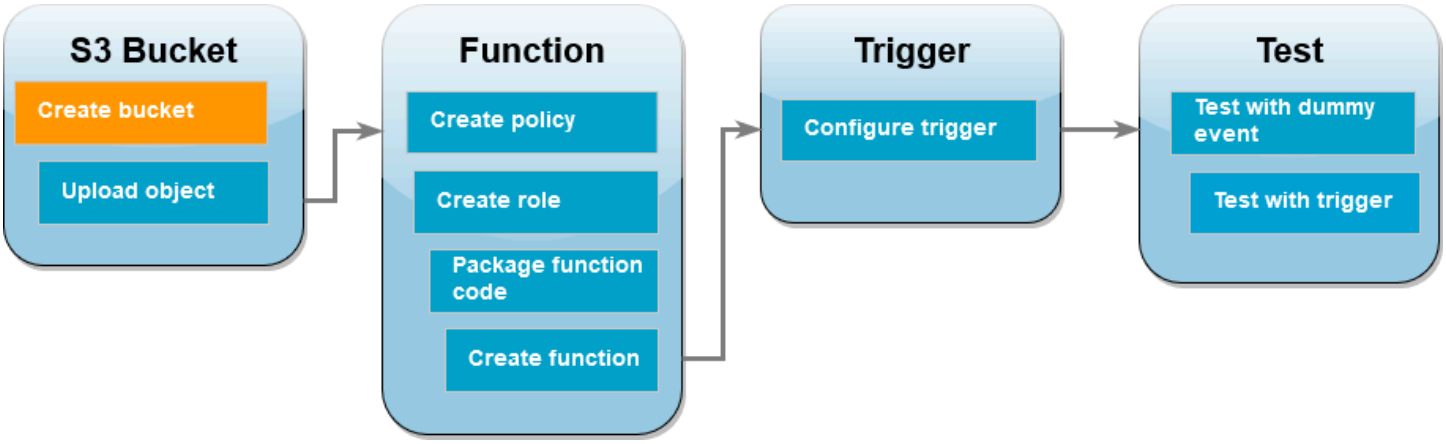
   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center.
2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center.

Se desejar usar a AWS CLI para concluir o tutorial, instale a versão mais recente da AWS Command Line Interface.

Para o código de função do Lambda, você pode usar Python ou Node.js. Instale as ferramentas compatíveis de linguagens e um gerenciador de pacotes para a linguagem que deseja usar.

Crie dois buckets do Amazon S3



Primeiro, crie dois buckets do Amazon S3. O primeiro bucket corresponde ao bucket de origem para o qual você enviará as imagens. O segundo bucket é usado pelo Lambda para salvar a miniatura redimensionada quando você invoca a função.

AWS Management Console

Criar os buckets do Amazon S3 (console)

1. Abra a página Buckets do console do Amazon S3.
2. Selecione Criar bucket.

3. Em General configuration (Configuração geral), faça o seguinte:

   a. Em Nome do bucket, insira um nome global exclusivo que atenda às regras de nomenclatura de buckets do Amazon S3. Os nomes dos buckets podem conter apenas letras minúsculas, números, pontos (.) e hifens (-).

   b. Em Região da AWS, escolha a Região da AWS mais próxima de sua localização geográfica. Posteriormente no tutorial, você deverá criar a função do Lambda na mesma Região da AWS, portanto, anote a região escolhida.

4. Deixe todas as outras opções com seus valores padrão e escolha Criar bucket.

5. Repita as etapas 1 a 4 para criar o bucket de destino. Em Nome do bucket, insira **DOC-EXAMPLE-SOURCE-BUCKET-resized**, em que **DOC-EXAMPLE-SOURCE-BUCKET** corresponde ao nome do bucket de origem que você acabou de criar.

AWS CLI

Criar os buckets do Amazon S3 (AWS CLI)

1. Execute o comando da CLI apresentado a seguir para criar o bucket de origem. O nome escolhido para o bucket deve ser globalmente exclusivo e seguir as Regras de nomeação de bucket para o Amazon S3. Os nomes podem conter somente letras minúsculas, números, pontos (.) e hifens (-). Em `region` e `LocationConstraint`, escolha a Região da AWS mais próxima de sua localização geográfica.

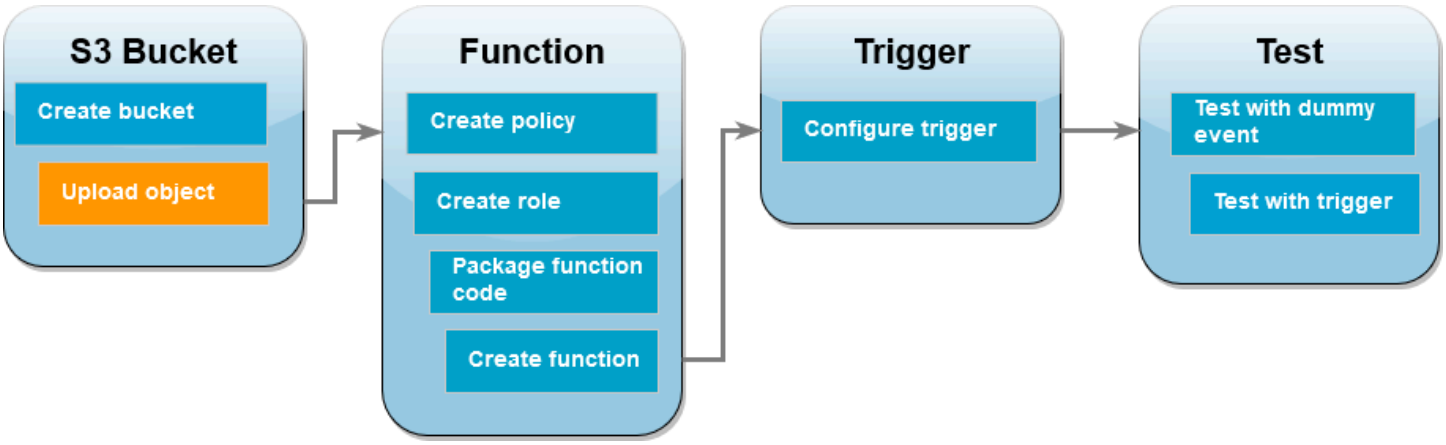
```
aws s3api create-bucket --bucket DOC-EXAMPLE-SOURCE-BUCKET --region us-west-2 \
--create-bucket-configuration LocationConstraint=us-west-2
```

   Posteriormente no tutorial, você deverá criar a função do Lambda na mesma Região da AWS em que o bucket de origem foi criado, portanto, anote a região escolhida.

2. Execute o comando apresentado a seguir para criar o bucket de destino. Para o nome do bucket, você deve usar **DOC-EXAMPLE-SOURCE-BUCKET-resized**, em que **DOC-EXAMPLE-SOURCE-BUCKET** é o nome do bucket de origem criado na etapa 1. Em `region` e `LocationConstraint`, escolha a mesma Região da AWS que você usou ao criar o bucket de origem.

```
aws s3api create-bucket --bucket DOC-EXAMPLE-SOURCE-BUCKET-resized --region us-
west-2 \
--create-bucket-configuration LocationConstraint=us-west-2
```

Faça upload de uma imagem de teste para o bucket de origem



Posteriormente no tutorial, você testará a função do Lambda ao invocá-la usando a AWS CLI ou o console do Lambda. Para confirmar que a função está funcionando corretamente, o bucket de origem precisa conter uma imagem de teste. Essa imagem pode ser qualquer arquivo JPG ou PNG que você escolher.

AWS Management Console

Fazer upload de uma imagem de teste para o bucket de origem (console)

1. Abra a página Buckets do console do Amazon S3.
2. Selecione o bucket de origem criado na etapa anterior.
3. Escolha Carregar.
4. Escolha Adicionar arquivos e use o seletor de arquivos para escolher o objeto que você deseja carregar.
5. Selecione Abrir e Carregar.

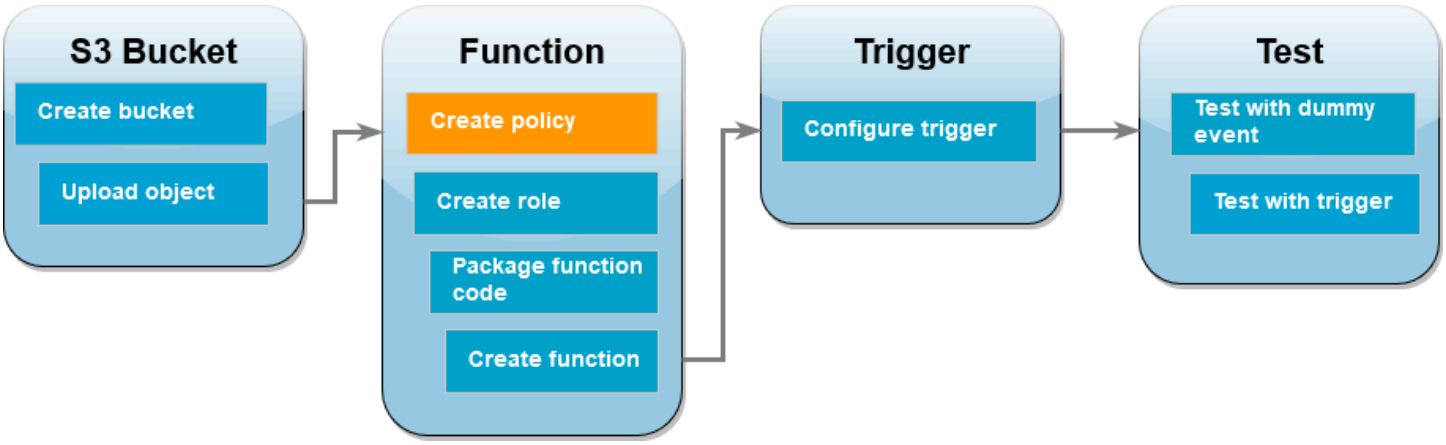
AWS CLI

Fazer upload de uma imagem de teste para o bucket de origem (AWS CLI)

- No diretório que contém a imagem que você deseja fazer upload, execute o comando da CLI apresentado a seguir. Substitua o parâmetro `--bucket` pelo nome do bucket de origem. Para os parâmetros `--key` e `--body`, use o nome de arquivo da imagem de teste.

```
aws s3api put-object --bucket DOC-EXAMPLE-SOURCE-BUCKET --key HappyFace.jpg --
body ./HappyFace.jpg
```

Criação de uma política de permissões



A primeira etapa na criação da função do Lambda é criar uma política de permissões. Essa política concede à sua função as permissões necessárias para acessar outros recursos da AWS. Para este tutorial, a política concede ao Lambda permissões de leitura e gravação para buckets do Amazon S3 e permite que ele grave no Amazon CloudWatch Logs.

AWS Management Console

Criar a política (console)

1. Abra a página Policies (Políticas) do console do AWS Identity and Access Management (IAM).
2. Escolha Criar política.
3. Escolha a guia JSON e cole a política personalizada a seguir no 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. Escolha Próximo.
5. No campo Detalhes da política, em Nome da política, insira **`LambdaS3Policy`**.
6. Escolha Criar política.

AWS CLI

Criar a política (AWS CLI)

1. Salve o JSON a seguir em um arquivo denominado `policy.json`.

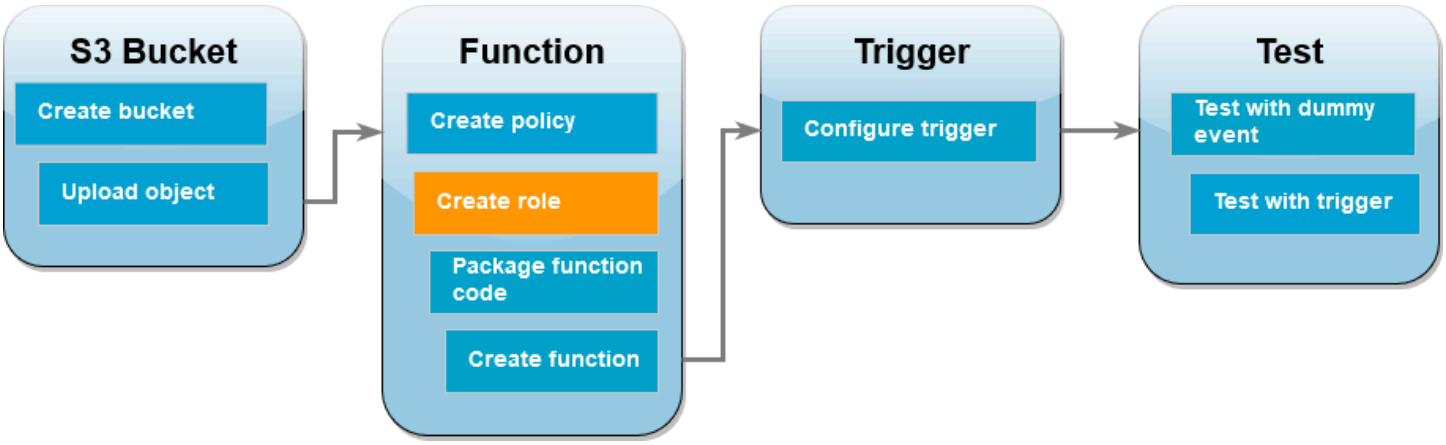
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
```

```
            "Resource": "arn:aws:s3:::*/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*/*"
        }
    ]
}
```

2. No diretório em que você salvou o documento de política JSON, execute o comando da CLI apresentado a seguir.

```
aws iam create-policy --policy-name LambdaS3Policy --policy-document file://
policy.json
```

Criar uma função de execução



Um perfil de execução é um perfil do IAM que concede a uma função do Lambda permissão para acessar os recursos e Serviços da AWS. Para conceder à sua função acesso de leitura e gravação a um bucket do Amazon S3, anexe a política de permissões criada na etapa anterior.

AWS Management Console

Criar um perfil de execução e anexar a política de permissões (console)

1. Abra a página Perfis no console do IAM.
2. Selecione Criar função.

3. Em Tipo de entidade confiável, selecione AWS service (Serviço da AWS), e em Caso de uso, selecione Lambda.
4. Escolha Próximo.
5. Adicione a política de permissões criada na etapa anterior fazendo o seguinte:
   a. Na caixa de pesquisa de política, insira **`LambdaS3Policy`**.
   b. Nos resultados da pesquisa, marque a caixa de seleção `LambdaS3Policy`.
   c. Escolha Próximo.
6. Em Detalhes do perfil, em Nome do perfil, insira **`LambdaS3Role`**.
7. Selecione Criar função.

AWS CLI

Criar um perfil de execução e anexar a política de permissões (AWS CLI)

1. Salve o JSON a seguir em um arquivo denominado `trust-policy.json`. Essa política de confiança permite que o Lambda use as permissões do perfil ao conceder à entidade principal de serviço a permissão `lambda.amazonaws.com` para chamar a ação `AssumeRole` do AWS Security Token Service (AWS STS).

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

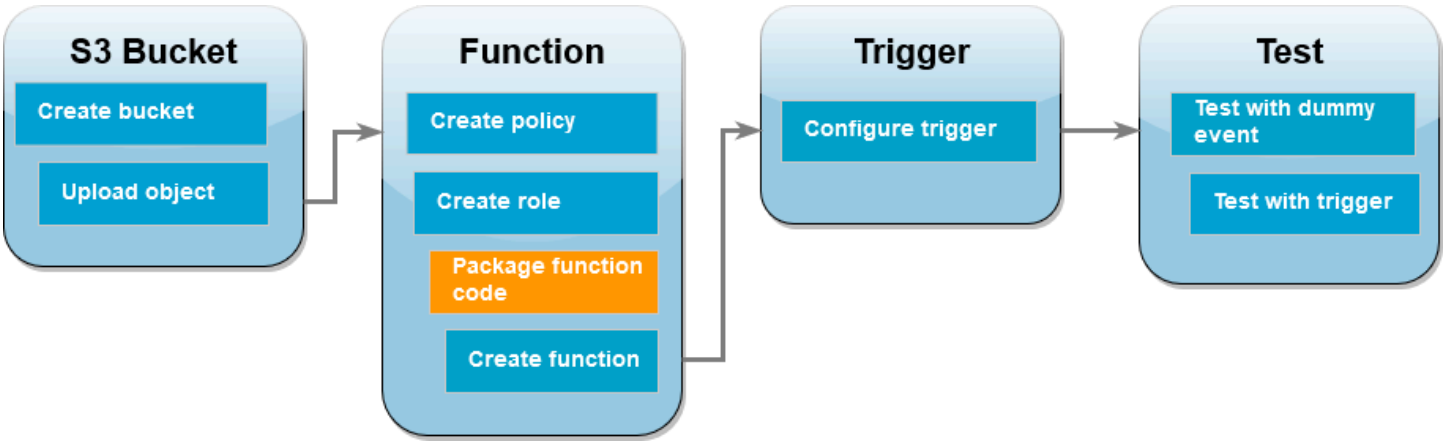
2. No diretório em que você salvou o documento de política de confiança JSON, execute o comando da CLI apresentado a seguir para criar o perfil de execução.

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

3. Para anexar a política de permissões criada na etapa anterior, execute o comando da CLI apresentado a seguir. Substitua o número da Conta da AWS no ARN da política pelo número da sua conta.

```
aws iam attach-role-policy --role-name LambdaS3Role --policy-arn
 arn:aws:iam::123456789012:policy/LambdaS3Policy
```

Crie o pacote de implantação de função



Para criar sua função, crie um pacote de implantação contendo o código de função e as dependências. Para essa função `CreateThumbnail`, o código de função usa uma biblioteca separada para o redimensionamento da imagem. Siga as instruções para a linguagem escolhida para criar um pacote de implantação contendo a biblioteca necessária.

Node.js

Criar o pacote de implantação (Node.js)

1. Crie um diretório denominado `lambda-s3` para o código de função e as dependências e navegue até ele.

```
mkdir lambda-s3
cd lambda-s3
```

2. Salve o código de função apresentado a seguir em um arquivo denominado `index.mjs`. Certifique-se de substituir `'us-west-2'` pela Região da AWS na qual você criou seus próprios buckets de origem e destino.

```
// 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. No diretório `lambda-s3`, instale a biblioteca sharp usando npm. Observe que a versão mais recente do sharp (0.33) não é compatível com o Lambda. Instale a versão 0.32.6 para concluir este tutorial.

```
npm install sharp@0.32.6
```

O comando npm `install` cria um diretório `node_modules` para seus módulos. Após esta etapa, sua estrutura de diretórios deve ser semelhante à apresentada a seguir.

```
lambda-s3
|- index.mjs
|- node_modules
|  |- base64js
|  |- bl
|  |- buffer
...
|- package-lock.json
|- package.json
```

4. Crie um pacote de implantação .zip contendo o código de função e as dependências. No MacOS e Linux, execute o comando apresentado a seguir.

```
zip -r function.zip .
```

No Windows, use o utilitário zip preferencial para criar um arquivo .zip. Certifique-se de que os arquivos `index.mjs`, `package.json` e `package-lock.json` e o diretório `node_modules` estejam todos na raiz do arquivo .zip.

Python

Criar o pacote de implantação (Python)

1. Salve o exemplo de código como um arquivo denominado `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. No mesmo diretório em que você criou o arquivo `lambda_function.py`, crie um novo diretório denominado `package` e instale a biblioteca Pillow (PIL) e o AWS SDK for Python (Boto3). Embora o runtime do Python no Lambda inclua uma versão do SDK do Boto3, recomendamos que você adicione todas as dependências da função ao pacote de implantação, mesmo que estejam inclusas no runtime. Para obter mais informações, consulte Dependências de runtime em Python.

```
mkdir package
pip install \
```

```
--platform manylinux2014_x86_64 \
--target=package \
--implementation cp \
--python-version 3.9 \
--only-binary=:all: --upgrade \
pillow boto3
```

A biblioteca Pillow contém código C/C++. Usando as opções `--platform manylinux_2014_x86_64` e `--only-binary=:all:`, o pip baixará e instalará uma versão do Pillow que contém binários pré-compilados compatíveis com o sistema operacional Amazon Linux 2. Isso garante que o pacote de implantação funcione no ambiente de execução do Lambda, independentemente do sistema operacional e da arquitetura da máquina de compilação local.

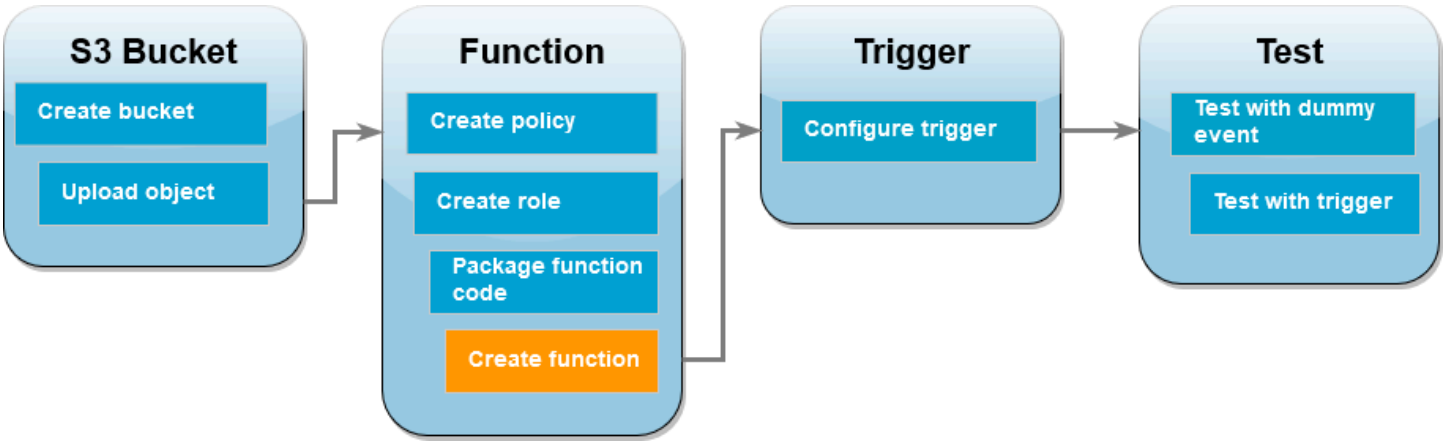
3. Crie um arquivo .zip contendo o código da aplicação e as bibliotecas Pillow e Boto3. No Linux ou no MacOS, execute os comandos a seguir na interface da linha de comando.

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

No Windows, use sua ferramenta de zip preferida para criar o arquivo `lambda_function.zip`. Certifique-se de que o arquivo `lambda_function.py` e as pastas que contêm as dependências estejam todos na raiz do arquivo .zip.

Você também pode criar seu pacote de implantação usando um ambiente virtual Python. Consulte Trabalhar com arquivos .zip para funções do Lambda em Python

Criar a função do Lambda



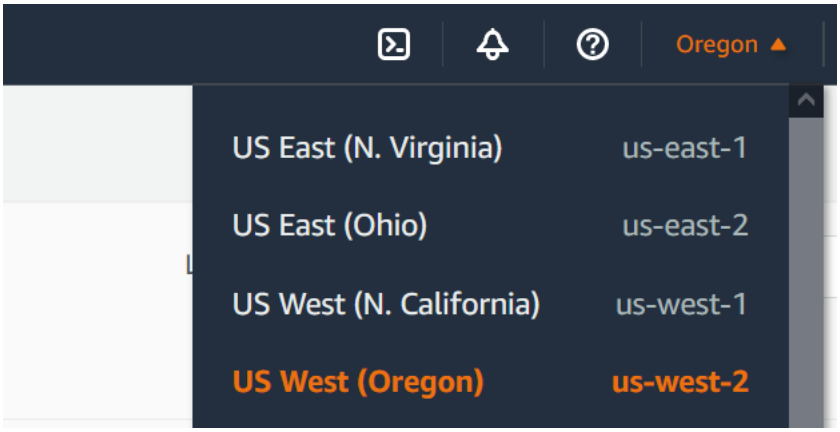
Você pode criar a função do Lambda usando a AWS CLI ou o console do Lambda. Siga as instruções da linguagem escolhida para criar a função.

AWS Management Console

Para criar a função (console)

Para criar a função do Lambda usando o console, primeiro é necessário criar uma função básica contendo algum código “Hello world”. Em seguida, substitua esse código pelo seu código de função ao fazer upload do arquivo.zip ou JAR criado na etapa anterior.

1. Abra a página Funções do console do Lambda.
2. Verifique se você está trabalhando na mesma Região da AWS em que criou o bucket do Amazon S3. É possível alterar a região usando a lista suspensa na parte superior da tela.



3. Escolha Create function (Criar função).
4. Escolha Author from scratch (Criar do zero).
5. Em Basic information (Informações básicas), faça o seguinte:

a. Em Function name (Nome da função), insira **`CreateThumbnail`**.

b. Em Runtime, escolha Node.js 18.x ou Python 3.9, de acordo com a linguagem que você escolheu para a função.

c. Em Architecture (Arquitetura), escolha x86_64.

6. Na guia Alterar função de execução padrão, faça o seguinte:

a. Expanda a guia e escolha Usar uma função existente.

b. Selecione a `LambdaS3Role` que você criou anteriormente.

7. Escolha Criar função.

Fazer upload do código da função (console)

1. No painel do Código-fonte, escolha Carregar de.
2. Escolha o arquivo .zip.
3. Escolha Carregar.
4. No seletor de arquivos, selecione o arquivo .zip e escolha Abrir.
5. Escolha Salvar.

AWS CLI

Criar a função (AWS CLI)

- Execute o comando da CLI para a linguagem escolhida. Para o parâmetro `role`, certifique-se de substituir `123456789012` por seu ID da Conta da AWS. Para o parâmetro `region`, substitua `us-west-2` pela região em que você criou os buckets do Amazon S3.

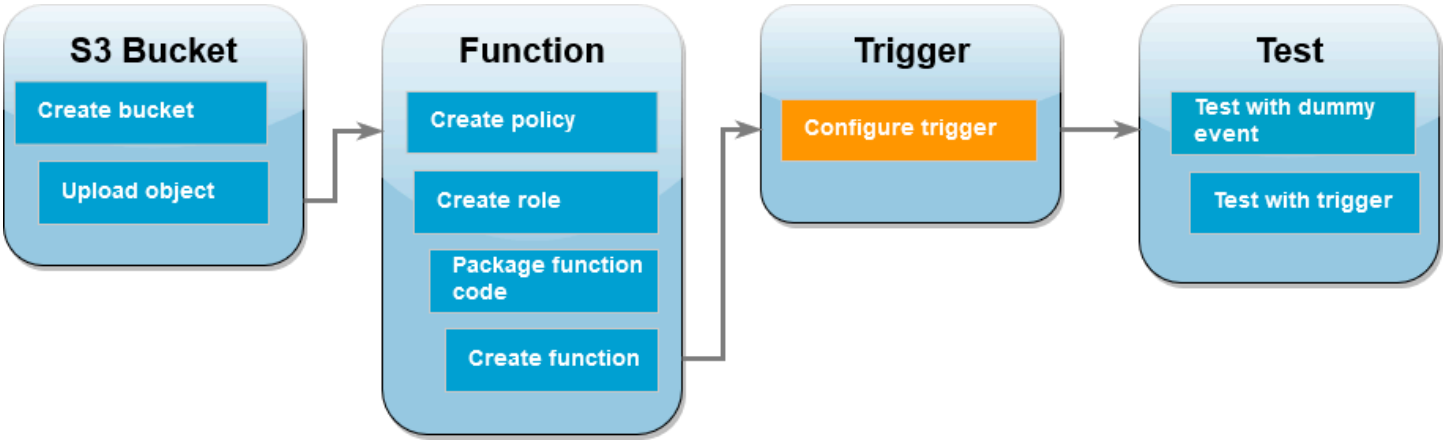
  - Para Node.js, execute o comando a seguir no diretório que contém o arquivo `function.zip`.

    ```
    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
    ```

  - Para Python, execute o comando a seguir no diretório que contém o arquivo `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
```

Configure o Amazon S3 para invocar a função



Para que a função do Lambda seja executada quando você faz upload de uma imagem no bucket de origem, é necessário configurar um acionador para a função. Você pode configurar o acionador do Amazon S3 usando o console ou a AWS CLI.

> **Important**
>
> Este procedimento configura o bucket do Amazon S3 para invocar a função toda vez que um objeto é criado no bucket. Certifique-se de configurar isso somente no bucket de origem. Se a função do Lambda criar objetos no mesmo bucket que a invoca, a função poderá ser invocada continuamente em um loop. Isso pode resultar em cobranças inesperadas para sua Conta da AWS.

AWS Management Console

Configurar o acionador do Amazon S3 (console)

1. Abra a página de Funções do console do Lambda e escolha sua função (`CreateThumbnail`).

2. Escolha Add trigger.
3. Selecione S3.
4. Em Bucket, selecione o bucket de origem.
5. Em Tipos de eventos, selecione Todos os eventos de criação de objetos.
6. Em Invocação recursiva, marque a caixa de seleção para confirmar que não é recomendável usar o mesmo bucket do Amazon S3 para entrada e saída. Saiba mais sobre padrões de invocação recursiva no Lambda lendo Recursive patterns that cause run-away Lambda functions no Serverless Land.
7. Escolha Adicionar.

   Quando você cria um acionador usando o console do Lambda, o Lambda cria automaticamente uma política baseada em recursos para conceder permissão ao serviço selecionado para invocar a função.

AWS CLI

Configurar o acionador do Amazon S3 (AWS CLI)

1. Para que o bucket de origem do Amazon S3 invoque a função ao adicionar um arquivo de imagem, primeiro é necessário configurar permissões para a função usando uma política baseada em recursos. Uma instrução de política baseada em recursos concede permissão a outros Serviços da AWS para invocar sua função. Para conceder permissão ao Amazon S3 para invocar a função, execute o comando da CLI apresentado a seguir. Certifique-se de substituir o parâmetro `source-account` por seu ID da Conta da AWS e usar o nome do bucket de origem.

   ```
   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
   ```

   A política que você define com este comando permite que o Amazon S3 invoque a função somente quando uma ação ocorre em seu bucket de origem.

> **Note**
>
> Embora os nomes de bucket do Amazon S3 sejam globalmente exclusivos, ao usar políticas baseadas em recursos, é uma prática recomendada especificar que o bucket deve pertencer à sua conta. Isso ocorre porque, se você excluir um bucket, é possível que outra Conta da AWS crie um bucket com o mesmo nome do recurso da Amazon (ARN).

2. Salve o JSON a seguir em um arquivo denominado `notification.json`. Quando aplicado ao bucket de origem, esse JSON configura o bucket para enviar uma notificação à sua função do Lambda sempre que um novo objeto é adicionado. Substitua o número da Conta da AWS e a Região da AWS no ARN da função do Lambda por seu número de conta e sua região.

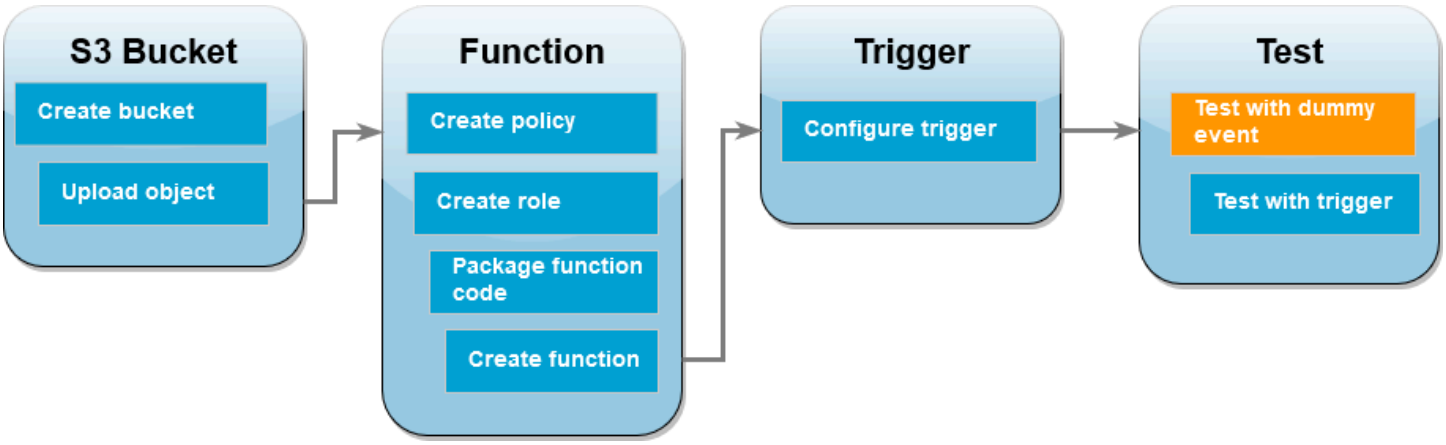
```
{
"LambdaFunctionConfigurations": [
    {
      "Id": "CreateThumbnailEventConfiguration",
      "LambdaFunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:CreateThumbnail",
      "Events": [ "s3:ObjectCreated:Put" ]
    }
  ]
}
```

3. Execute o comando da CLI apresentado a seguir para aplicar as configurações de notificação no arquivo JSON criado ao bucket de origem. Substitua `DOC-EXAMPLE-SOURCE-BUCKET` pelo nome do bucket de origem.

```
aws s3api put-bucket-notification-configuration --bucket DOC-EXAMPLE-SOURCE-
BUCKET \
--notification-configuration file://notification.json
```

Para saber mais sobre o comando `put-bucket-notification-configuration` e a opção `notification-configuration`, consulte put-bucket-notification-configuration na Referência de comandos da CLI da AWS.

Testar sua função do Lambda com um evento fictício



Antes de testar toda a configuração ao adicionar um arquivo de imagem ao bucket de origem do Amazon S3, teste se a função do Lambda está funcionando corretamente ao invocá-la com um evento fictício. Um evento no Lambda corresponde a um documento formatado em JSON que contém dados para sua função processar. Quando a função é invocada pelo Amazon S3, o evento enviado para a função contém informações como o nome do bucket, o ARN do bucket e a chave do objeto.

AWS Management Console

Testar a função do Lambda com um evento fictício (console)

1. Abra a página de Funções do console do Lambda e escolha sua função (`CreateThumbnail`).
2. Selecione a guia Testar.
3. Para criar o evento de teste, no painel Evento de teste, faça o seguinte:
   a. Em Ação de evento de teste, selecione Criar novo evento.
   b. Em Nome do evento, insira **`myTestEvent`**.
   c. Em Modelo, selecione S3 Put.
   d. Substitua os valores dos parâmetros apresentados a seguir por seus valores.
      - Em `awsRegion`, substitua `us-east-1` pela Região da AWS em que você criou os buckets do Amazon S3.
      - Em `name`, substitua `DOC-EXAMPLE-BUCKET` pelo nome do bucket de origem do Amazon S3.

- Em `key`, substitua `test%2Fkey` pelo nome do arquivo do objeto de teste que você fez upload no bucket de origem na etapa Faça upload de uma imagem de teste para o bucket de origem.

```
{
  "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. Escolha Salvar.

4. No painel Evento de teste, escolha Testar.
5. Para verificar se a função criou uma versão redimensionada da imagem e a armazenou em seu bucket de destino do Amazon S3, faça o seguinte:

   a. Abra a página Buckets do console do Amazon S3.

   b. Escolha o bucket de destino e confirme se o arquivo redimensionado está listado no painel Objetos.

AWS CLI

Testar a função do Lambda com um evento fictício (AWS CLI)

1. Salve o JSON a seguir em um arquivo denominado `dummyS3Event.json`. Substitua os valores dos seguintes parâmetros por seus valores:

   1. Em `awsRegion`, substitua `us-west-2` pela Região da AWS em que você criou os buckets do Amazon S3.
   2. Em `name`, substitua `DOC-EXAMPLE-SOURCE-BUCKET` pelo nome do bucket de origem do Amazon S3.
   3. Em `key`, substitua `HappyFace.jpg` pelo nome do arquivo do objeto de teste que você fez upload no bucket de origem na etapa Faça upload de uma imagem de teste para o bucket de origem.

```
{
  "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. No diretório em que você salvou o arquivo `dummyS3Event.json`, invoque a função ao executar o comando da CLI apresentado a seguir. Esse comando invoca a função do Lambda de forma síncrona ao especificar `RequestResponse` como o valor do parâmetro para o tipo de invocação. Para saber mais sobre invocação assíncrona e síncrona, consulte Chamada de funções do Lambda.

```
aws lambda invoke --function-name CreateThumbnail \
--invocation-type RequestResponse --cli-binary-format raw-in-base64-out \
--payload file://dummyS3Event.json outputfile.txt
```

A opção cli-binary-format será necessária se você estiver usando a versão 2 da AWS CLI. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte Opções de linhas de comando globais compatíveis com a AWS CLI.

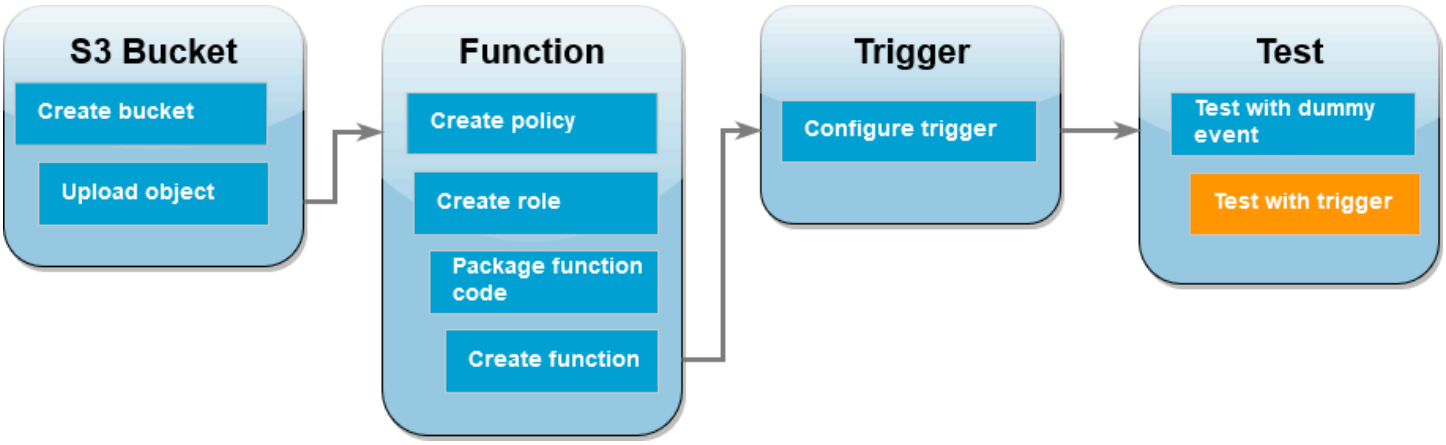
3. Verifique se a função criou uma versão em miniatura da imagem e a salvou no bucket de destino do Amazon S3. Execute o comando da CLI apresentado a seguir, substituindo `DOC-EXAMPLE-SOURCE-BUCKET-resized` pelo nome do bucket de destino.

```
aws s3api list-objects-v2 --bucket DOC-EXAMPLE-SOURCE-BUCKET-resized
```

Você deve ver saída semelhante ao seguinte: O parâmetro `Key` mostra o nome do arquivo de imagem redimensionado.

```
{
    "Contents": [
        {
            "Key": "resized-HappyFace.jpg",
            "LastModified": "2023-06-06T21:40:07+00:00",
            "ETag": "\"d8ca652ffe83ba6b721ffc20d9d7174a\"",
            "Size": 2633,
            "StorageClass": "STANDARD"
        }
    ]
}
```

Teste a função usando o acionador do Amazon S3



Agora que você confirmou que a função do Lambda está funcionando corretamente, você está com tudo pronto para testar a configuração completa ao adicionar um arquivo de imagem ao bucket de origem do Amazon S3. Quando você adiciona a imagem ao bucket de origem, a função do Lambda deve ser invocada automaticamente. A função cria uma versão redimensionada do arquivo e a armazena no bucket de destino.

AWS Management Console

Testar a função do Lambda usando o acionador do Amazon S3 (console)

1. Para fazer upload de uma imagem para o bucket do Amazon S3, faça o seguinte:
   a. Abra a página Buckets do console do Amazon S3 e escolha o bucket de origem.
   b. Escolha Carregar.
   c. Escolha Adicionar arquivos e use o seletor de arquivo para escolher o arquivo de imagem que deseja carregar. O objeto de imagem pode ser qualquer arquivo .jpg ou .png.
   d. Selecione Abrir e Carregar.
2. Verifique se o Lambda salvou uma versão redimensionada do arquivo de imagem no bucket de destino fazendo o seguinte:
   a. Retorne para a página Buckets do console do Amazon S3 e escolha o bucket de destino.
   b. No painel Objetos, deve ser possível visualizar dois arquivos de imagem redimensionados, sendo um de cada teste da função do Lambda. Para baixar a imagem redimensionada, selecione o arquivo e escolha Fazer download.

AWS CLI

Testar a função do Lambda usando o acionador do Amazon S3 (AWS CLI)

1. No diretório que contém a imagem que você deseja fazer upload, execute o comando da CLI apresentado a seguir. Substitua o parâmetro `--bucket` pelo nome do bucket de origem. Para os parâmetros `--key` e `--body`, use o nome de arquivo da imagem de teste. A imagem de teste pode ser qualquer arquivo .jpg ou .png.

```
aws s3api put-object --bucket DOC-EXAMPLE-SOURCE-BUCKET --key SmileyFace.jpg --
body ./SmileyFace.jpg
```

2. Verifique se a função criou uma versão em miniatura da imagem e a salvou no bucket de destino do Amazon S3. Execute o comando da CLI apresentado a seguir, substituindo `DOC-EXAMPLE-SOURCE-BUCKET-resized` pelo nome do bucket de destino.

```
aws s3api list-objects-v2 --bucket DOC-EXAMPLE-SOURCE-BUCKET-resized
```

Se a função for executada com êxito, você visualizará uma saída semelhante à apresentada a seguir. Agora, o bucket de destino deve conter dois arquivos redimensionados.


```
{
    "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"
        }
    ]
}
```


Limpe os recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Para excluir a política que você criou

1. Abra a página Policies (Políticas) do console do IAM.

2. Selecione a política que você criou (AWSLambdaS3Policy).
3. Escolha Policy actions (Ações de política) e Delete (Excluir).
4. Escolha Excluir.

Para excluir a função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Escolha Excluir.
4. Insira o nome do perfil no campo de entrada de texto e escolha Delete (Excluir).

Para excluir o bucket do S3

1. Abra o console Amazon S3.
2. Selecione o bucket que você criou.
3. Escolha Excluir.
4. Insira o nome do bucket no campo de entrada de texto.
5. Escolha Excluir bucket.

Usar o AWS Lambda com operações em lote do Amazon S3

É possível usar operações em lote do Amazon S3 para invocar uma função do Lambda em um grande conjunto de objetos do Amazon S3. O Amazon S3 rastreia o andamento das operações em lote, envia notificações e armazena um relatório de conclusão que mostra o status de cada ação.

Para executar uma operação em lote, crie um Amazon S3trabalho de operações em lote. Ao criar o trabalho, forneça um manifesto (a lista de objetos) e configure a ação a ser executada nesses objetos.

Quando o trabalho em lote é iniciado, o Amazon S3 invoca a função do Lambda de forma síncrona para cada objeto no manifesto. O parâmetro do evento inclui os nomes do bucket e do objeto.

O exemplo a seguir mostra o evento que o Amazon S3 envia à função do Lambda para um objeto que é chamado customerImage1.jpg no bucket DOC-EXAMPLE-BUCKET.

Example Eventos de solicitação em lote do 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"
        }
    ]
}
```

A função do Lambda deve retornar um objeto JSON com os campos, conforme mostrado no exemplo a seguir. É possível copiar o `invocationId` e `taskId` do parâmetro do evento. Você pode retornar uma string no`resultString`. O Amazon S3 salva o`resultString`no relatório de conclusão.

Example Resposta em lote do Amazon S3

```
{
  "invocationSchemaVersion": "1.0",
  "treatMissingKeysAs" : "PermanentFailure",
  "invocationId" : "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
  "results": [
    {
      "taskId": "dGFza2lkZ29lc2hlcmUK",
      "resultCode": "Succeeded",
      "resultString": "[\"Alice\", \"Bob\"]"
    }
  ]
}
```

Invocar funções do Lambda de operações em lote do Amazon S3

É possível invocar a função do Lambda com um ARN de função qualificado ou não qualificado. Se você quiser usar a mesma versão de função para todo o trabalho em lote, configure uma versão de função específica no parâmetro `FunctionARN` ao criar o trabalho. Se você configurar um alias ou o qualificador $LATEST, o trabalho em lote começará imediatamente a chamar a nova versão da função se o alias ou $LATEST for atualizado durante a execução do trabalho.

Observe que não é possível reutilizar uma função baseada em evento do Amazon S3 para operações em lote. Isso ocorre, pois a operação em lote do Amazon S3 transmite um parâmetro de evento diferente para a função do Lambda e espera uma mensagem de retorno com uma estrutura JSON específica.

Na política baseada em recursos criada para o trabalho em lote do Amazon S3, verifique se você definiu a permissão para o trabalho invocar a função do Lambda.

Na função de execução da função, defina uma política de confiança para que o Amazon S3 assuma a função quando ele executar a função.

Se a função usar o AWS SDK para gerenciar recursos do Amazon S3, será necessário adicionar permissões do Amazon S3 na função de execução.

Quando o trabalho é executado, o Amazon S3 inicia várias instâncias de função para processar os objetos do Amazon S3 em paralelo, até oLimite de simultaneidadeda função do. O Amazon S3 limita a aceleração de instâncias para evitar custo em excesso para trabalhos menores.

Se a função do Lambda retornar um código de resposta `TemporaryFailure`, o Amazon S3 tentará novamente realizar a operação.

Para obter mais informações sobre como gerenciar operações em lote do Amazon S3, consulte Executar operações em lote no Guia do desenvolvedor do Amazon S3.

Para obter um exemplo de como usar uma função do Lambda nas operações em lote do Amazon S3, consulte Invocar uma função do Lambda de operações em lote do Amazon S3 no Guia do desenvolvedor do Amazon S3.

Transformar objetos S3 com o S3 Object Lambda

Com o S3 Object Lambda, você pode adicionar seu próprio código às solicitações GET, HEAD e LIST do Amazon S3 para modificar e processar dados antes de serem retornados para uma aplicação. Você pode usar o código personalizado para modificar os dados retornados por solicitações GET, HEAD e LIST padrão do S3 para filtrar linhas, redimensionar imagens dinamicamente, editar dados confidenciais e muito mais. Desenvolvido pelas funções do AWS Lambda, seu código é executado em uma infraestrutura totalmente gerenciada pela AWS, eliminando a necessidade de criar e armazenar cópias derivadas de seus dados ou de executar proxies, tudo sem a necessidade de alterações nas aplicações.

Para obter mais informações, consulte Transformar objetos com o S3 Object Lambda.

Tutoriais

- Transformação de dados para sua aplicação com o Amazon S3 Object Lambda
- Detecção e edição de dados PII com o Amazon S3 Object Lambda e o Amazon Comprehend
- Como usar o Amazon S3 Object Lambda para colocar marca d'água em imagens de maneira dinâmica à medida que são recuperadas

Usar o AWS Lambda com o Secrets Manager

Sua função do AWS Lambda pode interagir com o AWS Secrets Manager usando a API do Secrets Manager ou qualquer um dos kits de desenvolvimento de software (SDKs) da AWS. Também é possível usar a extensão do Lambda para parâmetros e segredos da AWS com a finalidade de recuperar e armazenar em cache os segredos do AWS Secrets Manager em funções do Lambda sem usar um SDK. Consulte Use segredos do AWS Secrets Manager em funções do AWS Lambda para obter mais informações.

Usar o AWS Lambda com o Amazon SES

Ao usar o Amazon SES para receber mensagens, é possível configurá-lo para chamar sua função do Lambda quando as mensagens chegarem. O serviço pode invocar sua função do Lambda passando o evento de e-mail recebido que, na realidade, é uma mensagem do Amazon SES em um evento do Amazon SNS, como um parâmetro.

Example Evento de mensagem do 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"
    }
  ]
}
```

Para obter informações, consulte Ação do Lambda no Guia do desenvolvedor do Amazon SES.

Invocar funções do Lambda com notificações do Amazon SNS

Você pode usar uma função do Lambda para processar notificações do Amazon Simple Notification Service (Amazon SNS). O Amazon SNS oferece suporte às funções do Lambda como destino para mensagens enviadas para um tópico. É possível inscrever a sua função em tópicos na mesma conta ou em outras contas da AWS. Para obter uma descrição detalhada, consulte the section called "Tutorial".

O Lambda oferece suporte a acionadores do SNS somente para tópicos do SNS padrão. Não há suporte a tópicos FIFO.

Para invocação assíncrona, o Lambda enfileira a mensagem e processa novas tentativas. Se o Amazon SNS não puder acessar o Lambda ou se a mensagem for rejeitada, o Amazon SNS tentará novamente em intervalos crescentes ao longo de várias horas. Para obter detalhes, consulte Confiabilidade nas Perguntas frequentes do Amazon SNS.

> **Warning**
>
> Os mapeamentos da origem do evento do Lambda processam cada evento ao menos uma vez, podendo haver o processamento duplicado de registros. Para evitar possíveis problemas relacionados a eventos duplicados, é altamente recomendável tornar o código da função idempotente. Para saber mais, consulte Como tornar minha função do Lambda idempotente no Centro de Conhecimentos da AWS.

Tópicos

Adicionar um acionador de tópico do Amazon SNS para uma função do Lambda usando o console

Para adicionar um tópico do SNS como acionador de uma função do Lambda, a maneira mais fácil é usar o console do Lambda. Quando você adiciona o acionador por meio do console, o Lambda

configura automaticamente as permissões e assinaturas necessárias para começar a receber eventos do tópico do SNS.

Para adicionar um tópico do SNS como acionador para uma função do Lambda (console)

1. Abra a página Funções do console do Lambda.
2. Escolha o nome da função à qual você deseja adicionar o acionador.
3. Escolha Configuração e, em seguida, escolha Acionadores.
4. Escolha Add trigger.
5. Em Configuração do acionador, no menu suspenso, escolha SNS.
6. Em Tópico do SNS, escolha o tópico do SNS para assinar.

Adicionar manualmente um acionador de tópico do Amazon SNS para uma função do Lambda

Para configurar manualmente um acionador do SNS para uma função do Lambda, é necessário concluir as seguintes etapas:

- Definir uma política baseada em recurso para a sua função para permitir que o SNS a invoque.
- Inscrever a função do Lambda no tópico do Amazon SNS.

> **Note**
>
> Se seu tópico do SNS e sua função do Lambda estiverem em contas da AWS diferentes, também será necessário conceder permissões extras para permitir assinaturas entre contas do tópico do SNS. Para obter mais informações, consulte Concede permissões entre contas para assinatura do Amazon SNS.

Você pode usar o AWS Command Line Interface (AWS CLI) para concluir essas duas etapas. Primeiro, para definir uma política baseada em recursos para uma função do Lambda que permita invocações do SNS, use o comando da AWS CLI a seguir. Certifique-se de substituir o valor de `--function-name` pelo nome da função do Lambda e o valor de `--source-arn` pelo ARN do tópico do SNS.

```
aws lambda add-permission --function-name example-function \
```

```
    --source-arn arn:aws:sns:us-east-1:123456789012:sns-topic-for-lambda \
    --statement-id function-with-sns --action "lambda:InvokeFunction" \
    --principal sns.amazonaws.com
```

Para inscrever sua função no tópico do SNS, use o comando da AWS CLI a seguir. Substitua o valor de `--topic-arn` pelo ARN do tópico do SNS e o valor de `--notification-endpoint` pelo ARN da função do Lambda.

```
aws sns subscribe --protocol lambda \
    --region us-east-1 \
    --topic-arn arn:aws:sns:us-east-1:123456789012:sns-topic-for-lambda \
    --notification-endpoint arn:aws:lambda:us-east-1:123456789012:function:example-
function
```

Exemplo de formato de evento do SNS

O Amazon SNS invoca a função de forma assíncrona com um evento que contém uma mensagem e metadados.

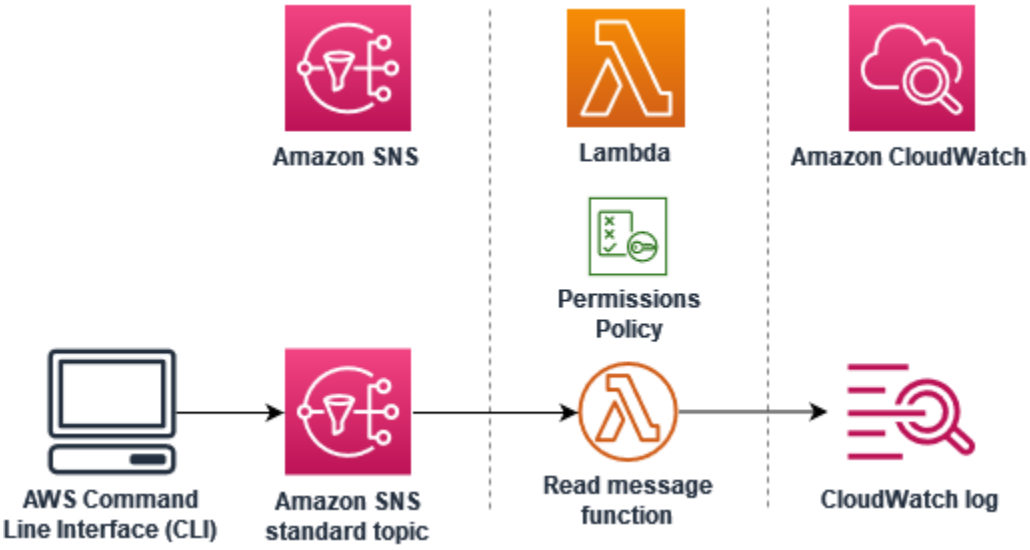
Example Evento da mensagem do Amazon SNS

```
{
  "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"
      }
    }
  ]
}
```

Como usar oAWS LambdaCom o Amazon Simple Notification Service

Neste tutorial, você usa uma função do Lambda em uma Conta da AWS para assinar um tópico do Amazon Simple Notification Service (Amazon SNS) em uma Conta da AWS separada. Quando você publica mensagens em seu tópico do Amazon SNS, sua função do Lambda lê o conteúdo da mensagem e a envia para o Amazon CloudWatch Logs. Para concluir este tutorial, use o AWS Command Line Interface (AWS CLI).



Para concluir este tutorial, execute as seguintes etapas:

- Na conta A, crie um tópico do Amazon SNS.
- Na conta B, crie uma função do Lambda que leia as mensagens do tópico.

- Na conta B, crie uma assinatura para o tópico.
- Publique mensagens para o tópico do Amazon SNS na conta A e confirme se a função do Lambda na conta B os envia para o CloudWatch Logs.

Ao concluir essas etapas, você aprenderá a configurar um tópico do Amazon SNS para invocar uma função do Lambda. Você também aprenderá como criar uma política do AWS Identity and Access Management (IAM) que dá permissão para um recurso em outra Conta da AWS para invocar o Lambda.

No tutorial, você usa duas Contas da AWS separadas. Os comandos da AWS CLI ilustram isso usando dois perfis nomeados `accountA` e `accountB`, cada um configurado para uso com uma Conta da AWS.diferente. Para saber como configurar a AWS CLI para usar perfis diferentes, consulte Configurações de arquivos de configuração e credenciais no AWS Command Line InterfaceGuia do usuário para a versão 2. Certifique-se de configurar o mesmo padrão de Região da AWS para ambos os perfis.

Se os perfis da AWS CLI criados para as duas Contas da AWS usarem nomes diferentes, ou se você usa o perfil padrão e um perfil nomeado, modifique os comandos da AWS CLI nas seguintes etapas, conforme necessário.

Pré-requisitos

Cadastre-se em uma Conta da AWS

Se você ainda não tem Conta da AWS, siga as etapas a seguir para criar uma.

Para cadastrar-se em uma Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

   Quando você se cadastra em uma Conta da AWS, um Usuário raiz da conta da AWS é criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como prática recomendada de segurança, atribua o acesso administrativo a um usuário e use somente o usuário-raiz para executar tarefas que exigem acesso de usuário-raiz.

AWS envia um e-mail de confirmação depois que o processo de cadastramento é concluído. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se cadastrar em uma Conta da AWS, proteja seu Usuário raiz da conta da AWS, habilite o AWS IAM Identity Center e crie um usuário administrativo para não usar o usuário raiz em tarefas cotidianas.

Proteja seu Usuário raiz da conta da AWS

1. Faça login no AWS Management Console como o proprietário da conta ao selecionar a opção Root user (Usuário raiz) e inserir o endereço de e-mail da Conta da AWS. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário raiz, consulte Signing in as the root user (Fazer login como usuário raiz) no Guia do usuárioInício de Sessão da AWS.
2. Ative a autenticação multifator (MFA) para seu usuário raiz.

   Para obter instruções, consulte Habilitar um dispositivo MFA virtual para o usuário raiz (console)Conta da AWS no Guia do Usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center.
2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para obter um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso dos usuários com o Diretório do Centro de Identidade do IAM padrão no Guia do usuário do AWS IAM Identity Center.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário IAM Identity Center, use a URL de login enviada ao seu endereço de e-mail quando você criou o usuário IAM Identity Center user.

Para obter ajuda com o login utilizando um usuário do IAM Identity Center, consulte Fazendo login no portal de acesso da AWS, no Guia do UsuárioInício de Sessão da AWS.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center.

2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center.

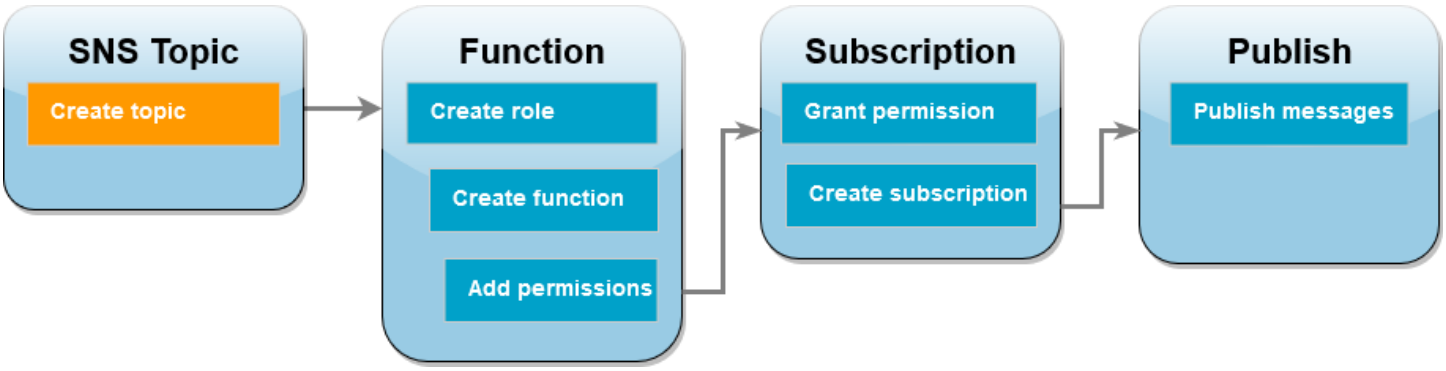
Instalar a AWS Command Line Interface

Se você ainda não instalou a AWS Command Line Interface, siga as etapas em Instalar ou atualizar a versão mais recente da AWS CLI para instalá-la.

O tutorial requer um terminal de linha de comando ou um shell para executar os comandos. No Linux e no macOS, use o gerenciador de pacotes e de shell de sua preferência.

> **Note**
>
> No Windows, alguns comandos da CLI do Bash que você costuma usar com o Lambda (como `zip`) não são compatíveis com os terminais integrados do sistema operacional. Para obter uma versão do Ubuntu com o Bash integrada no Windows, instale o Subsistema do Windows para Linux.

Criar um tópico do Amazon SNS (conta A)



Para criar um tópico do

- Na conta A, crie um tópico padrão do Amazon SNS usando o comando da AWS CLI a seguir.

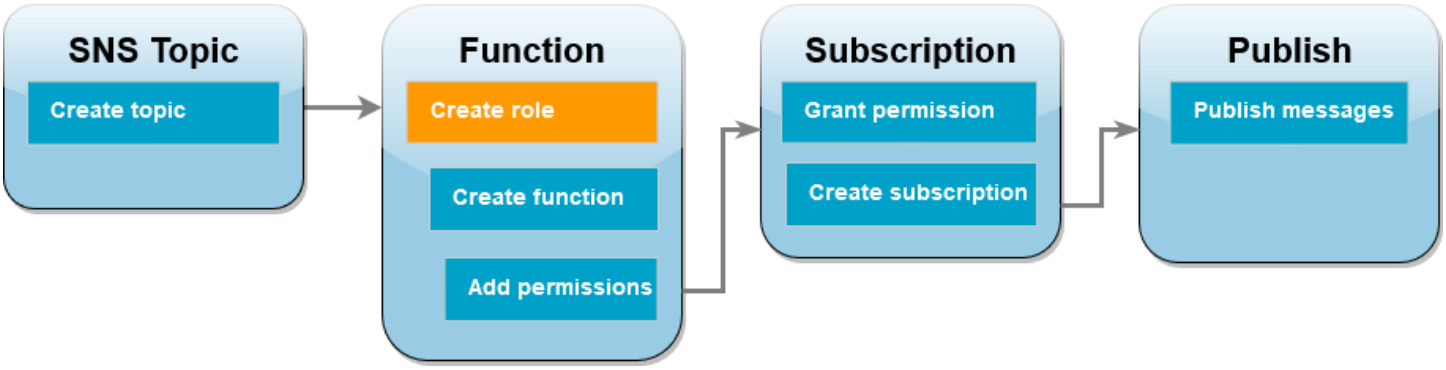
  ```
  aws sns create-topic --name sns-topic-for-lambda --profile accountA
  ```

  Você deve ver saída semelhante ao seguinte:

  ```
  {
      "TopicArn": "arn:aws:sns:us-west-2:123456789012:sns-topic-for-lambda"
  }
  ```

  Anote o nome do recurso da Amazon (ARN) do seu tópico. Você precisará dele posteriormente no tutorial ao adicionar permissões à sua função do Lambda para assinar o tópico.

Criar uma função de execução da função (conta B)



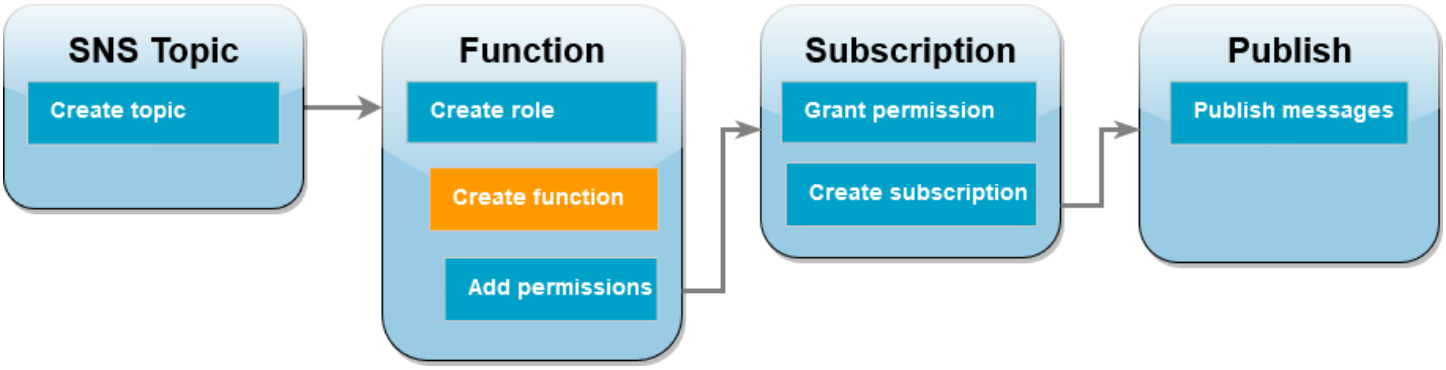
Uma função de execução é um perfil do IAM que concede a uma função do Lambda permissão para acessar serviços e recursos da AWS. Antes de criar sua função na conta B, você cria um perfil

que dá à função permissões básicas para gravar logs no CloudWatch Logs. Adicionaremos as permissões para ler seu tópico do Amazon SNS em uma etapa posterior.

Para criar uma função de execução

1. Na conta B, abra a página de perfis no console do IAM.
2. Selecione Criar função.
3. Em Tipo de entidade confiável, escolha Serviços da AWS.
4. Em Caso de uso, escolha Lambda.
5. Escolha Próximo.
6. Adicione uma política de permissões básica para o perfil ao fazer o seguinte:
   a. Na caixa de pesquisa Políticas de permissões, insira **`AWSLambdaBasicExecutionRole`**.
   b. Escolha Próximo.
7. Finalize a criação do perfil fazendo o seguinte:
   a. Em Detalhes do perfil, insira **`lambda-sns-role`** para Nome do perfil.
   b. Selecione Criar função.

Criar uma função Lambda (conta B)



Criar uma função do Lambda que processe suas mensagens do Amazon SNS. O código da função registra o conteúdo da mensagem de cada registro no Amazon CloudWatch Logs.

Este tutorial usa o runtime do Node.js 18.x, mas também fornecemos exemplos de arquivos em outras linguagens de runtime. Você pode selecionar a guia na caixa a seguir para ver o código do runtime do seu interesse. O código JavaScript que você usará nesta etapa é o primeiro exemplo mostrado na guia JavaScript.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório de Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consumir um evento do SNS com o Lambda usando TypeScript.

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

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
```

```
    throw err;
  }
}
```

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consuma um evento do SNS com o Lambda usando 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
}
```

Para criar a função

1. Crie um diretório para o projeto e depois mude para esse diretório.

```
mkdir sns-tutorial
cd sns-tutorial
```

2. Copie o código JavaScript de amostra em um novo arquivo denominado `index.js`.
3. Crie um pacote de implantação usando o comando `zip` a seguir.

```
zip function.zip index.js
```

4. Execute o comando da AWS CLI a seguir para criar sua função do Lambda na conta 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
```

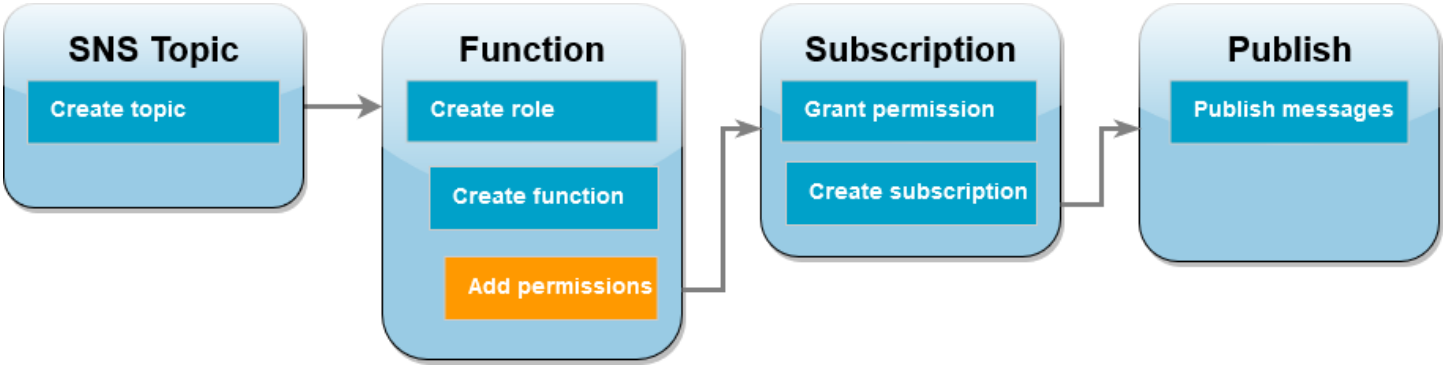
Você deve ver saída semelhante ao seguinte:

```
{
    "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. O nome do recurso da Amazon (ARN) da sua função. Você precisará dele posteriormente no tutorial ao adicionar permissões para permitir que o Amazon SNS invoque sua função.

Adicionar permissões à função (conta B)



Para que o Amazon SNS invoque a sua função, você precisa conceder permissão em uma declaração de uma política baseada em recursos. Você adiciona essa declaração usando o comando `add-permission` da AWS CLI.

Para conceder permissão ao Amazon SNS para invocar sua função

- Na conta B, execute o seguinte comando da AWS CLI usando o ARN para o tópico do Amazon SNS que você gravou anteriormente.

```
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
```

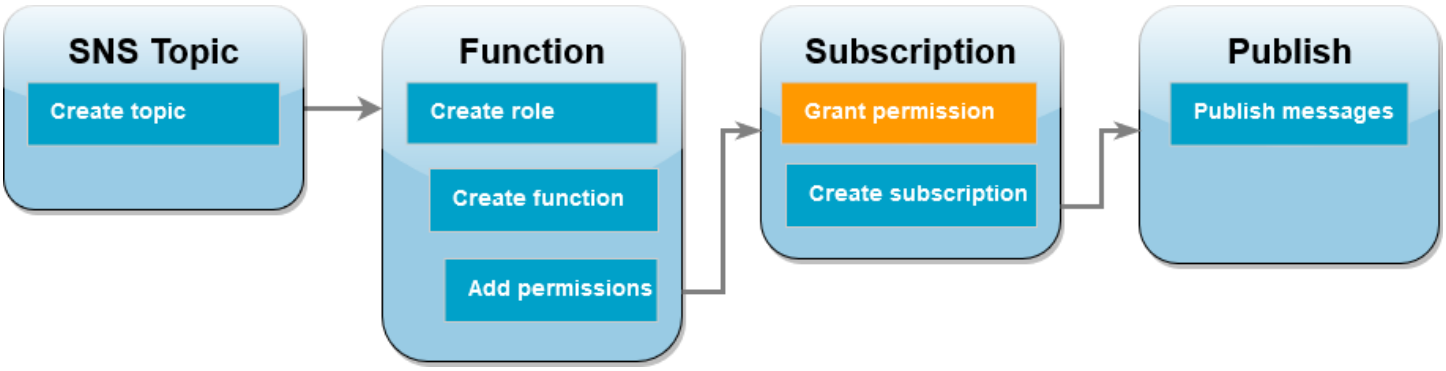
Você deve ver saída semelhante ao seguinte:

```
{
    "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 a conta com o tópico do Amazon SNS estiver hospedada em uma Região da AWS de aceitação, será necessário especificar a região na entidade principal. Por exemplo, se você estiver trabalhando com um tópico do Amazon SNS na região Ásia-Pacífico (Hong Kong), será necessário especificar `sns.ap-east-1.amazonaws.com` em vez de `sns.amazonaws.com` para a entidade principal.

Conceder permissão entre contas para assinatura do Amazon SNS (conta A)



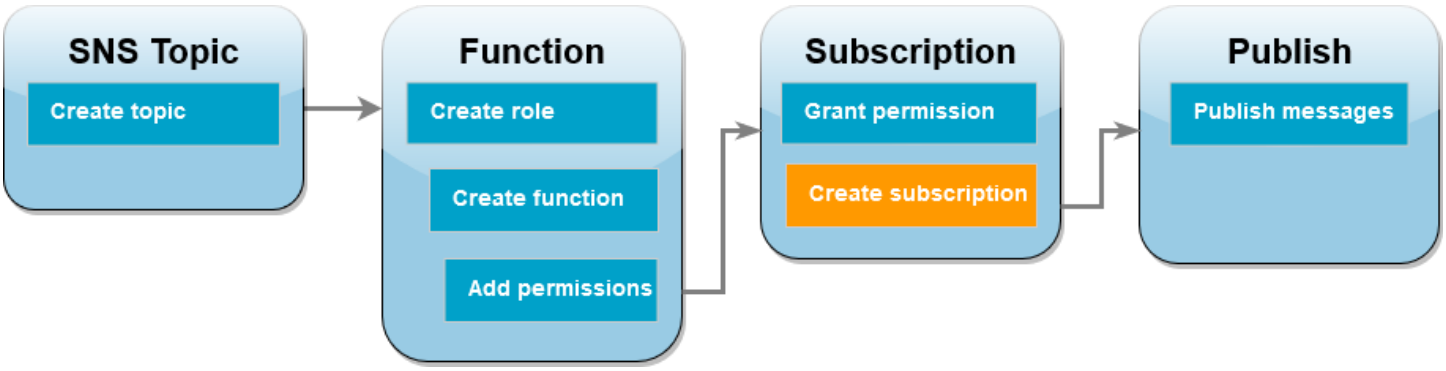
Para que a sua função do Lambda na conta B assine o tópico do Amazon SNS criado na conta A, você precisa conceder permissão para que a conta B assine seu tópico. Você concede essa permissão usando o comando `add-permission` da AWS CLI.

Para conceder permissão para a conta B assinar o tópico

- Na conta A, execute o comando da AWS CLI a seguir. Use o ARN para o tópico do Amazon SNS que você gravou anteriormente.

```
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
```

Criar uma assinatura (conta B)



Na conta B, agora você assina a sua função do Lambda no tópico do Amazon SNS criado no início do tutorial na conta A. Quando uma mensagem é enviada para esse tópico (`sns-topic-for-lambda`), o Amazon SNS invoca sua função do Lambda `Function-With-SNS` na conta B.

Criar uma assinatura

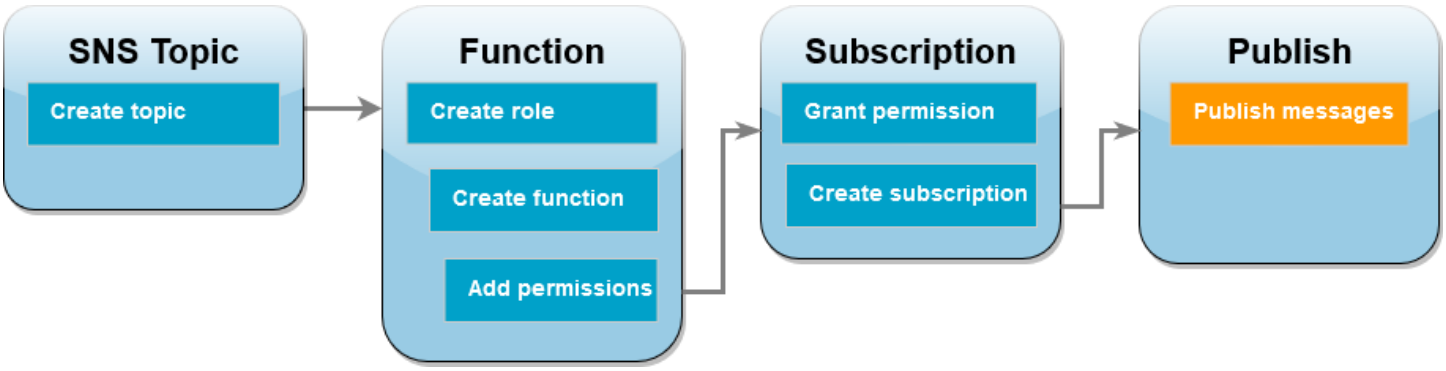
- Na conta B, execute o comando da AWS CLI a seguir. Use sua região padrão na qual você criou seu tópico e os ARNs para seu tópico e função do Lambda.

```
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
```

Você deve ver saída semelhante ao seguinte:

```
{
    "SubscriptionArn": "arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-
lambda:5d906xxxx-7c8x-45dx-a9dx-0484e31c98xx"
}
```

Publicar mensagens no tópico (conta A e conta B)



Agora que a sua função do Lambda na conta B está assinada em seu tópico do Amazon SNS na conta A, é hora de testar sua configuração publicando mensagens em seu tópico. Para confirmar se o Amazon SNS invocou sua função do Lambda, use o CloudWatch Logs para visualizar a saída da função.

Para publicar uma mensagem em seu tópico e ver a saída da sua função

1. Insira `Hello World` em um arquivo de texto e salve-o como `message.txt`.
2. No mesmo diretório em que você salvou seu arquivo de texto, execute o comando da AWS CLI a seguir na conta A. Use o ARN para seu próprio tópico.

   ```
   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
   ```

   Isso retornará um ID de mensagem com um identificador exclusivo, indicando que a mensagem foi aceita pelo Amazon SNS. Depois, o Amazon SNS tenta enviar a mensagem aos assinantes do tópico. Para confirmar se o Amazon SNS invocou sua função do Lambda, use o CloudWatch Logs para visualizar a saída da função:
3. Na conta B, abra a página Grupos de logs do console do Amazon CloudWatch.
4. Escolha o nome do grupo de logs para sua função (`/aws/lambda/Function-With-SNS`).
5. Escolha o stream de logs mais recente.
6. Se sua função foi invocada corretamente, você verá uma saída semelhante à seguinte, mostrando o conteúdo da mensagem que você publicou no seu tópico.

   ```
   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
```

Limpe os recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua Conta da AWS.

Em Account A (Conta A), limpe o tópico do Amazon SNS.

Para excluir o tópico do Amazon SNS

1. Abra a página Topics (Tópicos) no console do Amazon SNS.
2. Selecione o tópico que você criou.
3. Escolha Excluir.
4. Digite **`delete me`** no campo de entrada de texto.
5. Escolha Excluir.

Em Account B (Conta B), limpe sua função de execução, a função Lambda e a assinatura do Amazon SNS.

Para excluir a função de execução

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Escolha Excluir.
4. Insira o nome do perfil no campo de entrada de texto e escolha Delete (Excluir).

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Para excluir a inscrição do Amazon SNS

1. Abrir oPágina AssinaturasNo console do Amazon SNS.
2. Selecione a assinatura que você criou.
3. Escolha Delete (Excluir), Yes, Delete (Sim, excluir).

Melhores práticas para trabalhar com funções do AWS Lambda

As melhores práticas a seguir são recomendadas para usar o AWS Lambda:

Tópicos

Para obter mais informações sobre práticas recomendadas para aplicações do Lambda, consulte Application design no Serverless Land. Você também pode entrar em contato com sua equipe de conta da AWS e solicitar uma revisão da arquitetura.

Código da função

- Separe o manipulador do Lambda da lógica central. Isso permite que você crie uma função mais fácil para teste de unidade. No Node.js isso pode ser semelhante a:

```
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
}
```

- Aproveite a reutilização do ambiente de execução para melhorar a performance da função. Inicialize clientes SDK e conexões de banco de dados fora do manipulador de funções e armazene

em cache os ativos estáticos localmente no diretório `/tmp`. As invocações subsequentes processadas pela mesma instância da função podem reutilizar esses recursos. Isso economiza custos reduzindo o runtime da função.

Para evitar possíveis vazamentos de dados entre invocações, não use o ambiente de execução para armazenar dados do usuário, eventos ou outras informações com implicações de segurança. Se sua função depende de um estado mutável que não pode ser armazenado na memória dentro do manipulador, considere criar uma função separada ou versões separadas de uma função para cada usuário.

- Use uma diretiva de keep-alive para manter conexões persistentes. O Lambda limpa conexões ociosas ao longo do tempo. A tentativa de reutilizar uma conexão ociosa ao invocar uma função resultará em um erro de conexão. Para manter sua conexão persistente, use a diretiva keep-alive associada ao runtime. Para obter um exemplo, consulte Reutilizar conexões com keep-alive em Node.js.
- Use variáveis de ambiente para passar parâmetros operacionais para sua função. Por exemplo, se estiver gravando em um bucket do Amazon S3, em vez fixar no código o nome do bucket em que você está gravando, configure o nome do bucket como uma variável de ambiente.
- Controle as dependências no pacote de implantação da função. O ambiente de execução do AWS Lambda contém várias bibliotecas, como o AWS SDK for Node.js e os tempos de execução do Python (é possível encontrar uma lista completa: Runtimes do Lambda). Para habilitar o conjunto de recursos e atualizações de segurança mais recente, o Lambda atualizará periodicamente essas bibliotecas. Essas atualizações podem introduzir alterações sutis ao comportamento de sua função do Lambda. Para ter controle total das dependências usadas por sua função, empacote todas as dependências em seu pacote de implantação.
- Minimize o tamanho do pacote de implantação às necessidades do runtime. Isso reduzirá a quantidade de tempo necessária para que seu pacote de implantação seja obtido por download e desempacotado antes da invocação. Para funções criadas em Java ou .NET Core, evite fazer upload da biblioteca inteira do AWS SDK como parte de seu pacote de implantação. Em vez disso, dependa seletivamente dos módulos que coletam os componentes do SDK necessários (por exemplo, DynamoDB, módulos do SDK do Amazon S3 e bibliotecas principais do Lambda).
- Reduza o tempo necessário para o Lambda desempacotar pacotes de implantação criados em Java colocando seus arquivos `.jar` de dependências em um diretório lib/separado. Isso é mais rápido do que colocar todo o código de sua função em um único jar com um grande número de arquivos `.class`. Para obter instruções, consulte Implantar funções do Lambda em Java com arquivos .zip ou JAR.

- Minimize a complexidade de suas dependências. Prefira frameworks mais simples que sejam carregados rapidamente no startup do ambiente de execução. Por exemplo, prefira frameworks de injeção de dependência Java (IoC) mais simples, como Dagger ou Guice, em vez de frameworks mais complexos, como o Spring Framework.
- Evite usar código recursivo em sua função do Lambda, em que a função chame a si mesma automaticamente até que alguns critérios arbitrários sejam atendidos. Isso pode levar a um volume não intencional de invocações da função e a custos elevados. Se você fizer isso acidentalmente, defina a simultaneidade reservada da função como `0` imediatamente para limitar todas as invocações da função, enquanto você atualiza o código.
- Não use APIs não documentadas e não públicas no código da função Lambda. Para os tempos de execução gerenciados pelo AWS Lambda, o Lambda aplica periodicamente atualizações funcionais e de segurança às APIs internas do Lambda. Essas atualizações internas da API podem ser incompatíveis com versões anteriores, gerando consequências não intencionais, como falhas de invocação, caso sua função tenha dependência nessas APIs não públicas. Consulte a referência da API para obter uma lista de APIs disponíveis publicamente.
- Escreva um código idempotente. Escrever um código idempotente para suas funções garante que eventos duplicados sejam tratados da mesma maneira. Seu código deve validar eventos adequadamente e lidar corretamente com eventos duplicados. Para obter mais informações, consulte Como torno minha função do Lambda idempotente?.
- Evite usar o cache do DNS Java. As funções do Lambda já armazenam as respostas de DNS em cache. Se usar outro cache do DNS, você poderá experimentar tempos limite de conexão.

  A classe `java.util.logging.Logger` pode habilitar indiretamente o cache do DNS da JVM. Para substituir as configurações padrão, defina networkaddress.cache.ttl como 0 antes de inicializar o `logger`. Exemplo:

  ```
  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);
  }
  ```

  Para evitar falhas de `UnknownHostException`, é recomendável definir `networkaddress.cache.negative.ttl` como 0. Você pode definir

essa propriedade para uma função do Lambda com a variável de ambiente `AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0`.

Desabilitar o cache do DNS da JVM não desabilita o cache do DNS gerenciado do Lambda.

Configuração da função

- Os testes de performance de sua função Lambda são uma parte essencial para garantir que você escolha a configuração do tamanho da memória ideal. Qualquer aumento no tamanho da memória dispara um aumento equivalente na CPU disponível para sua função. O uso de memória para sua função é determinado por invocação e pode ser visualizado no Amazon CloudWatch. Em cada invocação, será feita uma entrada de `REPORT:`, conforme mostrado a seguir:

  ```
  REPORT RequestId: 3604209a-e9a3-11e6-939a-754dd98c7be3 Duration: 12.34 ms Billed
   Duration: 100 ms Memory Size: 128 MB Max Memory Used: 18 MB
  ```

  Ao analisar o campo `Max Memory Used:`, você poderá determinar se a função precisa de mais memória ou se você provisionou excessivamente o tamanho da memória de sua função.

  Para encontrar a configuração de memória correta para suas funções, recomendamos usar o projeto de código aberto AWS Lambda Power Tuning. Para obter mais informações, consulte AWS Lambda Power Tuning no GitHub.

  Para otimizar a performance da função, também recomendamos a implantação de bibliotecas que possam aproveitar as Advanced Vector Extensions 2 (AVX2). Isso permite processar cargas de trabalho exigentes, incluindo inferência de machine learning, processamento de mídia, High Performance Computing (HPC – Computação de alta performance, simulações científicas e modelagem financeira. Para obter mais informações, consulte Creating faster AWS Lambda functions with AVX2.
- Teste a carga de sua função do Lambda para determinar um valor de tempo limite ideal. É importante analisar por quanto tempo sua função é executada para que você possa determinar melhor qualquer problema com um serviço de dependência que possa aumentar a simultaneidade da função além do que você espera. Isso é importante, principalmente quando sua função do Lambda faz chamadas de rede para recursos que não podem lidar com a escalabilidade do Lambda.

- Use permissões mais restritivas ao definir políticas do IAM. Compreenda os recursos e operações necessários para sua função do Lambda e limite a função de execução a essas permissões. Para ter mais informações, consulte Gerenciando permissões no AWS Lambda.
- Familiarize-se com o Cotas Lambda. O tamanho da carga, os descritores de arquivos e o espaço / tmp geralmente são ignorados ao determinar os limites de recursos de runtime.
- Exclua as funções do Lambda que você não está mais usando. Fazendo isso, as funções não utilizadas não serão contadas desnecessariamente em relação ao limite do tamanho do pacote de implantação.
- Se você estiver usando o Amazon Simple Queue Service como uma fonte de eventos, certifique-se de que o valor do tempo de invocação esperado da função não exceda o valor do Tempo limite de visibilidade na fila. Isso se aplica aCreateFunctioneUpdateFunctionConfiguration.
  - No caso de CreateFunction, o AWS Lambda fará com que haja falha no processo de criação da função.
  - No caso de UpdateFunctionConfiguration, isso pode resultar em invocações duplicadas da função.

Escalabilidade de função

- Conheça suas restrições de throughput de upstream e downstream. Embora as funções do Lambda sejam dimensionadas perfeitamente com a carga, as dependências de upstream e downstream talvez não tenham os mesmos recursos de throughput. Se você precisar definir um limite para a escalabilidade da sua função, é possível configurar a simultaneidade reservada em sua função.
- Integre a tolerância do controle de utilização. Se sua função síncrona sofrer controle de utilização devido a níveis de tráfego que excedam a taxa de escalabilidade do Lambda, você poderá aplicar as seguintes estratégias para melhorar a tolerância ao controle de utilização:
  - Use tempos-limite, novas tentativas e recuo com jitter. A implementação dessas estratégias desobstrui a repetição de invocações e ajuda a garantir que o Lambda possa aumentar a escala verticalmente em poucos segundos a fim de minimizar o controle de utilização do usuário final.
  - Use simultaneidade provisionada. A simultaneidade provisionada é o número de ambientes de execução previamente inicializados que você deseja que o Lambda aloque à sua função. O Lambda processa as solicitações recebidas usando simultaneidade provisionada, quando disponível. Se for o caso, o Lambda também pode escalar sua função acima e além da

configuração de simultaneidade provisionada. A configuração da simultaneidade provisionada gera cobranças adicionais na sua conta da AWS.

Métricas e alarmes

- UseTrabalhar com métricas de funções Lambda e Alarmes do CloudWatch, em vez de criar ou atualizar uma métrica no código da função do Lambda. É uma forma muito mais eficiente de monitorar a integridade de suas funções do Lambda permitindo detectar problemas no início do processo de desenvolvimento. Por exemplo, é possível configurar um alarme com base na duração esperada da invocação da função do Lambda para lidar com gargalos ou latências atribuíveis ao código da função.
- Utilize sua biblioteca de logs e as Métricas e dimensões do AWS Lambda para detectar erros de aplicativo (por exemplo, ERR, ERROR, WARNING etc.)
- Use o AWS Cost Anomaly Detection para detectar atividades incomuns em sua conta. O Cost Anomaly Detection usa machine learning para monitorar continuamente seu custo e uso, minimizando alertas de falsos positivos. O Cost Anomaly Detection usa dados do AWS Cost Explorer, que tem um atraso de até 24 horas. Como resultado, ele pode levar até 24 horas para detectar uma anomalia após a ocorrência do uso. Para começar a usar o Cost Anomaly Detection, primeiro cadastre-se no Explorador de Custos. Em seguida, acesse o Cost Anomaly Detection.

Trabalhar com fluxos

- Teste com diferentes tamanhos de lotes e de registros para que a frequência de sondagem de cada origem de evento seja ajustada para a rapidez com que a função pode concluir sua tarefa. O parâmetro BatchSize CreateEventSourceMapping controla o número máximo de registros que podem ser enviados para sua função a cada invocação. Um tamanho de lote maior geralmente absorve de maneira mais eficiente a sobrecarga da invocação em um conjunto maior de registros aumentando o throughput.

  Por padrão, o Lambda invoca a função assim que os registros estão disponíveis. Se o lote que o Lambda lê da fonte de eventos tiver apenas um registro, o Lambda enviará apenas um registro à função. Para evitar a invocação da função com poucos registros, instrua a fonte de eventos para armazenar os registros em buffer por até cinco minutos, configurando uma janela de lotes. Antes de invocar a função, o Lambda continua a ler registros da fonte de eventos até coletar um lote

inteiro, até que a janela de lote expire ou até que o lote atinja o limite de carga útil de 6 MB. Para ter mais informações, consulte Comportamento de lotes.

> **Warning**
>
> Os mapeamentos da origem do evento do Lambda processam cada evento ao menos uma vez, podendo haver o processamento duplicado de registros. Para evitar possíveis problemas relacionados a eventos duplicados, é altamente recomendável tornar o código da função idempotente. Para saber mais, consulte Como tornar minha função do Lambda idempotente no Centro de Conhecimentos da AWS.

- Aumente o throughput de processamento de transmissão do Kinesis adicionando fragmentos. Uma transmissão do Kinesis é composto de um ou mais fragmentos. O Lambda sondará cada fragmento com no máximo uma invocação simultânea. Por exemplo, se o seu fluxo tiver 100 estilhaços ativos, haverá, no máximo, 100 invocações de função do Lambda em execução simultaneamente. O aumento do número de estilhaços aumentará diretamente o número máximo de invocações simultâneas de função do Lambda e poderá aumentar o throughput para processamento de transmissões do Kinesis. Se você estiver aumentando o número de fragmentos em uma transmissão do Kinesis, certifique-se de ter escolhido uma boa chave de partição (consulte Chaves de partição) para seus dados, para que os registros relacionados terminem nos mesmos fragmentos e seus dados sejam bem distribuídos.
- Use o Amazon CloudWatch no IteratorAge para determinar se a transmissão do Kinesis está sendo processada. Por exemplo, configure um alarme do CloudWatch com uma configuração máxima de 30000 (30 segundos).

Melhores práticas de segurança

- Monitore seu uso do AWS Lambda em relação às práticas recomendadas de segurança com o AWS Security Hub. O Security Hub usa controles de segurança para avaliar configurações de recursos e padrões de segurança que ajudam você a cumprir vários frameworks de conformidade. Para obter mais informações sobre como usar o Security Hub para avaliar os recursos do Lambda, consulte Controles do AWS Lambda no Guia do usuário do AWS Security Hub.
- Monitore os logs de atividade de rede do Lambda usando a proteção do Lambda do Amazon GuardDuty. A proteção do Lambda ajuda você a identificar possíveis ameaças à segurança nas invocações de funções do Lambda em sua Conta da AWS. Por exemplo, se uma de suas funções consultar um endereço IP associado a atividades relacionadas a criptomoeda. O GuardDuty

monitora os logs de atividade de rede gerados mediante a invocação de uma função do Lambda. Para saber mais, consulte Proteção do Lambda no Guia do usuário do Amazon GuardDuty.

Gerenciando permissões no AWS Lambda

É possível usar o AWS Identity and Access Management (IAM) para gerenciar permissões no AWS Lambda. Há duas categorias principais de permissões que você precisa considerar ao trabalhar com funções do Lambda:

- Permissões que suas funções do Lambda precisam para executar ações da API e acessar outros recursos da AWS.
- Permissões que outros usuários e entidades da AWS precisam para acessar suas funções do Lambda

As funções Lambda do geralmente precisam acessar outros recursos da AWS e realizar várias operações de API nesses recursos. Por exemplo, você pode ter uma função do Lambda que responda a um evento ao atualizar as entradas do banco de dados do Amazon DynamoDB. Nesse caso, sua função precisa de permissões para acessar o banco de dados, bem como permissões para colocar ou atualizar itens nesse banco de dados.

Você define as permissões que sua função do Lambda precisa em um perfil do IAM especial chamado perfil de execução. Nesse perfil, é possível anexar uma política que define cada permissão que sua função precisa para acessar outros recursos da AWS e ler de origens de eventos. Cada função do Lambda deve ter um perfil de execução. No mínimo, sua função de execução deve ter acesso ao Amazon CloudWatch porque as funções do Lambda são registradas no CloudWatch Logs por padrão. É possível anexar a política gerenciada pela `AWSLambdaBasicExecutionRole` ao seu perfil de execução para satisfazer esse requisito.

Para conceder permissões a outras contas, organizações e serviços da AWS para acessar seus recursos do Lambda, há algumas opções:

- É possível usar políticas baseadas em identidade para conceder a outros usuários acesso aos seus recursos do Lambda. As políticas baseadas em identidade podem se aplicar diretamente aos usuários ou a funções e grupos associados a um usuário.
- É possível usar políticas baseadas em recursos para conceder a outras contas e serviços da AWS permissões para acessar seus recursos do Lambda. Quando um usuário tenta acessar um recurso do Lambda, o Lambda considera as políticas baseadas em identidade e a política baseada em recursos do usuário. Quando um serviço da AWS, como o Amazon Simple Storage Service (Amazon S3), chama sua função do Lambda, o Lambda considera apenas a política baseada em recursos.

- É possível usar um modelo de controle de acesso por atributo (ABAC) para controlar o acesso às suas funções do Lambda. Com o ABAC, você pode anexar tags a uma função do Lambda, transmiti-las em determinadas solicitações de API ou anexá-las à entidade principal do IAM que está fazendo a solicitação. Especificar as mesmas tags no elemento de condição de uma política do IAM para controlar o acesso à função.

Na AWS, é prática recomendada conceder apenas as permissões necessárias para executar uma tarefa (permissões de privilégio mínimo). Para implementar isso no Lambda, recomendamos começar com uma política gerenciada pela AWS. Use essas políticas gerenciadas no estado em que se encontram ou como um ponto de partida para escrever suas próprias políticas mais restritivas.

Para ajudar você a ajustar suas permissões para acesso com privilégios mínimos, o Lambda fornece algumas condições adicionais que você pode incluir em suas políticas. Para ter mais informações, consulte the section called “Recursos e condições”.

Para obter mais informações sobre o IAM, consulte o Manual do usuário do IAM.

Definir permissões de uma função do Lambda com um perfil de execução

A função de execução de uma função do Lambda é uma perfil do AWS Identity and Access Management (IAM) que concede à função permissão para acessar serviços e recursos da AWS. Por exemplo, é possível criar uma função de execução que tenha permissão para enviar logs ao Amazon CloudWatch e carregar os dados de rastreamento no AWS X-Ray. Esta página fornece informações sobre como criar, visualizar e gerenciar o perfil de execução de uma função do Lambda.

O Lambda assume automaticamente seu perfil de execução quando você invoca sua função. Evite chamar `sts:AssumeRole` manualmente para assumir o perfil de execução no código da sua função. Se o caso de uso exigir que o perfil assuma a si mesmo, será necessário incluir o perfil em si como uma entidade principal confiável na política de confiança do perfil. Para obter mais informações sobre como modificar uma política de confiança de perfil, consulte Modificar a política de confiança de uma função (console) no Guia do usuário do IAM.

Para que o Lambda assuma seu perfil de execução de forma adequada, a política de confiança do perfil deve especificar a entidade principal de serviço do Lambda (`lambda.amazonaws.com`) como um serviço confiável.

Tópicos

- Criar uma função de execução no console do IAM
- Criar e gerenciar perfis com a AWS CLI
- Conceda acesso de menor privilégio à sua função de execução do Lambda
- Visualizar e atualizar permissões no perfil de execução
- Trabalhar com políticas gerenciadas pela AWS no perfil de execução
- Usar o ARN da função de origem para controlar o comportamento de acesso da função

Criar uma função de execução no console do IAM

Por padrão, o Lambda cria uma função de execução com permissões mínimas quando você cria uma função no console do Lambda. Especificamente, esse perfil de execução inclui a política gerenciada `AWSLambdaBasicExecutionRole`, que concede à sua função permissões básicas para registrar eventos no Amazon CloudWatch Logs.

Normalmente, suas funções precisam de permissões adicionais para realizar tarefas mais significativas. Por exemplo, você pode ter uma função do Lambda que responda a um evento ao atualizar as entradas do banco de dados do Amazon DynamoDB. É possível criar um perfil de execução com as permissões necessárias usando o console do IAM.

Como criar uma função de execução no console do IAM

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione Create role (Criar função).
3. Em Tipo de entidade confiável, selecione Serviço da AWS.
4. Em Use case (Caso de uso), escolha Lambda.
5. Escolha Próximo.
6. Selecione as políticas gerenciadas pela AWS que deseja anexar ao seu perfil. Por exemplo, se sua função precisar acessar o DynamoDB, selecione a política gerenciada AWSLambdaDynamoDBExecutionRole.
7. Escolha Próximo.
8. Digite um Role name e escolha Create role.

Para obter instruções, consulte Criar uma função para um serviço da AWS (console) no Guia do usuário do IAM.

Depois de criar o perfil de execução, anexe-o à sua função. Ao criar uma função no console do Lambda, você poderá anexar qualquer perfil de execução criado anteriormente à função. Se você quiser anexar um novo perfil de execução a uma função existente, siga as etapas em .

Criar e gerenciar perfis com a AWS CLI

Para criar uma função de execução com a AWS Command Line Interface (AWS CLI), use o comando create-role. Ao usar esse comando, é possível especificar a política de confiança em linha. A política de confiança de um perfil concede a permissão de entidades principais especificadas para assumir o perfil. No exemplo a seguir, você concede à entidade principal do serviço Lambda permissão para assumir seu perfil. Os requisitos para escapar de aspas na string JSON podem variar dependendo do 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"}]}'
```

Também é possível definir a política de confiança para a função usando um arquivo JSON separado. No exemplo a seguir, `trust-policy.json` é um arquivo no diretório atual.

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
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
                }
            ]
        }
```

```
    }
}
```

Para adicionar permissões à função, use o comando attach-policy-to-role. Os comandos a seguir adicionam a política gerenciada `AWSLambdaBasicExecutionRole` ao perfil de execução `lambda-ex`.

```
aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole
```

Depois de criar o perfil de execução, anexe-o à sua função. Ao criar uma função no console do Lambda, você poderá anexar qualquer perfil de execução criado anteriormente à função. Se você quiser anexar um novo perfil de execução a uma função existente, siga as etapas em .

Conceda acesso de menor privilégio à sua função de execução do Lambda

Quando você cria um perfil do IAM pela primeira vez para sua função do Lambda durante a fase de desenvolvimento, às vezes você pode conceder permissões além do que é necessário. Antes de publicar sua função no ambiente de produção, como prática recomendada, ajuste a política para incluir somente as permissões necessárias. Para obter mais informações, consulte Aplicar permissões de privilégio mínimo, no Guia do usuário do IAM.

Use o IAM Access Analyzer para ajudar a identificar as permissões necessárias para a política de função de execução do IAM. O IAM Access Analyzer revisa seus logs do AWS CloudTrail para o intervalo de datas especificado e gera um modelo de política com apenas as permissões que a função utilizou durante esse período. Você pode usar o modelo para criar uma política gerenciada com permissões refinadas e anexá-la à função do IAM. Dessa forma, você concede apenas as permissões necessárias à interação com os recursos da AWS, de acordo com a especificidade do caso de uso.

Para obter mais informações, consulte Generate policies based on access activity (Gerar políticas com base na atividade de acesso), no Guia do usuário do IAM.

Visualizar e atualizar permissões no perfil de execução

Este tópico aborda como você pode visualizar e atualizar o perfil de execução da sua função.

Tópicos

- Visualizar o perfil de execução de uma função

- Atualizar o perfil de execução de uma função

Visualizar o perfil de execução de uma função

Para visualizar o perfil de execução de uma função, use o console do Lambda.

Para visualizar o perfil de execução de uma função (console)

1. Abra a página Funções do console do Lambda.
2. Escolha o nome de uma função.
3. Escolha Configuration (Configuração) e depois Permissions (Permissões).
4. Em Perfil de execução, é possível ver o perfil que está sendo usado atualmente como perfil de execução da função. Por conveniência, você pode visualizar todos os recursos e ações que a função pode acessar na seção Resumo do recurso. Você também pode escolher um serviço na lista suspensa para ver as permissões relacionadas a ele.

Atualizar o perfil de execução de uma função

Adicione ou remova permissões da função de execução de uma função a qualquer momento ou configure a função para usar uma diferente. Se sua função precisar acessar outros serviços ou recursos, você deverá adicionar as permissões necessárias ao perfil de execução.

Ao adicionar permissões à sua função, faça também uma atualização simples em seu código ou configuração. Isso força as instâncias em execução da função, com credenciais desatualizadas, a serem encerradas e substituídas.

Para atualizar o perfil de execução de uma função, use o console do Lambda.

Para atualizar o perfil de execução de uma função (console)

1. Abra a página Funções do console do Lambda.
2. Escolha o nome de uma função.
3. Escolha Configuration (Configuração) e depois Permissions (Permissões).
4. Em Perfil de execução, escolha Editar.
5. Se você quiser atualizar sua função para usar um perfil diferente como perfil de execução, escolha o novo perfil no menu suspenso em Perfil existente.

> **Note**
>
> Se desejar atualizar as permissões em um perfil de execução existente, você só poderá fazer isso no console do AWS Identity and Access Management (IAM).

Se você quiser criar um novo perfil para usar como perfil de execução, escolha Criar um novo perfil baseado em modelos de políticas da AWS em Perfil de execução. Em seguida, insira um nome para seu novo perfil em Nome do perfil e especifique as políticas que você deseja anexar ao novo perfil em Modelos de política.

6. Escolha Salvar.

Trabalhar com políticas gerenciadas pela AWS no perfil de execução

As seguintes políticas gerenciadas da AWS oferecem permissões obrigatórias para o uso dos recursos do Lambda:

Alteração	Descrição	Data
AWSLambdaMSKExecutionRole: o Lambda adicionou a permissão kafka:DescribeClusterV2 a esta política.	`AWSLambdaMSKExecutionRole` concede permissões para ler e acessar registros de um cluster do Amazon Managed Streaming for Apache Kafka (Amazon MSK), gerenciar interfaces de rede elástica (ENIs) e gravar no CloudWatch Logs.	17 de junho de 2022
AWSLambdaBasicExecutionRole: o Lambda começou a monitorar mudanças feitas a essa política.	`AWSLambdaBasicExecutionRole` concede permissões para carregar registros para o CloudWatch	14 de fevereiro de 2022
AWSLambdaDynamoDBExecutionRole: o Lambda	`AWSLambdaDynamoDBExecutionRole` concede	14 de fevereiro de 2022

Alteração	Descrição	Data
começou a monitorar mudanças feitas a essa política.	permissões para ler registros de um fluxo do Amazon DynamoDB Streams e gravar no CloudWatch Logs.	
AWSLambdaKinesisExecutionRole: o Lambda começou a monitorar mudanças feitas a essa política.	`AWSLambdaKinesisExecutionRole` concede permissões para ler eventos de um fluxo de dados do Amazon Kinesis e gravar no CloudWatch Logs.	14 de fevereiro de 2022
AWSLambdaMSKExecutionRole: o Lambda começou a monitorar mudanças feitas a essa política.	`AWSLambdaMSKExecutionRole` concede permissões para ler e acessar registros de um cluster do Amazon Managed Streaming for Apache Kafka (Amazon MSK), gerenciar interfaces de rede elástica (ENIs) e gravar no CloudWatch Logs.	14 de fevereiro de 2022
AWSLambdaSQSQueueExecutionRole: o Lambda começou a monitorar mudanças feitas a essa política.	`AWSLambdaSQSQueueExecutionRole` concede permissões para ler uma mensagem de uma fila do Amazon Simple Queue Service (Amazon SQS) e gravar no CloudWatch Logs.	14 de fevereiro de 2022
AWSLambdaVPCAccessExecutionRole: o Lambda começou a monitorar mudanças feitas a essa política.	`AWSLambdaVPCAccessExecutionRole` concede permissões para gerenciar ENIs em uma Amazon VPC e gravar no CloudWatch Logs.	14 de fevereiro de 2022

Alteração	Descrição	Data
AWSXRayDaemonWriteAccess: o Lambda começou a monitorar mudanças feitas a essa política.	`AWSXRayDaemonWriteAccess` concede permissões para carregar dados de rastreamento no X-Ray.	14 de fevereiro de 2022
CloudWatchLambdaInsightsExecutionRolePolicy: o Lambda começou a monitorar mudanças feitas a essa política.	`CloudWatchLambdaInsightsExecutionRolePolicy` concede permissão para gravar métricas de runtime no CloudWatch Lambda Insights.	14 de fevereiro de 2022
AmazonS3ObjectLambdaExecutionRolePolicy: o Lambda começou a monitorar mudanças feitas a essa política.	`AmazonS3ObjectLambdaExecutionRolePolicy` concede permissões para interagir com o Lamda de objetos do Amazon Simple Storage Service (Amazon S3) e gravar no CloudWatch Logs.	14 de fevereiro de 2022

Para alguns atributos, o console do Lambda tenta adicionar permissões ausentes à sua função de execução em uma política gerenciada pelo cliente. Essas políticas podem tornar-se numerosas. Para evitar a criação de políticas adicionais, acrescente as políticas gerenciadas da AWS relevantes à função de execução antes de habilitar os atributos.

Quando você usa um mapeamento de fontes de eventos para invocar a função, o Lambda usa a função de execução a fim de ler dados de eventos. Por exemplo, um mapeamento da origem do evento para o Kinesis lê eventos de um fluxo de dados e os envia para a sua função em lotes.

Quando um serviço assume uma função em sua conta, você pode incluir as chaves de contexto de condição global `aws:SourceAccount` e `aws:SourceArn` em sua política de confiança de função para limitar o acesso à função apenas a solicitações geradas pelos recursos esperados. Para obter mais informações, consulte Prevenção de confused deputy entre serviços para o AWS Security Token Service.

Além das políticas gerenciadas da AWS, o console do Lambda fornece modelos para criar uma política personalizada com as permissões para casos de uso adicionais. Ao criar uma função no console do Lambda, opte por criar uma nova função de execução com permissões de um ou mais modelos. Esses modelos também são aplicados automaticamente quando você cria uma função a partir de um esquema, ou quando configura opções que exijam acesso a outros serviços. Os modelos de exemplo estão disponíveis no repositório do GitHub deste guia.

Usar o ARN da função de origem para controlar o comportamento de acesso da função

É comum que o seu código de função do Lambda faça solicitações de API para outros serviços da AWS. Para fazer essas solicitações, o Lambda gera um conjunto efêmero de credenciais, assumindo a função de execução da sua função. Essas credenciais estão disponíveis como variáveis de ambiente durante a invocação da sua função. Ao trabalhar com SDKs da AWS, não é necessário fornecer credenciais para o SDK diretamente no código. Por padrão, a cadeia de provedores de credenciais verifica sequencialmente cada local em que você pode definir credenciais e seleciona o primeiro disponível, que geralmente corresponde às variáveis de ambiente (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` e `AWS_SESSION_TOKEN`).

O Lambda injetará o ARN da função de origem no contexto de credenciais se a solicitação for uma solicitação de API da AWS proveniente de seu ambiente de execução. O Lambda também injetará o ARN da função de origem para as seguintes solicitações de API da AWS que o Lambda faz em seu nome de forma externa ao ambiente de execução:

Serviço	Ação	Motivo
CloudWatch Logs	`CreateLogGroup` , `CreateLogStream` , `PutLogEvents`	Para armazenar logs em um grupo de logs do CloudWatch Logs.
X-Ray	`PutTraceSegments`	Para enviar dados de rastreamento para o X-Ray.
Amazon EFS	`ClientMount`	Para conectar a função a um sistema de arquivos do Amazon Elastic File System (Amazon EFS).

Outras chamadas de API da AWS feitas pelo Lambda em seu nome de forma externa ao seu ambiente de execução e que usam o mesmo perfil de execução não contêm o ARN da função de origem. Alguns exemplos dessas chamadas de API fora do ambiente de execução incluem:

- Chamadas para o AWS Key Management Service (AWS KMS) para criptografar e descriptografar variáveis de ambiente automaticamente.
- Chamadas para o Amazon Elastic Compute Cloud (Amazon EC2) com a finalidade de criar interfaces de rede elástica (ENIs) para uma função habilitada para VPC.
- Chamadas para serviços da AWS, como o Amazon Simple Queue Service (Amazon SQS), para leitura de uma origem de evento configurada como um mapeamento da origem do evento.

Com o ARN da função de origem no contexto de credenciais, você pode verificar se uma chamada ao seu recurso veio do código de uma função do Lambda específica. Para verificar isso, use a chave de condição `lambda:SourceFunctionArn` em uma política baseada em identidade do IAM ou uma política de controle de serviços (SCP).

> **Note**
>
> Não é possível usar a chave de condição `lambda:SourceFunctionArn` em políticas baseadas em recursos.

Com essa chave de condição nas políticas baseadas em identidade ou SCPs, você pode implementar controles de segurança para as ações de API que o código da função realiza em outros serviços da AWS. Isso tem algumas aplicações de segurança importantes, como ajudar a identificar a origem de um vazamento de credenciais.

> **Note**
>
> A chave de condição `lambda:SourceFunctionArn` é diferente das chaves de condição `lambda:FunctionArn` e `aws:SourceArn`. A chave de condição `lambda:FunctionArn` aplica-se somente a mapeamentos da origem do evento e ajuda a definir quais funções a sua origem de evento pode invocar. A chave de condição `aws:SourceArn` se aplica apenas a políticas nas quais a função do Lambda é o recurso visado e ajuda a definir quais outros serviços e recursos da AWS podem invocar essa função. A chave de condição `lambda:SourceFunctionArn` pode ser aplicada a qualquer política baseada em

identidade ou SCP para definir as funções específicas do Lambda que têm permissões para fazer determinadas chamadas de API da AWS para outros recursos.

Para usar `lambda:SourceFunctionArn` na sua apólice, inclua-a como uma condição com qualquer um dos operadores de condição de ARN. O valor da chave deve ser um ARN válido.

Por exemplo, suponha que o código da função do Lambda faça uma chamada `s3:PutObject` que se destina a um bucket do Amazon S3 específico. Talvez você queira permitir que somente uma função do Lambda específica tenha acesso `s3:PutObject` a esse bucket. Nesse caso, a função de execução da função deve ter uma política anexada similar a esta:

Example política que concede acesso para uma função do Lambda específica a um recurso do 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"
                }
            }
        }
    ]
}
```

Essa política apenas permite acesso a `s3:PutObject` se a origem for a função do Lambda com o ARN `arn:aws:lambda:us-east-1:123456789012:function:source_lambda`. Essa política não permite acesso a `s3:PutObject` para qualquer outra identidade de chamada. Isso acontece mesmo que uma função ou entidade diferente faça uma chamada `s3:PutObject` com a mesma função de execução.

> **Note**
>
> A chave de condição `lambda:SourceFunctionARN` não é compatível com versões da função do Lambda ou aliases da função. Se você usar o ARN para uma determinada versão ou alias da função, ela não terá permissão para realizar a ação especificada. Certifique-se de usar o ARN não qualificado para sua função sem uma versão ou sufixo de alias.

Você também pode usar `lambda:SourceFunctionArn` em SCPs. Por exemplo, suponha que você queira restringir o acesso ao seu bucket a um único código de função do Lambda ou às chamadas de uma nuvem privada virtual (VPC) da Amazon específica. O SCP a seguir ilustra isso.

Example política que nega acesso ao Amazon S3 sob condições específicas

```
{
    "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"
```

```
                }
            }
        }
    ]
}
```

Essa política nega todas as ações do S3, a menos que sejam provenientes de uma função específica do Lambda com o ARN `arn:aws:lambda:*:123456789012:function:source_lambda` ou que sejam provenientes da VPC especificada. O operador `StringNotEqualsIfExists` instrui o IAM a processar essa condição somente se a chave `aws:SourceVpc` estiver presente na solicitação. Da mesma forma, o IAM apenas considerará o operador`ArnNotEqualsIfExists` somente se o `lambda:SourceFunctionArn` existir.

Conceder a outras entidades da AWS acesso às suas funções do Lambda

Para conceder permissões a outras contas, organizações e serviços da AWS para acessar seus recursos do Lambda, há algumas opções:

- É possível usar políticas baseadas em identidade para conceder a outros usuários acesso aos seus recursos do Lambda. As políticas baseadas em identidade podem se aplicar diretamente aos usuários ou a funções e grupos associados a um usuário.
- É possível usar políticas baseadas em recursos para conceder a outras contas e serviços da AWS permissões para acessar seus recursos do Lambda. Quando um usuário tenta acessar um recurso do Lambda, o Lambda considera as políticas baseadas em identidade e a política baseada em recursos do usuário. Quando um serviço da AWS, como o Amazon Simple Storage Service (Amazon S3), chama sua função do Lambda, o Lambda considera apenas a política baseada em recursos.
- É possível usar um modelo de controle de acesso por atributo (ABAC) para controlar o acesso às suas funções do Lambda. Com o ABAC, você pode anexar tags a uma função do Lambda, transmiti-las em determinadas solicitações de API ou anexá-las à entidade principal do IAM que está fazendo a solicitação. Especificar as mesmas tags no elemento de condição de uma política do IAM para controlar o acesso à função.

Para ajudar você a ajustar suas permissões para acesso com privilégios mínimos, o Lambda fornece algumas condições adicionais que você pode incluir em suas políticas. Para ter mais informações, consulte the section called “Recursos e condições”.

Trabalhando com políticas do IAM baseadas em identidade no Lambda

Use políticas baseadas em identidade no AWS Identity and Access Management (IAM) para conceder a usuários na conta acesso ao Lambda. As políticas baseadas em identidade podem se aplicar diretamente aos usuários ou a funções e grupos associados a um usuário. Também é possível conceder a usuários em outra conta permissão para assumir uma função na conta e acessar os recursos do Lambda. Esta página apresenta um exemplo de como as políticas baseadas em identidade podem ser usadas para o desenvolvimento de funções.

O Lambda fornece políticas gerenciadas pela AWS que concedem acesso a ações da API do Lambda e, em alguns casos, acesso a outros serviços da AWS usados para desenvolver e gerenciar

recursos do Lambda. Lambda atualiza as políticas gerenciadas conforme necessário para garantir que os usuários tenham acesso a novos recursos quando eles forem lançados.

- AWSLambda_FullAccess: concede acesso total a ações do Lambda e a outros serviços da AWS usados para desenvolver e manter recursos do Lambda. Esta política foi criada com a redução do escopo da política anterior AWSLambdaFullAccess.
- AWSLambda_ReadOnlyAccess: concede acesso somente leitura aos recursos do Lambda. Esta política foi criada com a redução do escopo da política anterior AWSLambdaReadOnlyAccess.
- AWSLambdaRole: concede permissões para invocar funções do Lambda.

As políticas gerenciadas do AWS concedem permissão a ações da API sem restringir as funções ou as camadas do Lambda que um usuário pode modificar. Para um controle refinado, crie as próprias políticas que limitam o escopo das permissões de um usuário.

Seções

Escrever um exemplo de política que conceda permissões de usuário a uma função

Use políticas baseadas em identidade do para permitir que os usuários executem operações em funções do Lambda.

> **Note**
>
> Para uma função definida como uma imagem de contêiner, a permissão do usuário para acessar a imagem DEVE ser configurada no Amazon Elastic Container Registry Para obter um exemplo, consulte Permissões do Amazon ECR.

A seguir, um exemplo de uma política de permissões com escopo limitado. Ele permite que um usuário crie e gerencie funções do Lambda com um prefixo designado (`intern-`) e configuradas com uma função de execução designada.

Example Política de desenvolvimento da função

```
{
    "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-*"
        }
    ]
}
```

As permissões na política são organizadas em declarações com base nos recursos e nas condições compatíveis.

- `ReadOnlyPermissions`: o console do Lambda usa essas permissões quando você procura e exibe funções. Elas não oferecem suporte a padrões ou condições de recursos.

```
            "Action": [
                "lambda:GetAccountSettings",
                "lambda:GetEventSourceMapping",
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration",
                "lambda:GetFunctionCodeSigningConfig",
                "lambda:GetFunctionConcurrency",
                "lambda:ListEventSourceMappings",
                "lambda:ListFunctions",
                "lambda:ListTags",
```

```
            "iam:ListRoles"
        ],
        "Resource": "*"
```

- `DevelopFunctions`: use qualquer ação do Lambda que opere em funções prefixadas com `intern-`, exceto `AddPermission` e `PutFunctionConcurrency`. `AddPermission` modifica a política baseada em recursos na função e pode ter implicações de segurança. `PutFunctionConcurrency` reserva capacidade de escalabilidade para uma função e pode consumir a capacidade de outras funções.

```
        "NotAction": [
            "lambda:AddPermission",
            "lambda:PutFunctionConcurrency"
        ],
        "Resource": "arn:aws:lambda:*:*:function:intern-*"
```

- `DevelopEventSourceMappings`: gerencie mapeamentos de origem de eventos em funções prefixadas com `intern-`. Essas ações operam em mapeamentos de origem do evento, mas é possível restringi-las por função com uma condição.

```
        "Action": [
            "lambda:DeleteEventSourceMapping",
            "lambda:UpdateEventSourceMapping",
            "lambda:CreateEventSourceMapping"
        ],
        "Resource": "*",
        "Condition": {
            "StringLike": {
                "lambda:FunctionArn": "arn:aws:lambda:*:*:function:intern-*"
            }
        }
```

- `PassExecutionRole`: exiba e passe apenas uma função chamada `intern-lambda-execution-role`, que deve ser criada e gerenciada por um usuário com permissões do IAM. `PassRole` é usado quando você atribui uma função de execução a uma função.

```
        "Action": [
            "iam:ListRolePolicies",
            "iam:ListAttachedRolePolicies",
```

```
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:PassRole",
                "iam:SimulatePrincipalPolicy"
            ],
            "Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
```

- `ViewLogs`: use o CloudWatch Logs para exibir logs de funções prefixadas com `intern-`.

```
            "Action": [
                "logs:*"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/lambda/intern-*"
```

Essa política permite que um usuário começar a usar com o Lambda, sem colocar os recursos de outros usuários em risco. Ela não permite que um usuário configure uma função para ser disparada ou acione outros serviços da AWS, o que exige permissões mais amplas do IAM. Ela também não inclui permissão para serviços que não ofereçam suporte a políticas de escopo limitado, como CloudWatch e X-Ray. Use as políticas somente leitura desses serviços para conceder ao usuário acesso a métricas e dados de rastreamento.

Ao configurar triggers para a sua função, você precisa de acesso para usar o produto da AWS que invoca a sua função. Por exemplo, para configurar um acionador do Amazon S3, é necessária a permissão para ações do Amazon S3 a fim de gerenciar notificações do bucket. Muitas dessas permissões estão incluídas na política gerenciada AWSLambdaFullAccess. As políticas de exemplo estão disponíveis no repositório do GitHub deste guia.

Escrever um exemplo de política que conceda permissões para usar camadas

A política a seguir concede permissões a um usuário para criar camadas e usá-las com funções. Os padrões de recursos permitem que o usuário trabalhe em qualquer região da AWS e com qualquer versão de camada, desde que o nome da camada comece com `test-`.

Example política de desenvolvimento da camada

```
{
    "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-*:*"
        }
    ]
}
```

Também é possível impor o uso da camada durante a criação da função e a configuração com a condição `lambda:Layer`. Por exemplo, evite que os usuários usem camadas publicadas por outras contas. A política a seguir adiciona uma condição às ações `CreateFunction` e `UpdateFunctionConfiguration` para exigir que todas as camadas especificadas venham da conta `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:*:*"
                    ]
                }
            }
```

```
        }
    ]
}
```

Para garantir que a condição se aplique, verifique se não há outras instruções concedendo ao usuário permissão para essas ações.

Implementar acesso entre contas com políticas baseadas em identidade

Aplique qualquer uma das políticas e instruções anteriores a uma função, que é possível acabar compartilhando com outra conta para conceder a ela acesso aos recursos do Lambda. Diferentemente de um usuário, um perfil não tem credenciais para autenticação. Em vez disso, ela tem uma política confiável que especifica quem pode assumir a função e usar as permissões.

Use funções entre contas para dar a contas nas quais você confia acesso a ações e recursos do Lambda. Se você só quiser conceder permissão para invocar uma função ou usar uma camada, use políticas baseadas em recursos.

Para obter mais informações, consulte Funções do IAM no Guia do usuário do IAM.

Trabalhar com políticas baseadas em recursos no Lambda

O Lambda oferece suporte a políticas de permissões baseadas em recursos para funções e camadas do Lambda. As políticas baseadas em recursos permitem conceder permissão de uso a outras contas ou organizações da AWS por recurso. Também é possível usar uma política baseada em recurso para permitir que um serviço da AWS invoque a função em seu nome.

Para funções do Lambda, conceda uma permissão à conta para invocar ou gerenciar uma função. Você também pode usar uma só política baseada em recursos para conceder permissões a uma organização inteira no AWS Organizations. Também é possível usar políticas baseadas em recurso para conceder permissão de invocação a um serviço da AWS que invoca uma função em resposta a atividades em sua conta.

Como visualizar a política baseada em recursos de uma função

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Configuração e, em seguida, escolha Permissões.
4. Role para baixo até Política baseada em recursos e escolha Exibir documento de política. A política baseada em recursos mostra as permissões aplicadas quando outra conta ou serviço da

AWS tenta acessar a função. O exemplo a seguir mostra uma instrução que permite ao Amazon S3 invocar uma função chamada `my-function` para um bucket chamado `DOC-EXAMPLE-BUCKET` na conta `123456789012`.

Example Política baseada em recurso

```
{
    "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"
              }
            }
        }
     ]
}
```

Para camadas do Lambda, é possível usar somente uma política baseada em recurso em uma versão específica da camada, em vez de toda a camada. Além de políticas que concedem permissão a uma única conta ou a várias as contas, para camadas, também é possível conceder permissão a todas as contas em uma organização.

> **Note**
>
> Só é possível atualizar políticas baseadas em recursos para recursos do Lambda dentro do escopo das ações de API AddPermission e AddLayerVersionPermission. Atualmente,

não é possível criar políticas para os recursos do Lambda em JSON ou usar condições não mapeadas para parâmetros dessas ações.

As políticas baseadas em recursos se aplicam a uma única função, versão, alias ou versão da camada. Elas concedem permissão para um ou mais serviços e contas. Para contas confiáveis que você deseja que tenham acesso a vários recursos, ou para usar ações de API não compatíveis com políticas baseadas em recursos, use funções entre contas.

Tópicos

Ações da API com suporte

As seguintes ações de API do Lambda são compatíveis com as políticas baseadas em recursos:

- CreateAlias
- DeleteAlias
- DeleteFunction
- DeleteFunctionConcurrency
- DeleteFunctionEventInvokeConfig
- DeleteProvisionedConcurrencyConfig
- GetAlias
- GetFunction
- GetFunctionConcurrency
- GetFunctionConfiguration
- GetFunctionEventInvokeConfig
- GetPolicy

- GetProvisionedConcurrencyConfig
- Invoke
- ListAliases
- ListFunctionEventInvokeConfigs
- ListProvisionedConcurrencyConfigs
- ListTags
- ListVersionsByFunction
- PublishVersion
- PutFunctionConcurrency
- PutFunctionEventInvokeConfig
- PutProvisionedConcurrencyConfig
- TagResource
- UntagResource
- UpdateAlias
- UpdateFunctionCode
- UpdateFunctionEventInvokeConfig

Conceder acesso de função aos serviços da AWS

Ao usar um serviço da AWS para invocar a função, você concede permissão em uma instrução em uma política baseada em recursos. É possível aplicar a instrução a toda a função a ser invocada ou gerenciada ou limitar a instrução a uma única versão ou alias.

> **Note**
>
> Quando você adiciona um acionador à função com o console do Lambda, este atualiza a política baseada em recursos da função para permitir que o serviço a invoque. Para conceder permissões a outras contas ou serviços que não estejam disponíveis no console do Lambda, é possível usar a AWS CLI.

Adicione uma instrução com o comando `add-permission`. A instrução de política baseada em recursos mais simples permite que um serviço invoque uma função. O comando a seguir concede permissão ao Amazon SNS para invocar uma função chamada `my-function`.

```
aws lambda add-permission --function-name my-function --action lambda:InvokeFunction --
statement-id sns \
    --principal sns.amazonaws.com --output text
```

A seguinte saída deverá ser mostrada:

```
{"Sid":"sns","Effect":"Allow","Principal":
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-
east-2:123456789012:function:my-function"}
```

Isso permite que o Amazon SNS chame a API `lambda:Invoke` para a função, mas não restrinja o tópico do Amazon SNS que aciona a invocação. Para garantir que a função só seja invocada por um recurso específico, especifique o Amazon Resource Name (ARN – Nome de recurso da Amazon) do recurso com a opção `source-arn`. O comando a seguir só permite que o Amazon SNS invoque a função para assinaturas de um tópico chamado `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
```

Alguns serviços podem invocar funções em outras contas. Se você especificar um ARN de origem que tenha o ID da conta, isso não será um problema. No entanto, para o Amazon S3 a origem é um bucket cujo ARN não tem um ID da conta. É possível que você consiga excluir o bucket e outra conta consiga criar um bucket com o mesmo nome. Use a opção `source-account` com o ID da sua conta para garantir que apenas os recursos na conta possam invocar a função.

```
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
```

Como conceder acesso de função a uma organização

Para conceder permissões a uma organização no AWS Organizations, especifique o ID da organização como o `principal-org-id`. Os seguinte comando AddPermission da AWS CLI concede acesso de invocação a todos os usuários na organização `o-a1b2c3d4e5f`.

```
aws lambda add-permission --function-name example \
```

```
    --statement-id PrincipalOrgIDExample --action lambda:InvokeFunction \
    --principal * --principal-org-id o-a1b2c3d4e5f
```

> **Note**
>
> Nesse comando, `Principal` é `*`. Isso significa que todos os usuários na organização `o-a1b2c3d4e5f` recebem permissões de invocação de função. Se você especificar uma conta ou função da AWS como a `Principal`, somente essa entidade principal recebe permissões de invocação de função, mas apenas se ela também fizer parte organização `o-a1b2c3d4e5f`.

Esse comando cria uma política baseada em recursos semelhante ao exemplo a seguir:

```
{
    "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"
                }
            }
        }
    ]
}
```

Para mais informações, consulte aws:PrincipalOrgID no guia do usuário do AWS Identity and Access Management.

Conceder acesso de função a outras contas

Para conceder permissões a outra conta da AWS, especifique o ID da conta como o `principal`. O exemplo a seguir concede permissão à conta `111122223333` para invocar `my-function` com o alias `prod`.

```
aws lambda add-permission --function-name my-function:prod --statement-id xaccount --
action lambda:InvokeFunction \
    --principal 111122223333 --output text
```

A seguinte saída deverá ser mostrada:

```
{"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:l
east-2:123456789012:function:my-function"}
```

A política baseada em recursos concede permissão para a outra conta acessar a função, mas não permite que os usuários nessa conta excedam suas permissões. Os usuários na outra conta devem ter as permissões de usuário correspondentes para usar a API do Lambda.

Para limitar o acesso a um usuário ou função em outra conta, especifique o ARN completo da identidade como o principal. Por exemplo, `arn:aws:iam::123456789012:user/developer`.

O alias limita qual versão a outra conta pode chamar. Ele exige que a outra conta inclua o alias no ARN da função.

```
aws lambda invoke --function-name arn:aws:lambda:us-west-2:123456789012:function:my-
function:prod out
```

A seguinte saída deverá ser mostrada:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "1"
}
```

Depois disso, o proprietário da função pode atualizar o alias para apontar para uma nova versão sem que o chamador precise alterar a maneira como eles invocam sua função. Isso garante que a outra conta não precise alterar o código para usar a nova versão, e ela tem permissão somente para invocar a versão da função associada ao alias.

Conceda acesso entre contas para a maioria das ações de API que operam em uma função existente. Por exemplo, é possível conceder acesso a `lambda:ListAliases` para obter uma lista de aliases ou a `lambda:GetFunction` para permitir que eles façam download do código da função.

Adicione cada permissão separadamente ou use `lambda:*` para conceder acesso a todas as ações da função especificada.

Para conceder permissão a outras contas para várias funções ou para ações que não operem em uma função, recomendamos usar as funções do IAM.

Conceder acesso de camada a outras contas

Para conceder permissão de uso de camadas a outra conta, adicione uma instrução à política de permissões da versão da camada usando o comando add-layer-version-permission. Em cada instrução, você pode conceder permissão a uma única conta, a todas as contas ou a uma organização.

O exemplo a seguir concede à conta 111122223333 acesso à versão 2 da camada `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
```

Você deve ver saída semelhante a:

```
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"}
```

As permissões se aplicam apenas a uma única versão de camada. Repita o processo sempre que criar uma nova versão da camada.

Para conceder permissão a todas as contas em uma organização, use a opção `organization-id`. O exemplo a seguir concede a todas as contas em uma organização permissão para usar a versão 3 de uma camada.

```
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
```

A seguinte saída deverá ser mostrada:

```
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"}}}"
```

Para conceder permissão a todas as contas da AWS, use * para a entidade principal e omita o ID da organização. Para várias contas ou organizações, é necessário adicionar várias instruções.

Limpar políticas baseadas em recursos

Para exibir a política baseada em recursos de uma função, use o comando `get-policy`.

```
aws lambda get-policy --function-name my-function --output text
```

A seguinte saída deverá ser mostrada:

```
{"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
```

Para versões e aliases, acrescente o número da versão ou o alias ao nome da função.

```
aws lambda get-policy --function-name my-function:PROD
```

Para remover permissões da função, use `remove-permission`.

```
aws lambda remove-permission --function-name example --statement-id sns
```

Use o comando `get-layer-version-policy` para visualizar as permissões em uma camada.

```
aws lambda get-layer-version-policy --layer-name my-layer --version-number 3 --output
 text
```

A seguinte saída deverá ser mostrada:

```
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"}}}"
```

Use `remove-layer-version-permission` para remover instruções da política.

```
aws lambda remove-layer-version-permission --layer-name my-layer --version-number 3 --
statement-id engineering-org
```

Usar controle de acesso baseado em atributos no Lambda

Com o controle de acesso por atributo (ABAC), você pode usar etiquetas para controlar o acesso às suas funções do Lambda. Você pode anexar etiquetas a uma função do Lambda, transmiti-las em determinadas solicitações de API ou anexá-las à entidade principal do AWS Identity and Access Management (IAM) que está fazendo a solicitação. Para obter mais informações sobre como o AWS concede acesso baseado em atributos, consulte Controlar o acesso aos recursos da AWS usando etiquetas, no Guia do usuário do IAM.

Você pode usar o ABAC para conceder privilégio mínimo sem especificar um nome do recurso da Amazon (ARN) ou padrão de ARN na política do IAM. Em vez disso, você pode especificar uma etiqueta no elemento de condição de uma política do IAM para controlar o acesso. A escalabilidade é mais fácil com o ABAC, pois você não precisa atualizar suas políticas do IAM ao criar novas funções. Em vez disso, adicione etiquetas às novas funções para controlar o acesso.

No Lambda, etiquetas funcionam no nível da função. Etiquetas não têm suporte em camadas, configurações de assinatura de código ou mapeamentos de fontes de eventos. Quando você marca uma função, essas etiquetas se aplicam a todas as versões e aliases associados a ela. Para obter informações sobre como marcar funções, consulte Utilizar etiquetas em funções do Lambda.

É possível usar as seguintes chaves de condição para controlar ações de funções:

- aws:ResourceTag/tag-key: controlar o acesso de acordo com as etiquetas anexadas a funções do Lambda.
- aws:RequestTag/tag-key: exigir que etiquetas estejam presentes em uma solicitação, como ao criar uma nova função.
- aws:PrincipalTag/tag-key: controle o que a entidade principal do IAM (a pessoa que está fazendo a solicitação) tem permissão para fazer com base nas etiquetas anexadas ao seu respectivo usuário ou perfil do IAM.
- aws:TagKeys: controle se chaves de etiquetas específicas podem ser usadas em uma solicitação.

Para obter uma lista completa de ações do Lambda que oferecem suporte ao ABAC, consulte Ações de função compatíveis e verifique a coluna Condition (Condição) na tabela.

As etapas a seguir demonstram uma das maneiras de configurar permissões usando o ABAC. Neste cenário de exemplo, você criará quatro políticas de permissões do IAM. Em seguida, anexará essas políticas a um novo perfil do IAM. Por último, você criará um usuário do IAM e concederá a ele permissão para assumir a nova função.

Tópicos

Pré-requisitos

Verifique se você tem um perfil de execução do Lambda. Você usará esse perfil ao conceder permissões do IAM e ao criar uma função do Lambda.

Etapa 1: exigir etiquetas em novas funções

Ao usar o ABAC com o Lambda, é uma prática recomendada exigir que todas as funções tenham etiquetas. Isso ajuda a garantir que suas políticas de permissões de ABAC funcionem conforme o esperado.

Crie uma política do IAM semelhante ao exemplo a seguir. Esta política usa as chaves de condição aws:RequestTag/tag-key, aws:ResourceTag/tag-key e aws:TagKeys para exigir que novas funções e a entidade principal do IAM que cria essas funções tenham a tag `project`. O modificador `ForAllValues` garante que `project` seja a única etiqueta permitida. Se você não incluir o modificador `ForAllValues`, os usuários poderão adicionar outras etiquetas à função, desde que também transmitam `project`.

Example – Exigir etiquetas em novas funções

```
{
  "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"
      }
    }
  }
}
```

Etapa 2: permitir ações com base em etiquetas anexadas a uma função do Lambda e a uma entidade principal do IAM

Crie uma segunda política do IAM usando a chave de condição aws:ResourceTag/tag-key para exigir que a etiqueta da entidade principal corresponda à etiqueta anexada à função. O exemplo de política a seguir permite que entidades principais com a etiqueta `project` invoquem funções do com a etiqueta `project`. Se uma função tiver outras etiquetas, a ação será negada.

Example – Exigir etiquetas correspondentes na função e na entidade principal do 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}"
        }
      }
    }
  ]
}
```

Etapa 3: conceder permissões de lista

Crie uma política que permita que a entidade principal liste funções do Lambda e perfis do IAM. Isso permite que a entidade principal veja todas as funções do Lambda e perfis do IAM no console e ao chamar as ações de API.

Example – Conceder permissões de lista do Lambda e do IAM

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllResourcesLambdaNoTags",
      "Effect": "Allow",
      "Action": [
        "lambda:GetAccountSettings",
        "lambda:ListFunctions",
        "iam:ListRoles"
      ],
      "Resource": "*"
    }
  ]
}
```

Etapa 4: conceder permissões do IAM

Crie uma política que permita iam:PassRole. Essa permissão é necessária quando você atribui um perfil de execução a uma função. No exemplo de política a seguir, substitua o ARN de exemplo pelo ARN do seu perfil de execução do Lambda.

> **Note**
>
> Não use a chave de condição `ResourceTag` em uma política com a ação `iam:PassRole`. Você não pode usar a etiqueta em uma função do IAM para controlar o acesso de quem pode transmitir essa função. Para obter mais informações sobre as permissões necessárias para transmitir uma função a um serviço, consulte Conceder permissões a um usuário para transmitir uma função a um serviço da AWS.

Example – Conceder permissão para transmitir a função de execução

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::111122223333:role/lambda-ex"
    }
  ]
}
```

Etapa 5: Criar o perfil do IAM

É uma prática recomendada usar funções para delegar permissões. Crie um perfil do IAM chamada `abac-project-role`:

- Em Step 1: Select trusted entity (Etapa 1: selecionar entidade confiável): escolha a conta da AWS e depois This account (Esta conta).
- Em Step 2: Add permissions (Etapa 2: adicionar permissões): anexe as quatro políticas do IAM que você criou nas etapas anteriores.
- Em Step 3: Name, review, and create (Etapa 3: nomear, revisar e criar): escolha Add tag (Adicionar etiqueta). Em Chave, digite `project`. Não insira nada em Value (Valor).

Etapa 6: criar o usuário do IAM

Crie um usuário do IAM chamado `abac-test-user`. Na seção Set permissions (Definir permissões), escolha Attach existing policies directly (Anexar políticas existentes diretamente) e escolha Create policy (Criar política). Insira a seguinte definição de política. Substitua *111122223333* pelo ID da sua conta da AWS. Essa política permite que `abac-test-user` assuma `abac-project-role`.

Example – Permitir que o usuário do IAM assuma o perfil ABAC

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::111122223333:role/abac-project-role"
  }
}
```

Etapa 7: Testar as permissões

1. Faça login no console do AWS como `abac-test-user`. Para obter mais informações, consulte Fazer login como usuário do IAM.
2. Alterne para a função `abac-project-role`. Para obter mais informações, consulte Alternar para uma função (console).
3. Crie uma função do Lambda:
   - Em Permissions (Permissões), escolha Change default execution role (Alterar a função de execução padrão) e, em Execution role (Função de execução), escolha Use an existing role (Usar uma função existente). Escolha o mesmo perfil de execução que você usou em Etapa 4: conceder permissões do IAM.
   - Em Advanced settings (Configurações avançadas), escolha Enable tags (Habilitar etiquetas) e escolha Add new tag (Adicionar nova etiqueta). Em Chave, digite `project`. Não insira nada em Value (Valor).
4. Testar a função.
5. Crie uma segunda função do Lambda e adicione uma etiqueta diferente, como `environment`. Essa operação deve falhar, pois a política de ABAC que você criou em Etapa 1: exigir etiquetas

em novas funções apenas permite que a entidade principal crie funções do com a etiqueta `project`.

6. Crie uma terceira função sem etiquetas. Essa operação deve falhar, pois a política de ABAC que você criou em Etapa 1: exigir etiquetas em novas funções não permite que a entidade principal crie funções sem etiquetas.

Essa estratégia de autorização permite controlar o acesso sem criar novas políticas para cada novo usuário. Para conceder acesso a novos usuários, basta dar a eles permissão para assumir o perfil que corresponde ao projeto atribuído.

Etapa 8: limpar os recursos

Para excluir a função do IAM

1. Abra a página Roles (Funções) no console do IAM.
2. Selecione o perfil criado na etapa 5.
3. Escolha Excluir.
4. Para confirmar a exclusão, insira o nome do perfil no campo de entrada de texto.
5. Escolha Excluir.

Para excluir o usuário do IAM

1. Abra a página Usuários no console do IAM.
2. Selecione o usuário do IAM que você criou na etapa 6.
3. Escolha Excluir.
4. Para confirmar a exclusão, insira o nome do usuário no campo de entrada de texto.
5. Escolha Excluir usuário.

Como excluir a função do Lambda

1. Abra a página Functions (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Ações, Excluir.
4. Digite **`delete`** no campo de entrada de texto e escolha Delete (Excluir).

Ajustar as seções de Recursos e Condições das políticas

Você pode restringir o escopo das permissões de um usuário especificando recursos e condições em uma política do AWS Identity and Access Management (IAM). Cada ação de API oferece suporte a uma combinação de tipos de condição e recurso que varia de acordo com o comportamento da ação.

Cada instrução de política do IAM concede permissão a uma ação realizada em um recurso. Quando a ação não atua em um recurso indicado, ou quando você concede permissão para executar a ação em todos os recursos, o valor do recurso na política é um curinga (*). Para muitas ações, restrinja os recursos que um usuário pode modificar especificando o nome do recurso da Amazon (ARN) de um recurso ou um padrão de ARN correspondente a vários recursos.

Para restringir as permissões por recurso especifique o recurso por ARN.

Formato do ARN de recurso Lambda

- Função: `arn:aws:lambda:us-west-2:123456789012:function:my-function`
- Versão da função: `arn:aws:lambda:us-west-2:123456789012:function:my-function:1`
- Alias da função: `arn:aws:lambda:us-west-2:123456789012:function:my-function:TEST`
- Mapeamento da fonte do evento: `arn:aws:lambda:us-west-2:123456789012:event-source-mapping:fa123456-14a1-4fd2-9fec-83de64ad683de6d47`
- Camada: `arn:aws:lambda:us-west-2:123456789012:layer:my-layer`
- Versão da camada: `arn:aws:lambda:us-west-2:123456789012:layer:my-layer:1`

Por exemplo, a política a seguir permite que um usuário na Conta da AWS `123456789012` invoque uma função denominada `my-function` na região da AWS Oeste dos EUA (Oregon).

Example invocar política de função

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Invoke",
            "Effect": "Allow",
```

```
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-function"
        }
    ]
}
```

Trata-se de um caso especial em que o identificador da ação (`lambda:InvokeFunction`) é diferente da operação da API (Invoke). Para outras ações, o identificador da ação é o nome da operação com o prefixo `lambda:`.

Seções

- Entender a seção Condição nas políticas
- Referência a funções na seção Recursos das políticas
- Ações de função compatíveis
- Ações de mapeamento de origem de eventos compatíveis
- Ações de camadas compatíveis

Entender a seção Condição nas políticas

As condições são um elemento opcional da política que aplica lógica adicional para determinar se uma ação é permitida. Além de condições comuns compatíveis com todas as ações, o Lambda define os tipos de condição que você pode usar para restringir os valores de parâmetros adicionais em algumas ações.

Por exemplo, a condição `lambda:Principal` permite restringir o serviço ou a conta, para que um usuário possa conceder acesso de invocação em uma política baseada em recursos de uma função. A política a seguir permite que um usuário conceda permissão para que tópicos do Amazon Simple Notification Service (Amazon SNS) invoquem uma função chamada `test`.

Example gerenciar permissões de política de função

```
{
    "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"
                }
            }
        }
    ]
}
```

A condição requer que o principal seja o Amazon SNS e não outro serviço ou outra conta. O padrão do recurso exige que o nome da função seja `test` e inclua um número de versão ou alias. Por exemplo, `test:v1`.

Para obter mais informações sobre recursos e condições do Lambda e outros serviços da AWS, consulte Ações, recursos e chaves de condição para serviços da AWS. na Referência a autorizações de serviços.

Referência a funções na seção Recursos das políticas

Você referencia uma função Lambda em uma instrução de política usando um nome do recurso da Amazon (ARN). O formato de um ARN de função depende se você estiver referenciando toda a função (não qualificado), a versão de uma função ou um alias (qualificado).

Ao fazer chamadas de API do Lambda, os usuários podem especificar uma versão ou alias passando um ARN de versão ou um ARN de alias no parâmtro GetFunction do `FunctionName` ou definindo um valor no parâmetro GetFunction do `Qualifier`. O Lambda toma decisões de autorização comparando o elemento de recurso na política do IAM com o `FunctionName` e `Qualifier` transmitido nas chamadas de API. Se houver incompatibilidade, o Lambda negará a solicitação.

Para permitir ou negar uma ação em sua função, é necessário usar os tipos de ARN de função corretos na declaração de política para obter os resultados esperados. Por exemplo, se sua política referenciar o ARN não qualificado, o Lambda aceitará solicitações que referenciam o ARN não qualificado, mas negará solicitações que referenciam um ARN qualificado.

> **Note**
>
> Você não pode usar um caractere curinga (*) para encontrar o valor correspondente ao ID da conta. Para obter mais informações sobre a sintaxe aceita, consulte Referência a políticas JSON do IAM, no Guia do usuário do IAM.

Example permitir invocação de um ARN não qualificado

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction"
        }
    ]
}
```

Se sua política referenciar um ARN qualificado específico, o Lambda aceitará solicitações que referenciam esse ARN, mas negará solicitações que referenciam o ARN não qualificado ou outro ARN qualificado, por exemplo, `myFunction:2`.

Example permitir invocação de um ARN qualificado específico

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction:1"
        }
    ]
}
```

Se sua política referenciar qualquer ARN qualificado usando `:*`, o Lambda aceitará qualquer ARN qualificado, mas negará solicitações que referenciem o ARN não qualificado.

Example permitir invocação de qualquer ARN qualificado

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction:*"
        }
    ]
}
```

Se sua política referenciar qualquer ARN usando *, o Lambda aceitará qualquer ARN qualificado ou não qualificado.

Example permitir invocação de qualquer ARN qualificado ou não qualificado

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction*"
        }
    ]
}
```

Ações de função compatíveis

As ações que operam em uma função podem ser restritas a uma função específica pelo ARN da função, da versão ou do alias, conforme descrito na tabela a seguir. Ações que não oferecem suporte a restrições de recursos apenas podem ser concedidas a todos os recursos (*).

Ações de função

Ação	Recurso	Condição
AddPermission	Função	lambda:Principal
RemovePermission		aws:ResourceTag/${TagKey}

Ação	Recurso	Condição
	Versão da função Alias da função	`lambda:FunctionUrlAuthType`

Invoke
 	
Permissão: `lambda:InvokeFunction`
	Função
	
	Versão da função
	
	Alias da função
		`aws:ResourceTag/${TagKey}`
		
		`lambda:EventSourceToken`

CreateFunction	Função	`lambda:CodeSigningConfigArn`
		
		`lambda:Layer`
		
		`lambda:VpcIds`
		
		`lambda:SubnetIds`
		
		`lambda:SecurityGroupIds`
		
		`aws:ResourceTag/${TagKey}`
		
		`aws:RequestTag/${TagKey}`
		
		`aws:TagKeys`

Ação	Recurso	Condição

UpdateFunctionConfiguration	Função	`lambda:CodeSigningConfigArn`
		`lambda:Layer`
		`lambda:VpcIds`
		`lambda:SubnetIds`
		`lambda:SecurityGroupIds`
		`aws:ResourceTag/${TagKey}`

Ação	Recurso	Condição
CreateAlias	Função	`aws:ResourceTag/${TagKey}`
DeleteAlias		
DeleteFunction		
DeleteFunctionCodeSigningConfig		
DeleteFunctionConcurrency		
GetAlias		
GetFunction		
GetFunctionCodeSigningConfig		
GetFunctionConcurrency		
GetFunctionConfiguration		
GetPolicy		
ListProvisionedConcurrencyConfigs		
ListAliases		
ListTags		
ListVersionsByFunction		
PublishVersion		
PutFunctionCodeSigningConfig		
PutFunctionConcurrency		
UpdateAlias		
UpdateFunctionCode		

Ação	Recurso	Condição
CreateFunctionUrlConfig DeleteFunctionUrlConfig GetFunctionUrlConfig UpdateFunctionUrlConfig	Função Alias da função	lambda:FunctionUrlAuthType lambda:FunctionArn aws:ResourceTag/${TagKey}
ListFunctionUrlConfigs	Função	lambda:FunctionUrlAuthType
DeleteFunctionEventInvokeConfig GetFunctionEventInvokeConfig ListFunctionEventInvokeConfigs PutFunctionEventInvokeConfig UpdateFunctionEventInvokeConfig	Função	aws:ResourceTag/${TagKey}
DeleteProvisionedConcurrencyConfig GetProvisionedConcurrencyConfig PutProvisionedConcurrencyConfig	Alias da função Versão da função	aws:ResourceTag/${TagKey}
GetAccountSettings ListFunctions	*	Nenhum
TagResource	Função	aws:ResourceTag/${TagKey} aws:RequestTag/${TagKey} aws:TagKeys
UntagResource	Função	aws:ResourceTag/${TagKey} aws:TagKeys

Ações de mapeamento de origem de eventos compatíveis

Para mapeamentos de origens de eventos, você pode restringir permissões para excluir e atualizar a uma origem de evento específica. A condição `lambda:FunctionArn` permite restringir quais funções um usuário pode configurar uma fonte de evento para invocar.

Para essas ações, o recurso é o mapeamento da fonte do evento, portanto, o Lambda fornece uma condição que permite restringir permissões com base na função que o mapeamento da fonte do evento invoca.

Ações de mapeamento da fonte de eventos

Ação	Recurso	Condição
DeleteEventSourceMapping UpdateEventSourceMapping	Mapeamento de origem do evento	`lambda:FunctionArn`
CreateEventSourceMapping GetEventSourceMapping	*	`lambda:FunctionArn`
ListEventSourceMappings	*	Nenhum

Ações de camadas compatíveis

Ações de camadas permitem restringir as camadas que um usuário pode gerenciar ou usar com uma função. Ações relacionadas a permissões e uso de camadas atuam em uma versão de uma camada, enquanto `PublishLayerVersion` atua no nome de uma camada. Use ambos com curingas para restringir as camadas com as quais um usuário pode trabalhar por nome.

> **Note**
>
> A ação GetLayerVersion também abrange GetLayerVersionByArn. O Lambda não oferece suporte para`GetLayerVersionByArn`como uma ação do IAM.

Ações de camada

Ação	Recurso	Condição
AddLayerVersionPermission RemoveLayerVersionPermission GetLayerVersion GetLayerVersionPolicy DeleteLayerVersion	Versão da camada	Nenhum
ListLayerVersions PublishLayerVersion	Camada	Nenhum
ListLayers	*	Nenhum

Segurança no AWS Lambda

A segurança na nuvem na AWS é a nossa maior prioridade. Como cliente da AWS, você contará com um datacenter e uma arquitetura de rede criados para atender aos requisitos das organizações com as maiores exigências de segurança.

A segurança é uma responsabilidade compartilhada entre a AWS e você. O modelo de responsabilidade compartilhada descreve isto como segurança da nuvem e segurança na nuvem:

- Segurança da nuvem: a AWS é responsável pela proteção da infraestrutura que executa produtos da AWS na Nuvem AWS. A AWS também fornece serviços que podem ser usados com segurança. Auditores de terceiros testam e verificam regularmente a eficácia da nossa segurança como parte dos compliance programsAWS. Para saber mais sobre os programas de conformidade que se aplicam ao AWS Lambda, consulte AWS Services in Scope by Compliance Program.
- Segurança na nuvem: sua responsabilidade é determinada pelo serviço da AWS que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da empresa e as leis e regulamentos aplicáveis.

Esta documentação ajuda a entender como aplicar o modelo de responsabilidade compartilhada ao usar o Lambda. Os tópicos a seguir mostram como configurar o Lambda para atender aos seus objetivos de segurança e conformidade. Saiba também como usar outros serviços da AWS que ajudam a monitorar e proteger os recursos do Lambda.

Para obter mais informações sobre como colocar os princípios de segurança em prática para aplicações do Lambda, consulte Security no Serverless Land.

Tópicos

Proteção de dados no AWS Lambda

O modelo de responsabilidade compartilhada da AWS se aplica à proteção de dados no AWS Lambda. Conforme descrito nesse modelo, a AWS é responsável por proteger a infraestrutura global que executa toda a Nuvem AWS. Você é responsável por manter o controle sobre seu conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para obter mais informações sobre a privacidade de dados, consulte as Perguntas frequentes sobre privacidade de dados. Para mais informações sobre a proteção de dados na Europa, consulte o artigo AWS Shared Responsibility Model and GDPR no Blog de segurança da AWS.

Para fins de proteção de dados, recomendamos que você proteja as credenciais da Conta da AWS e configure as contas de usuário individuais com o AWS IAM Identity Center ou o AWS Identity and Access Management (IAM). Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com os recursos da AWS. Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure o registro em log das atividades da API e do usuário com o AWS CloudTrail.
- Use as soluções de criptografia da AWS, juntamente com todos os controles de segurança padrão dos Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sigilosos armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-2 ao acessar a AWS por meio de uma interface de linha de comando ou uma API, use um endpoint do FIPS. Para ter mais informações sobre endpoints do FIPS, consulte Federal Information Processing Standard (FIPS) 140-2.

É altamente recomendável que nunca sejam colocadas informações de identificação confidenciais, como endereços de e-mail dos seus clientes, em marcações ou campos de formato livre, como um campo Nome. Isso inclui usar o console, a API, a AWS CLI ou os SDKs da AWS ao trabalhar com o Lambda ou outros Serviços da AWS. Quaisquer dados inseridos em tags ou campos de texto de formato livre usados para nomes podem ser usados para logs de faturamento ou de diagnóstico. Se

você fornecer um URL para um servidor externo, recomendamos fortemente que não sejam incluídas informações de credenciais no URL para validar a solicitação a esse servidor.

Seções

Criptografia em trânsito

Os endpoints da API do Lambda oferecem suporte a conexões seguras somente em HTTPS. Ao gerenciar recursos do Lambda com o AWS Management Console, o AWS SDK ou a API do Lambda, toda a comunicação é criptografada com Transport Layer Security (TLS). Para obter uma lista completa de endpoints de API, consulte Regiões e endpoints da AWS no Referência geral da AWS.

Quando você conecta a função a um sistema de arquivos, o Lambda usa a Criptografia em trânsito para todas as conexões. Para obter mais informações, consulte Criptografia de dados no Amazon EFS no Manual do usuário do Amazon Elastic File System.

Quando você usavariáveis de ambiente, é possível habilitar os ajudantes de criptografia de console para usar a criptografia no lado do cliente para proteger as variáveis de ambiente em trânsito. Para ter mais informações, consulte Proteger variáveis de ambiente no Lambda.

Criptografia inativa

O Lambda sempre criptografa variáveis de ambiente em repouso. Por padrão, o Lambda usa um AWS KMS key que o Lambda cria em sua conta para criptografar suas variáveis de ambiente. Esse Chave gerenciada pela AWS é nomeado `aws/lambda`.

De acordo com cada função, você pode configurar o Lambda para usar a chave gerenciada pelo cliente em vez de Chave gerenciada pela AWS para criptografar suas variáveis de ambiente. Para ter mais informações, consulte Proteger variáveis de ambiente no Lambda.

O Lambda sempre criptografa os arquivos dos quais você faz upload no Lambda, incluindo pacotes de implantação e arquivos de camada.

O Amazon CloudWatch Logs e o AWS X-Ray também criptografam dados por padrão, e podem ser configurados para usarem uma chave gerenciada pelo cliente. Para obter mais detalhes, consulte Criptografar dados de log no CloudWatch Logs e Proteção de dados no AWS X-Ray.

Gerenciamento de identidade e acesso para AWS Lambda

O AWS Identity and Access Management (IAM) é um AWS service (Serviço da AWS) que ajuda a controlar o acesso aos recursos da AWS de forma segura. Os administradores do IAM controlam quem pode ser autenticado (conectado) e autorizado (ter permissões) para usar os recursos do Lambda. O IAM é um AWS service (Serviço da AWS) que pode ser usado sem custo adicional.

Tópicos

Público

O uso do AWS Identity and Access Management (IAM) varia dependendo do trabalho que é realizado no Lambda.

Usuário do serviço: se você usar o serviço Lambda para fazer o trabalho, o administrador fornecerá as credenciais e as permissões necessárias. À medida que usar mais recursos do Lambda para fazer seu trabalho, você poderá precisar de permissões adicionais. Entender como o acesso é gerenciado pode ajudá-lo a solicitar as permissões corretas ao seu administrador. Se não for possível acessar um recurso no Lambda, consulte Solução de problemas de identidade e acesso do AWS Lambda.

Administrador do serviço: se você for o responsável pelos recursos do Lambda na empresa, provavelmente terá acesso total ao Lambda. Cabe a você determinar quais funcionalidades e recursos do Lambda os usuários do serviço devem acessar. Assim, você deve enviar solicitações ao administrador do IAM para alterar as permissões dos usuários de seu serviço. Revise as informações nesta página para entender os Introdução ao IAM. Para saber mais sobre como a empresa pode usar o IAM com o Lambda, consulte Como o AWS Lambda funciona com o IAM.

Administrador do IAM: se você é um administrador do IAM, talvez queira saber detalhes sobre como pode escrever políticas para gerenciar o acesso ao Lambda. Para visualizar exemplos de políticas baseadas em identidade do Lambda que podem ser usadas no IAM, consulte Exemplos de políticas baseadas em identidade para o AWS Lambda.

Autenticando com identidades

A autenticação é a forma como você faz login na AWS utilizando suas credenciais de identidade. É necessário ser autenticado (fazer login na AWS) como o usuário raiz da Usuário raiz da conta da AWS, como usuário do IAM ou assumindo um perfil do IAM.

É possível fazer login na AWS como uma identidade federada utilizando credenciais fornecidas por uma fonte de identidades. AWS IAM Identity Center Os usuários do IAM Identity Center, a autenticação única da empresa e as suas credenciais do Google ou do Facebook são exemplos de identidades federadas. Quando você faz login como uma identidade federada, o administrador já configurou anteriormente a federação de identidades utilizando perfis do IAM. Quando você acessa a AWS utilizando a federação, está indiretamente assumindo um perfil.

É possível fazer login no AWS Management Console ou no de acesso da AWS dependendo do tipo de usuário que você é. Para obter mais informações sobre como fazer login na AWS, consulte Conta da AWS Como fazer login na sua no Início de Sessão da AWS Guia do usuário.

Se você acessar a AWS programaticamente, a AWS fornecerá um kit de desenvolvimento de software (SDK) e uma interface da linha de comando (CLI) para você assinar criptograficamente as solicitações utilizando as suas credenciais. Se você não utilizar as ferramentas da AWS, deverá assinar as solicitações por conta própria. Para obter mais informações sobre como utilizar o método recomendado para assinar solicitações por conta própria, consulte Assinar solicitações de API da AWS no Guia do usuário do IAM.

Independentemente do método de autenticação usado, também pode ser exigido que você forneça mais informações de segurança. Por exemplo, a AWS recomenda o uso da autenticação multifator (MFA) para aumentar a segurança de sua conta. Para saber mais, consulte Autenticação multifator no Guia do usuário do AWS IAM Identity Center e Usar a autenticação multifator (MFA) naAWS no Guia do usuário do IAM.

Usuário raiz da Conta da AWS

Ao criar uma Conta da AWS, você começa com uma identidade de login que tem acesso completo a todos os atributos e Serviços da AWS na conta. Essa identidade, denominada usuário raiz da

Conta da AWS, e é acessada por login com o endereço de e-mail e a senha que você usou para criar a conta. É altamente recomendável não utilizar o usuário raiz para tarefas diárias. Proteja as credenciais do usuário raiz e use-as para executar as tarefas que somente ele puder executar. Para obter a lista completa das tarefas que exigem login como usuário raiz, consulte Tarefas que exigem credenciais de usuário raiz no Guia do usuário do IAM.

Identidade federada

Como prática recomendada, exija que os usuários, inclusive os que precisam de acesso de administrador, usem a federação com um provedor de identidades para acessar Serviços da AWS usando credenciais temporárias.

Identidade federada é um usuário de seu diretório de usuários corporativos, um provedor de identidades da web AWS Directory Service, o , o diretório do Centro de Identidade ou qualquer usuário que acesse os Serviços da AWS usando credenciais fornecidas por meio de uma fonte de identidade. Quando as identidades federadas acessam Contas da AWS, elas assumem perfis que fornecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, recomendamos usar o .AWS IAM Identity Center É possível criar usuários e grupos no Centro de Identidade do IAM ou se conectar e sincronizar com um conjunto de usuários e grupos em sua própria fonte de identidade para uso em todas as suas Contas da AWS e aplicações. Para obter mais informações sobre o IAM Identity Center, consulte “O que é o IAM Identity Center?” no Guia do usuário do AWS IAM Identity Center.

Usuários e grupos do IAM

Um usuário do IAM é uma identidade dentro da Conta da AWS que tem permissões específicas para uma única pessoa ou aplicação. Sempre que possível, recomendamos contar com credenciais temporárias em vez de criar usuários do IAM com credenciais de longo prazo, como senhas e chaves de acesso. No entanto, se você tiver casos de utilização específicos que exijam credenciais de longo prazo com usuários do IAM, recomendamos alternar as chaves de acesso. Para obter mais informações, consulte Alterne as chaves de acesso regularmente para casos de utilização que exijam credenciais de longo prazo no Guia do usuário do IAM.

Um grupo do IAM é uma identidade que especifica uma coleção de usuários do IAM. Não é possível fazer login como um grupo. É possível utilizar grupos para especificar permissões para vários usuários de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por exemplo, é possível ter um grupo chamado IAMAdmins e atribuir a esse grupo permissões para administrar recursos do IAM.

Usuários são diferentes de perfis. Um usuário é exclusivamente associado a uma pessoa ou a uma aplicação, mas um perfil pode ser assumido por qualquer pessoa que precisar dela. Os usuários têm credenciais permanentes de longo prazo, mas os perfis fornecem credenciais temporárias. Para saber mais, consulte Quando criar um usuário do IAM (em vez de um perfil) no Guia do usuário do IAM.

Perfis do IAM

Um perfil do IAM é uma identidade dentro da Conta da AWS que tem permissões específicas. Ele é semelhante a um usuário do IAM, mas não está associado a uma pessoa específica. É possível assumir temporariamente um perfil do IAM no AWS Management Console alternando perfis. É possível assumir um perfil chamando uma operação de API da AWS CLI ou da AWS, ou utilizando um URL personalizado. Para obter mais informações sobre métodos para a utilização de perfis, consulte Usar perfis do IAM no Guia do usuário do IAM.

Perfis do IAM com credenciais temporárias são úteis nas seguintes situações:

- Acesso de usuário federado: para atribuir permissões a identidades federadas, é possível criar um perfil e definir permissões para ele. Quando uma identidade federada é autenticada, essa identidade é associada ao perfil e recebe as permissões definidas pelo mesmo. Para obter mais informações sobre perfis para federação, consulte Criar uma função para um provedor de identidade de terceiros no Guia do usuário do IAM. Se você utilizar o IAM Identity Center, configure um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o Centro de identidade do IAM correlaciona o conjunto de permissões a um perfil no IAM. Para obter informações sobre conjuntos de permissões, consulte Conjuntos de permissões no Guia do usuário do AWS IAM Identity Center.
- Permissões temporárias para usuários do IAM: um usuário ou um perfil do IAM pode assumir um perfil do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- Acesso entre contas: é possível utilizar um perfil do IAM para permitir que alguém (um principal confiável) em outra conta acesse recursos na sua conta. Os perfis são a principal forma de conceder acesso entre contas. No entanto, alguns Serviços da AWS permitem anexar uma política diretamente a um recurso (em vez de utilizar um perfil como proxy). Para saber a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Como os perfis do IAM diferem das políticas baseadas em recurso no Guia do usuário do IAM.
- Acesso entre serviços: alguns Serviços da AWS usam recursos em outros Serviços da AWS. Por exemplo, quando você faz uma chamada em um serviço, é comum que esse serviço execute aplicações no Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso

utilizando as permissões do principal de chamada, utilizando um perfil de serviço ou um perfil vinculado a serviço.

- Encaminhamento de sessões de acesso (FAS): qualquer pessoa que utilizar uma função ou usuário do IAM para realizar ações na AWS é considerada uma entidade principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O recurso FAS utiliza as permissões da entidade principal que chama um AWS service (Serviço da AWS), combinadas às permissões do AWS service (Serviço da AWS) solicitante, para realizar solicitações para serviços downstream. As solicitações de FAS só são feitas quando um serviço recebe uma solicitação que exige interações com outros Serviços da AWS ou com recursos para serem concluídas. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte Encaminhar sessões de acesso.
- Perfil de serviço: um perfil de serviço é um Perfil do IAM que um serviço assume para executar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do usuário do IAM.
- Função vinculada a serviço: um perfil vinculado a serviço é um tipo de perfil de serviço vinculada a um AWS service (Serviço da AWS). O serviço pode assumir o perfil para executar uma ação em seu nome. Os perfis vinculados ao serviço aparecem em sua Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não pode editar as permissões para perfis vinculados a serviço.

- Aplicações em execução no Amazon EC2: é possível usar um perfil do IAM para gerenciar credenciais temporárias para aplicações em execução em uma instância do EC2 e fazer solicitações da AWS CLI ou da AWS API. É preferível fazer isso a armazenar chaves de acesso na instância do EC2. Para atribuir um perfil da AWS a uma instância do EC2 e disponibilizá-la para todas as suas aplicações, crie um perfil de instância que seja anexado a ela. Um perfil de instância contém o perfil e permite que os programas em execução na instância do EC2 obtenham credenciais temporárias. Para obter mais informações, consulte Usar uma função do IAM para conceder permissões a aplicações em execução nas instâncias do Amazon EC2 no Guia do usuário do IAM.

Para saber se deseja usar as funções do IAM, consulte When to create an IAM role (instead of a user) [Quando criar um perfil do IAM (em vez de um usuário)] no Guia do usuário do IAM.

Gerenciamento do acesso utilizando políticas

Você controla o acesso na AWS criando políticas e anexando-as a identidades ou recursos da AWS. Uma política é um objeto na AWS que, quando associado a uma identidade ou recurso, define suas permissões. A AWS avalia essas políticas quando uma entidade principal (usuário, usuário raiz ou sessão de perfil) faz uma solicitação. As permissões nas políticas determinam se a solicitação será permitida ou negada. A maioria das políticas são armazenadas na AWS como documentos JSON. Para obter mais informações sobre a estrutura e o conteúdo de documentos de políticas JSON, consulte Visão geral das políticas JSON no Guia do usuário do IAM.

Os administradores podem usar AWS as políticas JSON da para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

Por padrão, usuários e funções não têm permissões. Para conceder aos usuários permissão para executar ações nos recursos de que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM a perfis, e os usuários podem assumir os perfis.

As políticas do IAM definem permissões para uma ação independente do método usado para executar a operação. Por exemplo, suponha que você tenha uma política que permite a ação `iam:GetRole`. Um usuário com essa política pode obter informações de perfis do AWS Management Console, da AWS CLI ou da API da AWS.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário, grupo de usuários ou perfil do IAM. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Criar políticas do IAM no Guia do usuário do IAM.

As políticas baseadas em identidade podem ser categorizadas ainda mais como políticas embutidas ou políticas gerenciadas. As políticas em linha são anexadas diretamente a um único usuário, grupo ou perfil. As políticas gerenciadas são políticas independentes que podem ser anexadas a vários usuários, grupos e perfis na Conta da AWS. As políticas gerenciadas incluem políticas gerenciadas pela AWS e políticas gerenciadas pelo cliente. Para saber como escolher entre uma política gerenciada ou uma política em linha, consulte Escolher entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

Políticas baseadas em recurso

Políticas baseadas em recurso são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o recurso ao qual a política está anexada, a política define quais ações uma entidade principal especificada pode executar nesse recurso e em que condições. Você deve especificar uma entidade principal em uma política baseada em recursos. As entidades principais podem incluir contas, usuários, perfis, usuários federados ou Serviços da AWS.

Políticas baseadas em recursos são políticas em linha que estão localizadas nesse serviço. Não é possível utilizar as políticas gerenciadas da AWS do IAM em uma política baseada em recursos.

Listas de controle de acesso (ACLs)

As listas de controle de acesso (ACLs) controlam quais entidades principais (membros, usuários ou funções da conta) têm permissões para acessar um recurso. As ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

Amazon S3, AWS WAF e Amazon VPC são exemplos de serviços que são compatíveis com a ACLs. Para saber mais sobre ACLs, consulte Visão geral da lista de controle de acesso (ACL) no Guia do desenvolvedor do Amazon Simple Storage Service.

Outros tipos de política

A AWS aceita tipos de política menos comuns. Esses tipos de política podem definir o máximo de permissões concedidas a você pelos tipos de política mais comuns.

- Limites de permissões: um limite de permissões é um recurso avançado no qual você define o máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM (usuário ou perfil do IAM). É possível definir um limite de permissões para uma entidade. As permissões resultantes são a interseção das políticas baseadas em identidade de uma entidade com seus limites de permissões. As políticas baseadas em recurso que especificam o usuário ou o perfil no campo `Principal` não são limitadas pelo limite de permissões. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações sobre limites de permissões, consulte Limites de permissões para identidades do IAM no Guia do usuário do IAM.

- Políticas de controle de serviço (SCPs): SCPs são políticas JSON que especificam as permissões máximas para uma organização ou unidade organizacional (UO) no AWS Organizations. O AWS Organizations é um serviço para agrupar e gerenciar centralmente várias Contas da AWS pertencentes à sua empresa. Se você habilitar todos os atributos em uma organização, poderá aplicar políticas de controle de serviço (SCPs) a qualquer uma ou a todas as contas. O SCP limita as permissões para entidades em contas-membro, incluindo cada .Usuário raiz da conta da AWS Para obter mais informações sobre o Organizações e SCPs, consulte How SCPs work (Como os SCPs funcionam) noAWS Organizations Guia do usuário do .
- Políticas de sessão: são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para um perfil ou um usuário federado. As permissões da sessão resultante são a interseção das políticas baseadas em identidade do usuário ou do perfil e das políticas de sessão. As permissões também podem ser provenientes de uma política baseada em recurso. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações, consulte Políticas de sessão no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como a AWS determina se deve permitir uma solicitação quando há vários tipos de política envolvidos, consulte Lógica da avaliaçãode políticas no Guia do usuário do IAM.

Como o AWS Lambda funciona com o IAM

Antes de usar o IAM para gerenciar o acesso ao Lambda, saiba quais recursos do IAM estão disponíveis para uso com o Lambda.

atributos do IAM que você pode usar com o AWS Lambda

atributo do IAM	Suporte para Lambda
Políticas baseadas em identidade	Sim
Políticas baseadas em atributos	Sim
Ações de políticas	Sim

atributo do IAM	Suporte para Lambda
atributos de políticas	Sim
Chaves de condição de política (específicas do serviço)	Sim
ACLs	Não
ABAC (tags em políticas)	Parcial
Credenciais temporárias	Sim
Sessões de acesso direto (FAS)	Não
Perfis de serviço	Sim
Perfis vinculados ao serviço	Parcial

Para ter uma visão de alto nível de como o Lambda e AWS outros serviços funcionam com a maioria dos recursos do IAM, AWSconsulte os serviços que funcionam com o IAM no Guia do usuário do IAM.

Políticas baseadas em identidade para Lambda

Suporta políticas baseadas em identidade	Sim

As políticas baseadas em identidade são documentos de políticas de permissões JSON que é possível anexar a uma identidade, como usuário, grupo de usuários ou perfil do IAM. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Criar políticas do IAM no Guia do usuário do IAM.

Com as políticas baseadas em identidade do IAM, é possível especificar ações ou atributos permitidos ou negados, bem como as condições sob as quais as ações são permitidas ou negadas. Você não pode especificar a entidade principal em uma política baseada em identidade porque ela se aplica ao usuário ou perfil à qual ela está anexado. Para saber mais sobre todos os elementos

que podem ser usados em uma política JSON, consulte Referência de elementos da política JSON do IAM no Guia do Usuário do IAM.

Exemplos de políticas baseadas em identidade para Lambda

Para ver exemplos de políticas baseadas em identidade do Lambda, consulte. Exemplos de políticas baseadas em identidade para o AWS Lambda

Políticas baseadas em recursos no Lambda

Oferece suporte a políticas baseadas em atributos	Sim

Políticas baseadas em recurso são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o recurso ao qual a política está anexada, a política define quais ações uma entidade principal especificada pode executar nesse recurso e em que condições. Você deve especificar uma entidade principal em uma política baseada em recursos. As entidades principais podem incluir contas, usuários, perfis, usuários federados ou Serviços da AWS.

Para permitir o acesso entre contas, é possível especificar uma conta inteira ou as entidades do IAM em outra conta como a entidade principal em uma política baseada em recurso. Adicionar uma entidade principal entre contas à política baseada em recurso é apenas metade da tarefa de estabelecimento da relação de confiança. Quando a entidade principal e o recurso estão em diferentes Contas da AWS, um administrador do IAM da conta confiável também deve conceder à entidade principal (usuário ou perfil) permissão para acessar o recurso. Eles concedem permissão ao anexar uma política baseada em identidade para a entidade. No entanto, se uma política baseada em recurso conceder acesso a uma entidade principal na mesma conta, nenhuma outra política baseada em identidade será necessária. Para obter mais informações, consulte Como os perfis do IAM diferem de políticas baseadas em recursos no Guia do usuário do IAM.

Você pode anexar uma política baseada em recursos a uma função ou camada do Lambda. Essa política define quais diretores podem realizar ações na função ou na camada.

Para saber como anexar uma política baseada em recursos a uma função ou camada, consulte. Trabalhar com políticas baseadas em recursos no Lambda

Ações políticas para Lambda

Oferece suporte a ações de políticas	Sim

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos, e em que condições.

O elemento `Action` de uma política JSON descreve as ações que é possível utilizar para permitir ou negar acesso em uma política. As ações de política geralmente têm o mesmo nome que a operação de API da AWS associada. Existem algumas exceções, como ações somente de permissão, que não têm uma operação de API correspondente. Algumas operações também exigem várias ações em uma política. Essas ações adicionais são chamadas de ações dependentes.

Incluem ações em uma política para conceder permissões para executar a operação associada.

Para ver uma lista de ações do Lambda, consulte Ações definidas por AWS Lambda na Referência de Autorização de Serviço.

As ações de política no Lambda usam o seguinte prefixo antes da ação:

```
lambda
```

Para especificar várias ações em uma única instrução, separe-as com vírgulas.

```
"Action": [
      "lambda:action1",
      "lambda:action2"
         ]
```

Para ver exemplos de políticas baseadas em identidade do Lambda, consulte. Exemplos de políticas baseadas em identidade para o AWS Lambda

Recursos de políticas para Lambda

Oferece suporte a atributos de políticas	Sim

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos, e em que condições.

O elemento de política `Resource` JSON especifica o objeto ou os objetos aos quais a ação se aplica. As instruções devem incluir um elemento `Resource` ou um elemento `NotResource`. Como prática recomendada, especifique um recurso utilizando seu Nome do recurso da Amazon (ARN). Isso pode ser feito para ações que são compatíveis com a um tipo de recurso específico, conhecido como permissões em nível de recurso.

Para ações que não oferecem suporte a permissões em nível de recurso, como operações de listagem, use um caractere curinga (*) para indicar que a instrução se aplica a todos os recursos.

```
"Resource": "*"
```

Para ver uma lista dos tipos de recursos do Lambda e seus ARNs, consulte Tipos de recursos definidos por AWS Lambda na Referência de Autorização de Serviço. Para saber com quais ações é possível especificar o ARN de cada atributo, consulte Ações definidas pelo AWS Lambda.

Para ver exemplos de políticas baseadas em identidade do Lambda, consulte. Exemplos de políticas baseadas em identidade para o AWS Lambda

Chaves de condição de política para Lambda

Suporta chaves de condição de política específicas de serviço	Sim

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Condition` (ou `Condition` bloco de) permite que você especifique condições nas quais uma instrução está em vigor. O elemento `Condition` é opcional. É possível criar expressões condicionais que usam atendentes de condição, como “igual a” ou “menor que”, para fazer a condição da política corresponder aos valores na solicitação.

Se você especificar vários elementos `Condition` em uma instrução ou várias chaves em um único elemento `Condition`, a AWS os avaliará utilizando uma operação lógica `AND`. Se você especificar vários valores para uma única chave de condição, a AWS avaliará a condição utilizando uma

operação lógica `OR`. Todas as condições devem ser atendidas antes que as permissões da instrução sejam concedidas.

Você também pode utilizar variáveis de espaço reservado ao especificar as condições. Por exemplo, é possível conceder a um usuário do IAM permissão para acessar um recurso somente se ele estiver marcado com seu nome de usuário do IAM. Para obter mais informações, consulte Elementos de política do IAM: variáveis e tags no Guia do usuário do IAM.

A AWS oferece suporte a chaves de condição globais e chaves de condição específicas do serviço. Para ver todas as AWS chaves de condição globais da , consulte AWSChaves de contexto de condição globais da no Guia do usuário do IAM.

Para ver uma lista das chaves de condição do Lambda, consulte Chaves de condição AWS Lambda na Referência de autorização de serviço. Para saber com quais ações e recursos é possível usar uma chave de condição, consulte Ações definidas pelo AWS Lambda.

Para ver exemplos de políticas baseadas em identidade do Lambda, consulte. Exemplos de políticas baseadas em identidade para o AWS Lambda

ACLs em Lambda

Oferece suporte a ACLs	Não

As listas de controle de acesso (ACLs) controlam quais entidades principais (membros, usuários ou perfis da conta) têm permissões para acessar um recurso. As ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

ABAC com Lambda

Oferece suporte a ABAC (tags em políticas)	Parcial

O controle de acesso baseado em atributo (ABAC) é uma estratégia de autorização que define permissões com base em atributos. Na AWS, esses atributos são chamados de tags. É possível anexar tags a entidades do IAM (usuários ou perfis) e a muitos recursos da AWS. A marcação de entidades e atributos é a primeira etapa do ABAC. Em seguida, você cria políticas de ABAC para permitir operações quando a tag da entidade principal corresponder à tag do atributo que ela está tentando acessar.

O ABAC é útil em ambientes que estão crescendo rapidamente e ajuda em situações onde o gerenciamento de políticas se torna um problema.

Para controlar o acesso baseado em tags, forneça informações sobre as tags no elemento de condição de uma política usando as `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou `aws:TagKeys` chaves de condição.

Se um serviço oferecer suporte às três chaves de condição para cada tipo de recurso, o valor será Sim para o serviço. Se um serviço oferecer suporte às três chaves de condição somente para alguns tipos de recursos, o valor será Parcial.

Para obter mais informações sobre o ABAC, consulte O que é ABAC? no Guia do usuário do IAM. Para visualizar um tutorial com etapas para configurar o ABAC, consulte Usar controle de acesso baseado em atributos (ABAC) no Guia do usuário do IAM.

Para obter mais informações sobre a marcação de recursos do Lambda, consulte. Usar controle de acesso baseado em atributos no Lambda

Usando credenciais temporárias com o Lambda

Oferece suporte a credenciais temporárias	Sim

Alguns Serviços da AWS não funcionam quando você faz login usando credenciais temporárias. Para obter informações adicionais, incluindo quais Serviços da AWS funcionam com credenciais temporárias, consulte Serviços da AWS que funcionam com o IAM no Guia do usuário do IAM.

Você está usando credenciais temporárias se faz login no AWS Management Console usando qualquer método, exceto um nome de usuário e uma senha. Por exemplo, quando você acessa a AWS usando o link de autenticação única (SSO) da sua empresa, esse processo cria automaticamente credenciais temporárias. Você também cria automaticamente credenciais temporárias quando faz login no console como usuário e, em seguida, alterna perfis. Para obter mais informações sobre como alternar perfis, consulte Alternar para um perfil (console) no Guia do usuário do IAM.

É possível criar credenciais temporárias manualmente usando a AWS CLI ou a API da AWS. Em seguida, você pode usar essas credenciais temporárias para acessar aAWS .AWS A recomenda que você gere credenciais temporárias dinamicamente em vez de usar chaves de acesso de longo prazo. Para obter mais informações, consulte Credenciais de segurança temporárias no IAM.

Sessões de acesso direto para Lambda

Suporte para o recurso Encaminhamento de sessões de acesso (FAS)	Não

Quando você usa um usuário ou um perfil do IAM para executar ações na AWS, você é considerado uma entidade principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O recurso FAS utiliza as permissões da entidade principal que chama um AWS service (Serviço da AWS), combinadas às permissões do AWS service (Serviço da AWS) solicitante, para realizar solicitações para serviços downstream. As solicitações de FAS só são feitas quando um serviço recebe uma solicitação que exige interações com outros Serviços da AWS ou com recursos para serem concluídas. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte Encaminhar sessões de acesso.

Funções de serviço para Lambda

Oferece suporte a perfis de serviço	Sim

O perfil de serviço é um perfil do IAM que um serviço assume para realizar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do usuário do IAM.

No Lambda, uma função de serviço é conhecida como função de execução.

> **Warning**
>
> Alterar as permissões de uma função de execução pode interromper a funcionalidade do Lambda.

Funções vinculadas a serviços para Lambda

Oferece suporte a perfis vinculados ao serviço	Parcial

Um perfil vinculado ao serviço é um tipo de perfil de serviço vinculado a um AWS service (Serviço da AWS). O serviço pode assumir o perfil de executar uma ação em seu nome. Os perfis vinculados ao serviço aparecem em sua Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não pode editar as permissões para perfis vinculados a serviço.

O Lambda não tem perfis vinculados a serviços, mas o Lambda@Edge tem. Para obter mais informações, consulte Funções vinculadas a serviços para Lambda @Edge no Amazon Developer Guide. CloudFront

Para obter detalhes sobre como criar ou gerenciar funções vinculadas a serviços, consulte Serviços do AWS que funcionam com o IAM. Encontre um serviço na tabela que inclua um `Yes` na coluna Função vinculada ao serviço. Escolha o link Sim para visualizar a documentação do perfil vinculado a esse serviço .

Exemplos de políticas baseadas em identidade para o AWS Lambda

Por padrão, os usuários e os perfis não têm permissões para criar ou modificar os recursos do Lambda. Eles também não podem executar tarefas usando o AWS Management Console, a AWS Command Line Interface (AWS CLI) ou a API AWS. Para conceder aos usuários permissão para executar ações nos recursos de que precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM aos perfis, e os usuários podem assumir os perfis.

Para saber como criar uma política baseada em identidade do IAM usando esses exemplos de documento de política JSON, consulte Criação de políticas do IAM no Guia do Usuário do IAM.

Para obter detalhes sobre ações e tipos de recursos definidos pelo Lambda, incluindo o formato dos ARNs para cada um dos tipos de recursos, consulte Ações, recursos e chaves de condição AWS Lambda na Referência de Autorização de Serviço.

Tópicos

- Práticas recomendadas de políticas
- Usar o console do Lambda
- Permitir que os usuários exibam as próprias permissões

Práticas recomendadas de políticas

As políticas baseadas em identidade determinam se alguém pode criar, acessar ou excluir recursos do Lambda da sua conta. Essas ações podem incorrer em custos para a Conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece com as políticas gerenciadas pela AWS e avance para as permissões de privilégio mínimo: para começar a conceder permissões a seus usuários e workloads, utilize as políticas gerenciadas pela AWS que concedem permissões para muitos casos de utilização comuns. Elas estão disponíveis na sua Conta da AWS. É recomendável que você reduza ainda mais as permissões definindo políticas gerenciadas pelo cliente da AWS específicas para seus casos de uso. Para obter mais informações, consulte Políticas gerenciadas pela AWS ou Políticas gerenciadas pela AWS para perfis de trabalho no Guia do usuário do IAM.
- Aplique permissões de privilégio mínimo: ao definir permissões com as políticas do IAM, conceda apenas as permissões necessárias para executar uma tarefa. Você faz isso definindo as ações que podem ser executadas em recursos específicos sob condições específicas, também conhecidas como permissões de privilégio mínimo. Para obter mais informações sobre como utilizar o IAM para aplicar permissões, consulte Políticas e permissões no IAM no Guia do usuário do IAM.
- Use condições nas políticas do IAM para restringir ainda mais o acesso: é possível adicionar uma condição às políticas para limitar o acesso a ações e recursos. Por exemplo, é possível escrever uma condição de política para especificar que todas as solicitações devem ser enviadas utilizando SSL. Você também pode utilizar condições para conceder acesso a ações de serviço, se elas forem usadas por meio de um AWS service (Serviço da AWS) específico, como o AWS CloudFormation. Para obter mais informações, consulte Elementos de política JSON do IAM: Condition no Guia do usuário do IAM.
- Use o IAM Access Analyzer para validar suas políticas do IAM a fim de garantir permissões seguras e funcionais: o IAM Access Analyzer valida as políticas novas e existentes para que elas sigam a linguagem de política do IAM (JSON) e as práticas recomendadas do IAM. O IAM Access Analyzer oferece mais de 100 verificações de política e recomendações acionáveis para ajudá-lo a criar políticas seguras e funcionais. Para obter mais informações, consulte Validação de políticas do IAM Access Analyzer no Guia do usuário do IAM.
- Exigir autenticação multifator (MFA): se houver um cenário que exija usuários do IAM ou um usuário raiz em sua Conta da AWS, ative a MFA para obter segurança adicional. Para exigir MFA quando as operações de API forem chamadas, adicione condições de MFA às suas políticas. Para

obter mais informações, consulte Configuração de acesso à API protegido por MFA no Guia do usuário do IAM.

Para obter mais informações sobre as práticas recomendadas do IAM, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.

Usar o console do Lambda

Para acessar o console do AWS Lambda, você deve ter um conjunto mínimo de permissões. Essas permissões devem permitir que você liste e visualize detalhes sobre os recursos do Lambda em seu. Conta da AWS Se você criar uma política baseada em identidade que seja mais restritiva do que as permissões mínimas necessárias, o console não funcionará como pretendido para entidades (usuários ou perfis) com essa política.

Não é necessário conceder permissões mínimas do console para usuários que fazem chamadas somente à AWS CLI ou à AWS API. Em vez disso, permita o acesso somente a ações que correspondam a operação de API que estiverem tentando executar.

Para obter um exemplo de política que concede acesso mínimo ao desenvolvimento de funções, consulte Escrever um exemplo de política que conceda permissões de usuário a uma função. Além das APIs do Lambda, o console do Lambda usa outros serviços para exibir a configuração do acionador e permite que você adicione novos acionadores. Se os usuários utilizam o Lambda com outros serviços, eles também precisam de acesso a esses serviços. Para obter mais detalhes sobre como configurar outros serviços do Lambda, consulte Invocando o Lambda com eventos de outros serviços da AWS.

Permitir que os usuários exibam as próprias permissões

Este exemplo mostra como é possível criar uma política que permite que os usuários do IAM visualizem as políticas gerenciadas e em linha anexadas a sua identidade de usuário. Essa política inclui permissões para concluir essa ação no console ou de forma programática usando a AWS CLI ou a AWS API.

```
{
    "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": "*"
        }
    ]
}
```

Políticas gerenciadas pela AWS do AWS Lambda

Uma política gerenciada pela AWS é uma política independente criada e administrada pela AWS. As políticas gerenciadas pela AWS são criadas para fornecer permissões a vários casos de uso comuns a fim de que você possa começar a atribuir permissões a usuários, grupos e perfis.

Lembre-se de que as políticas gerenciadas pela AWS podem não conceder permissões de privilégio mínimo para seus casos de uso específicos porque estão disponíveis para todos AWS os clientes da usarem. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo cliente da específicas para seus casos de uso.

Você não pode alterar as permissões definidas em políticas gerenciadas pela AWS. Se a AWS atualiza as permissões definidas em uma política gerenciada pela AWS, a atualização afeta

todas as identidades de entidades principais (usuários, grupos e perfis) às quais a política está vinculada. É mais provável que a AWS atualize uma política gerenciada pela AWS quando um novo AWS service (Serviço da AWS) é lançado ou novas operações de API são disponibilizadas para os serviços existentes.

Para obter mais informações, consulte AWSPolíticas gerenciadas pela no Guia do usuário do IAM.

Tópicos

AWSpolítica gerenciada: AWSLambda_FullAccess

Esta política concede acesso total às ações do Lambda. Também concede permissões a outros serviços da AWS usados para desenvolver e manter recursos do Lambda.

Você pode associar a política `AWSLambda_FullAccess` aos seus usuários, grupos e perfis.

Detalhes de permissão

Esta política inclui as seguintes permissões:

- `lambda`: concede às entidades principais acesso total ao Lambda.

- `cloudformation`: permite que as entidades principais descrevam as pilhas do AWS CloudFormation e listem os recursos nessas pilhas.
- `cloudwatch`— Permite que os diretores listem as CloudWatch métricas da Amazon e obtenham dados métricos.
- `ec2`: permite que as entidades principais descrevam grupos de segurança, sub-redes e VPCs.
- `iam`: permite que as entidades principais obtenham políticas, versões de políticas, funções, políticas de funções, políticas de funções anexadas e a lista de funções. Essa política também permite que as entidades principais passem perfis para o Lambda. A permissão `PassRole` é usada quando você atribui uma função de execução a uma função.
- `kms`: permite que as entidades principais listem aliases.
- `logs`— Permite que os diretores descrevam os grupos de CloudWatch registros da Amazon. Para grupos de logs associados a uma função do Lambda, essa política permite que a entidade principal descreva fluxos de log, obtenha eventos de log e filtre eventos de log.
- `states`: permite que as entidades principais descrevam e listem máquinas de estado do AWS Step Functions.
- `tag`: permite que as entidades principais obtenham recursos com base em suas tags.
- `xray`: permite que as entidades principais obtenham resumos de rastreamento de AWS X-Ray e recuperem uma lista de rastreamentos especificados por ID.

Para obter mais informações sobre essa política, incluindo o documento de política JSON e as versões da política, consulte o Guia AWSLambda_FullAccessde referência de políticas AWS gerenciadas.

AWSpolítica gerenciada: AWSLambda_ReadOnlyAccess

Essa política concede acesso somente para leitura aos recursos do Lambda e a outros serviços da AWS usados para desenvolver e manter recursos do Lambda.

Você pode associar a política `AWSLambda_ReadOnlyAccess` aos seus usuários, grupos e perfis.

Detalhes de permissão

Esta política inclui as seguintes permissões:

- `lambda`: permite que as entidades principais obtenham e listem todos os recursos.
- `cloudformation`: permite que as entidades principais descrevam e listem pilhas do AWS CloudFormation e listem os recursos nessas pilhas.

- `cloudwatch`— Permite que os diretores listem as CloudWatch métricas da Amazon e obtenham dados métricos.
- `ec2`: permite que as entidades principais descrevam grupos de segurança, sub-redes e VPCs.
- `iam`: permite que as entidades principais obtenham políticas, versões de políticas, funções, políticas de funções, políticas de funções anexadas e a lista de funções.
- `kms`: permite que as entidades principais listem aliases.
- `logs`— Permite que os diretores descrevam os grupos de CloudWatch registros da Amazon. Para grupos de logs associados a uma função do Lambda, essa política permite que a entidade principal descreva fluxos de log, obtenha eventos de log e filtre eventos de log.
- `states`: permite que as entidades principais descrevam e listem máquinas de estado do AWS Step Functions.
- `tag`: permite que as entidades principais obtenham recursos com base em suas tags.
- `xray`: permite que as entidades principais obtenham resumos de rastreamento de AWS X-Ray e recuperem uma lista de rastreamentos especificados por ID.

Para obter mais informações sobre essa política, incluindo o documento de política JSON e as versões da política, consulte o Guia AWSLambda_ReadOnlyAccessde referência de políticas AWS gerenciadas.

AWSpolítica gerenciada: AWSLambdaBasicExecutionRole

Essa política concede permissões para fazer upload de registros para o CloudWatch Logs.

Você pode associar a política `AWSLambdaBasicExecutionRole` aos seus usuários, grupos e perfis.

Para obter mais informações sobre essa política, incluindo o documento de política JSON e as versões da política, consulte o Guia AWSLambdaBasicExecutionRolede referência de políticas AWS gerenciadas.

AWSpolítica gerenciada: AWSLambdaDynamoDBExecutionRole

Essa política concede permissões para ler registros de um stream do Amazon DynamoDB e gravar em Logs. CloudWatch

Você pode associar a política `AWSLambdaDynamoDBExecutionRole` aos seus usuários, grupos e perfis.

Para obter mais informações sobre essa política, incluindo o documento de política JSON e as versões da política, consulte o Guia AWSLambdaDynamoDBExecutionRolede referência de políticas AWS gerenciadas.

AWSpolítica gerenciada: AWSLambdaENIManagementAccess

Essa política concede permissões para criar, descrever e excluir interfaces de rede elásticas usadas por uma função do Lambda habilitada para VPC.

Você pode associar a política `AWSLambdaENIManagementAccess` aos seus usuários, grupos e perfis.

Para obter mais informações sobre essa política, incluindo o documento de política JSON e as versões da política, consulte o Guia AWSLambdaENIManagementAccessde referência de políticas AWS gerenciadas.

AWSpolítica gerenciada: AWSLambdaExecute

Essa política `PUT` concede `GET` acesso ao Amazon Simple Storage Service e acesso total aos CloudWatch registros.

Você pode associar a política `AWSLambdaExecute` aos seus usuários, grupos e perfis.

Para obter mais informações sobre essa política, incluindo o documento de política JSON e as versões da política, consulte o Guia AWSLambdaExecutede referência de políticas AWS gerenciadas.

AWSpolítica gerenciada: AWSLambdaInvocation -DynamoDB

Essa política concede acesso de leitura ao Amazon DynamoDB Streams.

Você pode associar a política `AWSLambdaInvocation-DynamoDB` aos seus usuários, grupos e perfis.

Para obter mais informações sobre essa política, incluindo o documento de política JSON e as versões da política, consulte AWSLambdaInvocation-DynamoDB no Guia de referência de políticas AWSgerenciadas.

AWSpolítica gerenciada: AWSLambdaKinesisExecutionRole

Essa política concede permissões para ler eventos de um stream de dados do Amazon Kinesis e gravar CloudWatch em Logs.

Você pode associar a política `AWSLambdaKinesisExecutionRole` aos seus usuários, grupos e perfis.

Para obter mais informações sobre essa política, incluindo o documento de política JSON e as versões da política, consulte o Guia AWSLambdaKinesisExecutionRolede referência de políticas AWS gerenciadas.

AWSpolítica gerenciada: AWSLambdaMSKExecutionRole

Essa política concede permissões para ler e acessar registros de um cluster Amazon Managed Streaming for Apache Kafka, gerenciar interfaces de rede elásticas e gravar em Logs. CloudWatch

Você pode associar a política `AWSLambdaMSKExecutionRole` aos seus usuários, grupos e perfis.

Para obter mais informações sobre essa política, incluindo o documento de política JSON e as versões da política, consulte o Guia AWSLambdaMSKExecutionRolede referência de políticas AWS gerenciadas.

AWSpolítica gerenciada: AWSLambdaRole

Esta política concede permissões para invocar as funções do Lambda.

Você pode associar a política `AWSLambdaRole` aos seus usuários, grupos e perfis.

Para obter mais informações sobre essa política, incluindo o documento de política JSON e as versões da política, consulte o Guia AWSLambdaRolede referência de políticas AWS gerenciadas.

AWSpolítica gerenciada: AWSLambdaSQSQueueExecutionRole

Essa política concede permissões para ler e excluir mensagens de uma fila do Amazon Simple Queue Service e concede permissões de gravação para CloudWatch Logs.

Você pode associar a política `AWSLambdaSQSQueueExecutionRole` aos seus usuários, grupos e perfis.

Para obter mais informações sobre essa política, incluindo o documento de política JSON e as versões da política, consulte o Guia AWSLambdaSQSQueueExecutionRolede referência de políticas AWS gerenciadas.

AWSpolítica gerenciada: AWSLambdaVPCAccessExecutionRole

Essa política concede permissões para gerenciar interfaces de rede elásticas em uma Amazon Virtual Private Cloud e gravar em CloudWatch Logs.

Você pode associar a política `AWSLambdaVPCAccessExecutionRole` aos seus usuários, grupos e perfis.

Para obter mais informações sobre essa política, incluindo o documento de política JSON e as versões da política, consulte o Guia AWSLambdaVPCAccessExecutionRolede referência de políticas AWS gerenciadas.

Atualizações do Lambda para políticas gerenciadas pela AWS

Alteração	Descrição	Data
AWSLambdaVPCAccess ExecutionRole— Mudança	A Lambda atualizou a `AWSLambdaVPCAccess ExecutionRole` política para permitir a ação. `ec2:DescribeSubnets`	5 de janeiro de 2024
AWSLambda_ReadOnly Access— Mudança	O Lambda atualizou a política `AWSLambda_ReadOnly Access` para permitir que as entidades principais listem as pilhas do AWS CloudForm ation.	27 de julho de 2023
O AWS Lambda iniciou o rastreamento das alterações	O AWS Lambda começou a monitorar as alterações para as políticas gerenciadas da AWS.	27 de julho de 2023

Solução de problemas de identidade e acesso do AWS Lambda

Use as informações a seguir para ajudar a diagnosticar e corrigir problemas comuns que você possa encontrar ao trabalhar com o Lambda e o IAM.

Tópicos

- Não tenho autorização para executar uma ação no Lambda
- Não estou autorizado a realizar iam: PassRole

- Quero permitir que pessoas de fora da minha acessem meus Conta da AWS recursos do Lambda

Não tenho autorização para executar uma ação no Lambda

Se você receber uma mensagem de erro informando que não tem autorização para executar uma ação, suas políticas deverão ser atualizadas para permitir que você realize a ação.

O erro do exemplo a seguir ocorre quando o usuário do IAM `mateojackson` tenta usar o console para visualizar detalhes sobre um atributo *`my-example-widget`* fictício, mas não tem as permissões `lambda:`*`GetWidget`* fictícias.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 lambda:GetWidget on resource: my-example-widget
```

Nesse caso, a política do usuário `mateojackson` deve ser atualizada para permitir o acesso ao atributo *`my-example-widget`* usando a ação `lambda:`*`GetWidget`*.

Se você precisar de ajuda, entre em contato com seu administrador AWS. Seu administrador é a pessoa que forneceu suas credenciais de login.

Não estou autorizado a realizar iam: PassRole

Se você receber uma mensagem de erro informando que não tem autorização para executar a ação `iam:PassRole`, as políticas deverão ser atualizadas para permitir que você passe uma função para o Lambda.

Alguns Serviços da AWS permitem que você passe um perfil existente para o serviço, em vez de criar um novo perfil de serviço ou perfil vinculado ao serviço. Para fazer isso, é preciso ter permissões para passar o perfil para o serviço.

O erro de exemplo a seguir ocorre quando uma usuária do IAM chamada `marymajor` tenta usar o console para executar uma ação no Lambda. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar o perfil para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se você precisar de ajuda, entre em contato com seu administrador AWS. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero permitir que pessoas de fora da minha acessem meus Conta da AWS recursos do Lambda

É possível criar um perfil que os usuários de outras contas ou pessoas fora da sua organização possam utilizar para acessar seus recursos. É possível especificar quem é confiável para assumir o perfil. Para serviços compatíveis com as políticas baseadas em recursos ou listas de controle de acesso (ACLs), é possível utilizar essas políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte:

- Para saber se o Lambda oferece suporte a esses recursos, consulte Como o AWS Lambda funciona com o IAM.
- Para saber como conceder acesso a seus atributos em todas as Contas da AWS pertencentes a você, consulte Fornecimento de acesso a um usuário do IAM em outra Conta da AWS pertencente a você no Guia de usuário do IAM.
- Para saber como conceder acesso a seus recursos para terceiros Contas da AWS, consulte Fornecimento de acesso a Contas da AWS pertencentes a terceiros no Guia do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte Conceder acesso a usuários autenticados externamente (federação de identidades) no Guia do usuário do IAM.
- Para saber a diferença entre usar perfis e políticas baseadas em atributos para acesso entre contas, consulte Como os perfis do IAM diferem de políticas baseadas em atributos no Guia do usuário do IAM.

Criar uma estratégia de governança para funções e camadas do Lambda

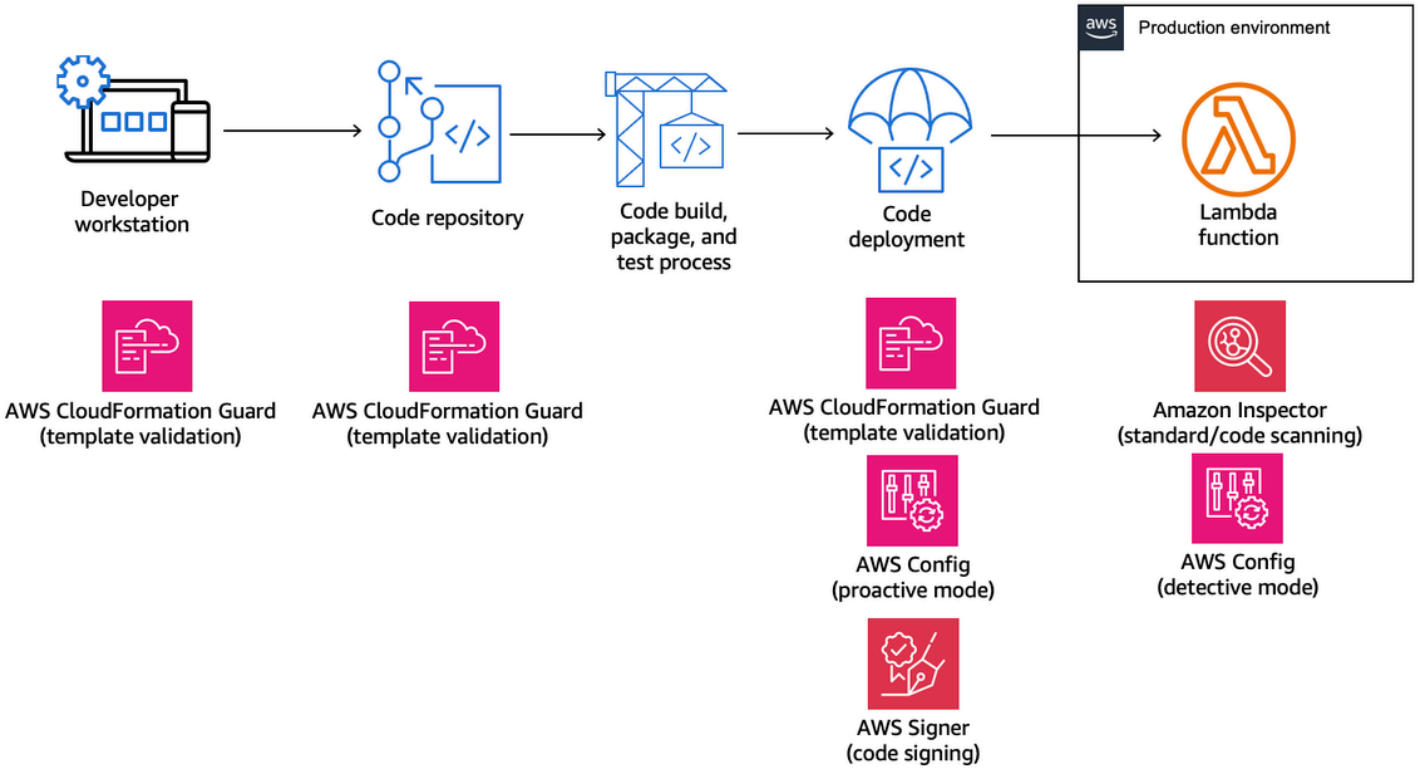
Para criar e implantar aplicações nativas de nuvem e com tecnologia sem servidor, você deve permitir agilidade e velocidade de comercialização com governança e barreiras de proteção adequadas. Você pode definir as prioridades em nível de negócios enfatizando a agilidade como a principal prioridade ou enfatizando a aversão ao risco por meio de governança, barreiras de proteção e controles. De uma maneira realista, você não terá uma estratégia de “uma ou outra”, mas

uma estratégia de “e” que equilibre a agilidade e as barreiras de proteção no seu ciclo de vida de desenvolvimento do software. Não importa onde esses requisitos se enquadram no ciclo de vida da sua empresa. Provavelmente os recursos de governança se tornarão um requisito de implementação nos seus processos e conjuntos de ferramentas.

Veja alguns exemplos de controles de governança que uma organização pode implementar para o Lambda:

- As funções do Lambda não devem ser acessíveis ao público.
- As funções do Lambda devem ser anexadas a uma VPC.
- As funções do Lambda não devem usar runtimes obsoletos.
- As funções do Lambda devem ser marcadas com um conjunto de tags obrigatórias.
- As camadas do Lambda não devem ser acessíveis fora da organização.
- As funções do Lambda com um grupo de segurança anexado devem ter tags correspondentes entre a função e o grupo de segurança.
- As funções do Lambda com uma camada anexada devem usar uma versão aprovada
- As variáveis de ambiente do Lambda devem ser criptografadas em repouso com uma chave gerenciada pelo cliente.

O seguinte diagrama é um exemplo de estratégia de governança detalhada que implementa controles e políticas em todo o processo de desenvolvimento e implantação de software:



Os tópicos a seguir explicam como implementar controles para desenvolver e implantar funções do Lambda na sua organização, tanto para a startup quanto para a empresa. Sua organização talvez já tenha ferramentas implementadas. Os tópicos a seguir adotam uma abordagem modular para esses controles para que você possa escolher os componentes de que realmente precisa.

Tópicos

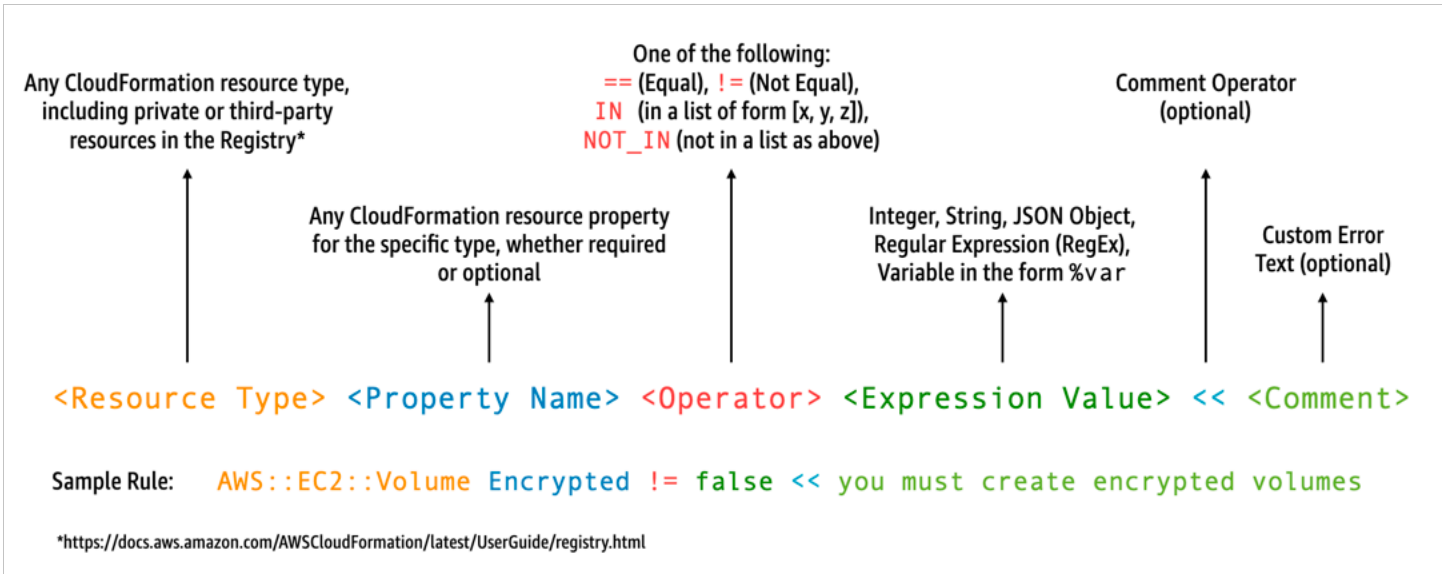
- Controles proativos para Lambda com AWS CloudFormation Guard
- Implementar controles preventivos para Lambda com o AWS Config
- Detectar implantações e configurações do Lambda não compatíveis com o AWS Config
- Assinatura de código do Lambda com AWS Signer
- Automatizar as avaliações de segurança para o Lambda com o Amazon Inspector
- Implementar a observabilidade para segurança e conformidade do Lambda

Controles proativos para Lambda com AWS CloudFormation Guard

AWS CloudFormation Guardé uma ferramenta de avaliação de código aberto e de uso geral. policy-as-code Ele pode ser usado para governança e conformidade preventivas, ao validar modelos de Infraestrutura como Código (IaC) e composições de serviços em relação às regras de política. Essas regras podem ser personalizadas com base nos requisitos da equipe ou da organização. Para funções do Lambda, as regras do Guard podem ser usadas para controlar a criação de recursos e as atualizações de configuração, ao definir as configurações de propriedade necessárias durante a criação ou a atualização de uma função do Lambda.

Os administradores de conformidade definem a lista de controles e políticas de governança que são necessários para implantar e atualizar as funções do Lambda. Os administradores da plataforma implementam os controles em pipelines de CI/CD, como webhooks de validação pré-confirmação com repositórios de código, e fornecem aos desenvolvedores ferramentas de linha de comando para validar modelos e códigos em estações de trabalho locais. Os desenvolvedores criam código, validam modelos com ferramentas de linha de comando e, em seguida, confirmam o código nos repositórios, que são validados automaticamente por meio dos pipelines de CI/CD antes da implantação em um ambiente da AWS.

O Guard permite que você crie suas regras e implemente seus controles com uma linguagem específica de domínio como a seguir.



Por exemplo, suponha que você deseje garantir que os desenvolvedores escolham somente os runtimes mais recentes. Você pode especificar duas políticas diferentes, uma para identificar os

runtimes que já estão descontinuados e outra para identificar os runtimes que serão descontinuados em breve. Para fazer isso, você pode criar o seguinte arquivo `etc/rules.guard`:

```
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.>>
            }
        }
    }
}
```

Agora, suponha que você escreva o seguinte `iac/lambda.yaml` CloudFormation modelo que define uma função 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
```

Depois de instalar o utilitário Guard, valide o modelo:

```
cfn-guard validate --rules etc/rules.guard --data iac/lambda.yaml
```

O resultado se parece com:

```
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

      }
    }
```

```
  }
}
```

O Guard permite que os desenvolvedores vejam em suas estações de trabalho locais que precisam atualizar o modelo para usar um runtime permitido pela organização. Isso acontece antes da confirmação em um repositório de código e, posteriormente, apresentar falha nas verificações em um pipeline de CI/CD. Como resultado, os desenvolvedores recebem esse feedback sobre como desenvolver modelos em conformidade e dedicar o tempo à criação de códigos que agreguem valor comercial. Esse controle pode ser aplicado na estação de trabalho local do desenvolvedor, em um webhook de validação pré-confirmação e/ou no pipeline de CI/CD antes da implantação.

Advertências

Se você estiver usando modelos do AWS Serverless Application Model (AWS SAM) para definir funções do Lambda, saiba que precisa atualizar a regra do Guard para pesquisar o tipo de recurso `AWS::Serverless::Function` como a seguir.

```
let lambda_functions = Resources.*[
    Type == "AWS::Serverless::Function"
]
```

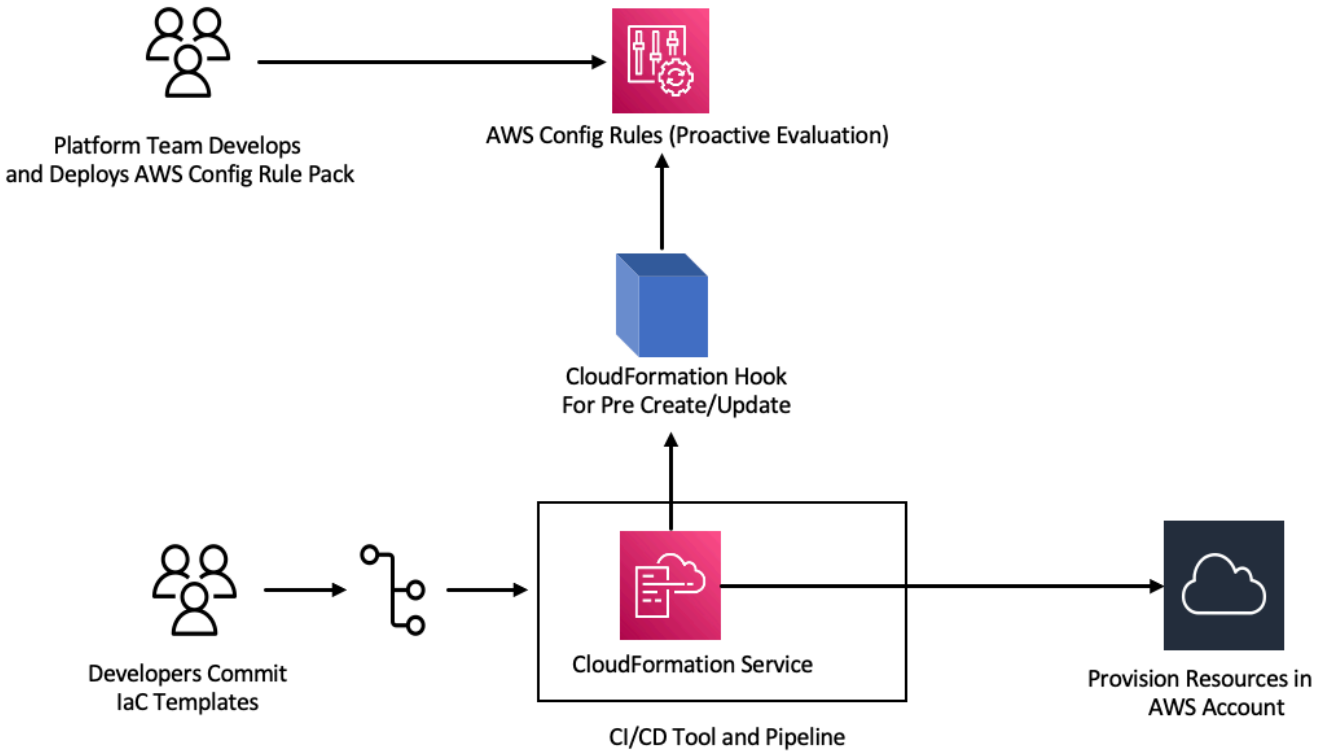
O Guard também espera que as propriedades sejam incluídas na definição do recurso. Enquanto isso, os modelos do AWS SAM permitem que as propriedades sejam especificadas em uma seção Globals separada. As propriedades definidas na seção Globals não são validadas com suas regras do Guard.

Conforme descrito na documentação de solução de problemas do Guard, saiba que o Guard não é compatível com funções intrínsecas de forma abreviada, como `!GetAtt` ou `!Sub`. Em vez disso, ele requer o uso de formas expandidas: `Fn::GetAtt` e `Fn::Sub`. (O exemplo anterior não avalia a propriedade Role. Portanto, a função intrínseca de forma abreviada foi usada para simplificar.)

Implementar controles preventivos para Lambda com o AWS Config

É essencial garantir a conformidade nas aplicações com tecnologia sem servidor o mais cedo possível no processo de desenvolvimento. Neste tópico, abordaremos como implementar controles preventivos usando o AWS Config. Isso permite que você implemente verificações de conformidade mais cedo no processo de desenvolvimento e implemente os mesmos controles nos seus pipelines de CI/CD. Isso também padroniza os controles em um repositório de regras gerenciado de forma centralizada para que você possa aplicar os controles de forma consistente em todas as suas contas da AWS.

Por exemplo, suponha que os administradores de conformidade tenham definido um requisito para garantir que todas as funções do Lambda incluam rastreamento do AWS X-Ray. Com o modo proativo do AWS Config, você pode executar verificações de conformidade nos recursos da função do Lambda antes da implantação, reduzindo o risco de implantar funções do Lambda configuradas incorretamente e economizando o tempo dos desenvolvedores ao fornecer feedback mais rápido sobre a infraestrutura como modelos de código. A seguir, é apresentada uma visualização do fluxo de controles preventivos com o AWS Config:



Considere a exigência de que todas as funções do Lambda tenham o rastreamento ativado. Em resposta, a equipe da plataforma identifica a necessidade de uma regra do AWS Config específica ser executada de forma proativa em todas as contas. Essa regra sinaliza qualquer função do Lambda que não tenha uma configuração de rastreamento do X-Ray configurada como um recurso fora de conformidade. A equipe desenvolve uma regra, coloca-a em um pacote de conformidade e implanta o pacote de conformidade em todas as contas da AWS para garantir que todas as contas da organização apliquem uniformemente esses controles. Você pode criar a regra na sintaxe AWS CloudFormation Guard 2.x.x, que assume o seguinte formato:

```
rule name when condition { assertion }
```

Veja a seguir um exemplo de regra do Guard que verifica se as funções do Lambda têm o rastreamento habilitado:

```
rule lambda_tracing_check {
  when configuration.tracingConfig exists {
      configuration.tracingConfig.mode == "Active"
  }
}
```

A equipe da plataforma toma medidas adicionais ao exigir que cada implantação do AWS CloudFormation invoque um hook pré-criação/atualização. A equipe assume total responsabilidade pelo desenvolvimento desse hook e pela configuração do pipeline, fortalecendo o controle centralizado das regras de conformidade e sustentando a aplicação consistente delas em todas as implantações. Para desenvolver, empacotar e registrar um hook, consulte Desenvolver hooks do AWS CloudFormation na documentação da interface de linha de comando do CloudFormation (CFN-CLI). Você pode usar a CLI do CloudFormation para criar o projeto do hook:

```
cfn init
```

Esse comando solicita a você algumas informações básicas sobre o projeto do hook e cria um projeto com os seguintes arquivos:

```
README.md
<hook-name>.json
rpdk.log
src/handler.py
template.yml
hook-role.yaml
```

Como desenvolvedor de hooks, você precisa adicionar o tipo de recurso de destino desejado no arquivo de configuração `<hook-name>.json`. Na configuração abaixo, um hook é configurado para ser executado antes que qualquer função do Lambda seja criada usando o CloudFormation. Você também pode adicionar manipuladores semelhantes para ações de `preUpdate` e `preDelete`.

```
    "handlers": {
        "preCreate": {
            "targetNames": [
                "AWS::Lambda::Function"
            ],
            "permissions": []
        }
    }
```

Você também precisa garantir que o hook do CloudFormation tenha as permissões apropriadas para chamar as APIs do AWS Config. Você pode fazer isso atualizando o arquivo de definição de perfil denominado `hook-role.yaml`. Por padrão, o arquivo de definição de perfil tem a política de confiança a seguir, que permite que o CloudFormation assuma o perfil.

```
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - hooks.cloudformation.amazonaws.com
                - resources.cloudformation.amazonaws.com
```

Para permitir que esse hook chame APIs de configuração, você deve adicionar as permissões a seguir à declaração da política. Em seguida, você envia o projeto do hook usando o comando `cfn submit`, em que o CloudFormation cria um perfil para você com as permissões necessárias.

```
      Policies:
        - PolicyName: HookTypePolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - "config:Describe*"
                  - "config:Get*"
```

```
                  - "config:List*"
                  - "config:SelectResourceConfig"
                Resource: "*
```

Em seguida, você precisa criar uma função do Lambda em um arquivo `src/handler.py`. Nesse arquivo, você encontra métodos denominados `preCreate`, `preUpdate` e `preDelete` já criados quando você iniciou o projeto. Seu objetivo é criar uma função comum e reutilizável que chame a API `StartResourceEvaluation` do AWS Config no modo proativo usando o AWS SDK for Python (Boto3). Essa chamada de API usa as propriedades do recurso como entrada e avalia o recurso em relação à definição da regra.

```
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.")
```

Agora, você pode chamar a função comum do manipulador para o hook de pré-criação. Veja um exemplo do manipulador:

```
@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}")
```

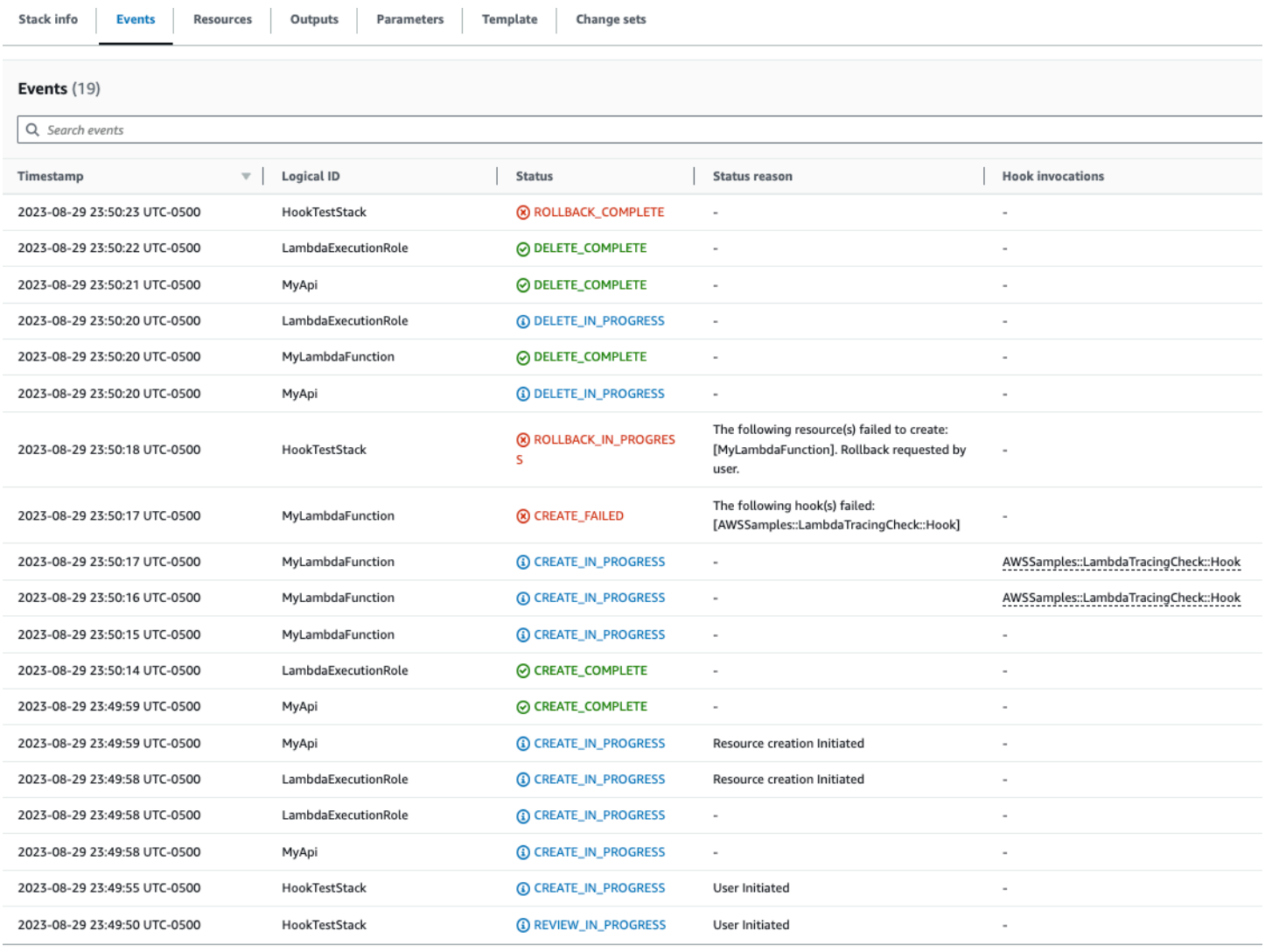
Após essa etapa, você poderá registrar o hook e configurá-lo para ouvir todos os eventos de criação de funções do AWS Lambda.

Um desenvolvedor prepara o modelo de infraestrutura como código (IaC) para um microsserviço com tecnologia sem servidor usando o Lambda. Essa preparação inclui a adesão aos padrões internos, seguida pelo teste local e pela confirmação do modelo para o repositório. Veja um exemplo de modelo de 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
```

Como parte do processo de CI/CD, quando o modelo do CloudFormation é implantado, o serviço do CloudFormation invoca o hook de pré-criação/atualização logo antes de provisionar o tipo de recurso da `AWS::Lambda::Function`. O hook utiliza regras do AWS Config executadas no modo proativo para verificar se a configuração da função do Lambda inclui a configuração de rastreamento obrigatória. A resposta do hook determina a próxima etapa. Se estiver em conformidade, o hook indicará êxito e o CloudFormation continuará a provisionar os recursos. Caso contrário, a implantação da pilha do CloudFormation apresentará falha, o pipeline será interrompido imediatamente e o sistema registrará os detalhes para análise posterior. As notificações de conformidade são enviadas aos investidores relevantes.

Você pode encontrar as informações de êxito/falha do hook no console do CloudFormation:



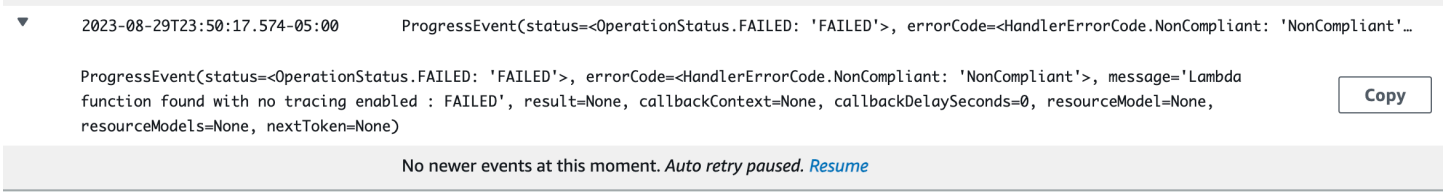
Stack info | Events | Resources | Outputs | Parameters | Template | Change sets

Events (19)

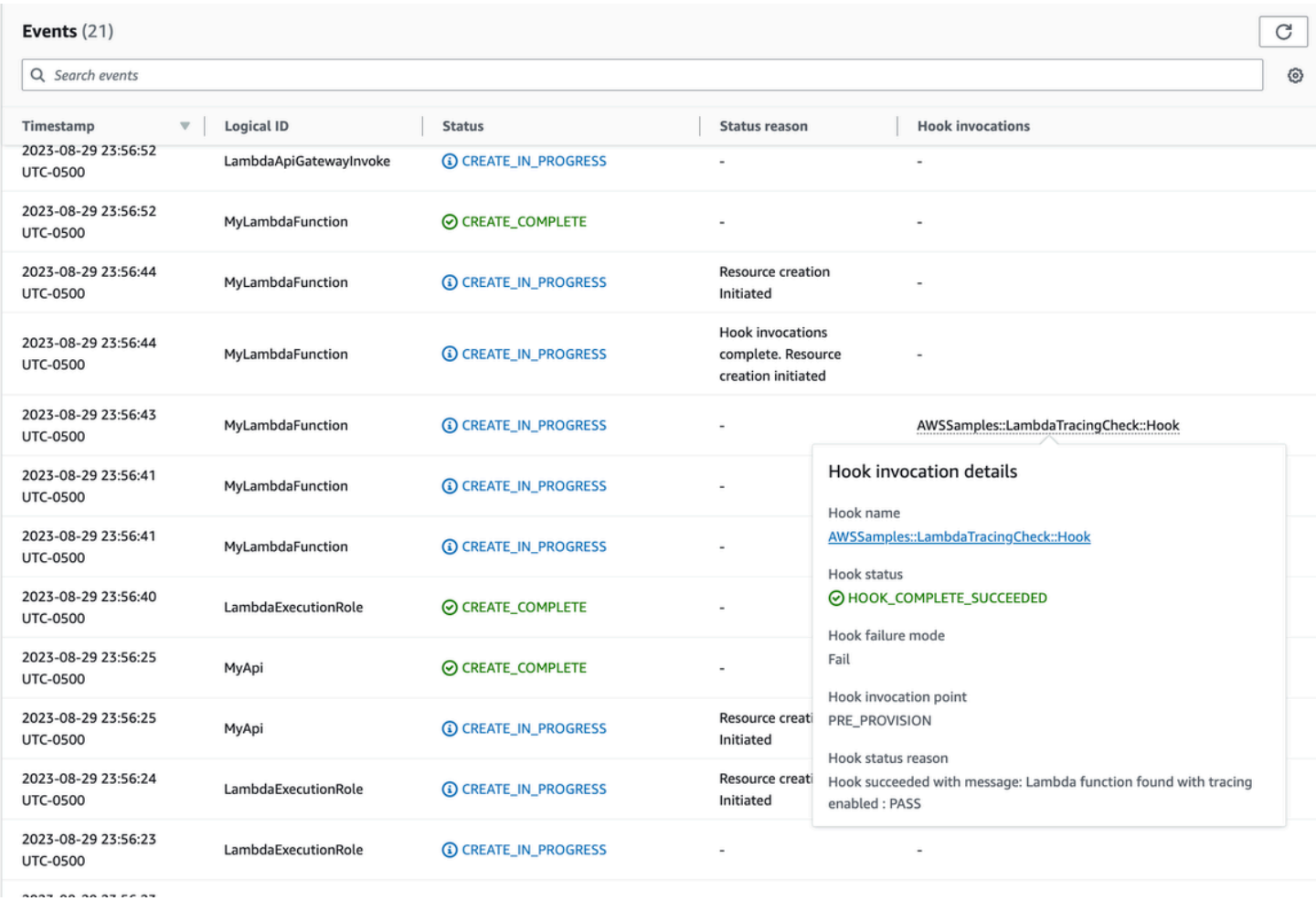
Search events

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 você tiver logs habilitados para o hook do CloudFormation, poderá capturar o resultado da avaliação do hook. Veja um exemplo de log de um hook com status de falha, indicando que a função do Lambda não tem o X-Ray ativado:



Se o desenvolvedor optar por alterar a IaC para atualizar o valor de `TracingConfig Mode` para `Active` e reimplantar, o hook será executado com êxito e a pilha prosseguirá com a criação do recurso do Lambda.



Dessa forma, você pode implementar controles preventivos com o AWS Config em modo proativo ao desenvolver e implantar recursos com tecnologia sem servidor nas suas contas da AWS. Ao integrar regras do AWS Config ao pipeline de CI/CD, você pode identificar e, opcionalmente, bloquear

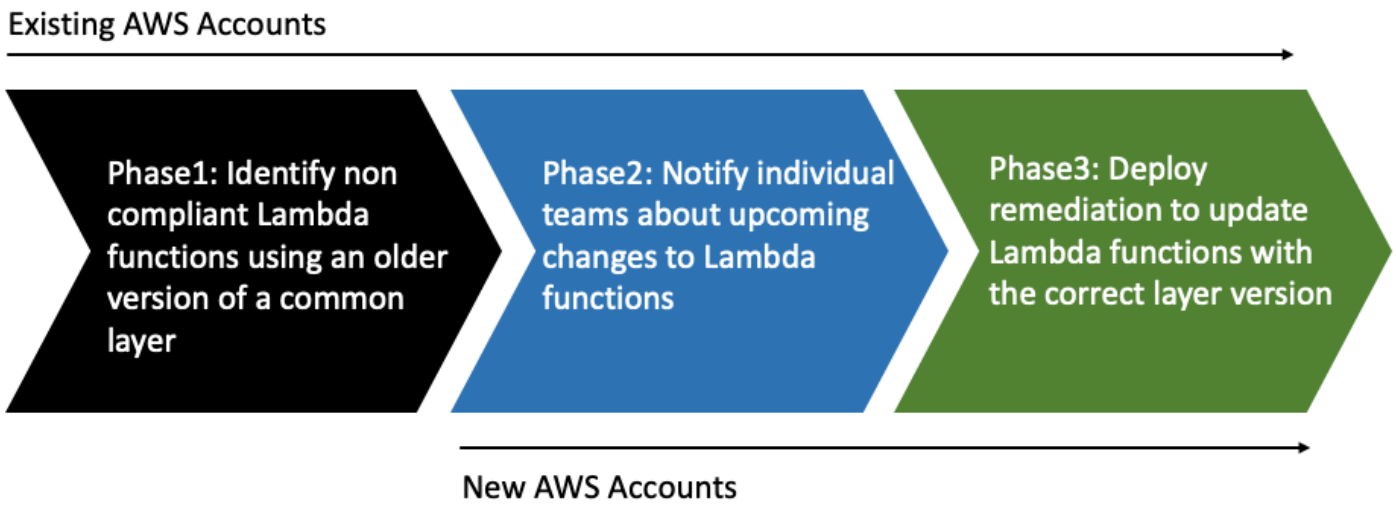
implantações de recursos fora de conformidade, como funções do Lambda que não possuem uma configuração de rastreamento ativa. Isso garante que somente recursos em conformidade com as políticas de governança mais recentes sejam implantados nos ambientes da AWS.

Detectar implantações e configurações do Lambda não compatíveis com o AWS Config

Além da avaliação proativa, o AWS Config também pode detectar de forma reativa implantações e configurações de recursos que não estejam em conformidade com suas políticas de governança. Isso é importante porque as políticas de governança evoluem à medida que a organização aprende e implementa novas melhores práticas.

Considere um cenário em que você define uma política totalmente nova ao implantar ou atualizar as funções do Lambda: todas as funções do Lambda devem sempre usar uma versão específica e aprovada da camada do Lambda. Você pode configurar o AWS Config para monitorar funções novas ou atualizadas para configurações de camadas. Se o AWS Config detectar uma função que não está usando uma versão de camada aprovada, ele sinalizará a função como um recurso fora de conformidade. Opcionalmente, você pode configurar o AWS Config para remediar automaticamente o recurso ao especificar uma ação de remediação usando um documento de automação do AWS Systems Manager. Por exemplo, você pode criar um documento de automação em Python usando o AWS SDK for Python (Boto3), que atualiza a função fora de conformidade para apontar para a versão da camada aprovada. Portanto, o AWS Config funciona como um controle de detecção e correção, automatizando o gerenciamento da conformidade.

Vamos detalhar esse processo em três fases importantes de implementação:



Fase 1: identificar os recursos de acesso

Comece ativando o AWS Config em todas as contas e configurando-o para registrar as funções do AWS Lambda. Isso permite que o AWS Config observe quando as funções do Lambda são criadas

ou atualizadas. Em seguida, você pode configurar regras de política personalizadas para verificar violações de políticas específicas, que usam a sintaxe AWS CloudFormation Guard. As regras de proteção assumem a seguinte forma geral:

```
rule name when condition { assertion }
```

A seguir, é mostrado um exemplo de regra que verifica se uma camada não está configurada para uma versão antiga da camada:

```
rule desiredlayer when configuration.layers !empty {
    some configuration.layers[*].arn != CONFIG_RULE_PARAMETERS.OldLayerArn
}
```

Vamos entender a sintaxe e a estrutura da regra:

- Nome da regra: o nome da regra no exemplo fornecido é `desiredlayer`.
- Condição: essa cláusula especifica a condição segundo a qual a regra deve ser verificada. No exemplo fornecido, a condição é `configuration.layers !empty`. Isso significa que o recurso deve ser avaliado somente quando a propriedade `layers` na configuração não está vazia.
- Afirmação: após a cláusula `when`, uma declaração determina o que a regra verifica. A declaração `some configuration.layers[*].arn != CONFIG_RULE_PARAMETERS.OldLayerArn` verifica se algum dos ARNs de camada do Lambda não corresponde ao valor `OldLayerArn`. Se não houver correspondência, a declaração será verdadeira e a regra será aprovada; caso contrário, ela será reprovada.

`CONFIG_RULE_PARAMETERS` é um conjunto especial de parâmetros configurado com a regra AWS Config. Nesse caso, `OldLayerArn` é um parâmetro dentro de `CONFIG_RULE_PARAMETERS`. Isso permite que os usuários forneçam um ARN específico que considerem antigo ou desativado e, em seguida, a regra verifica se alguma função do Lambda está usando este ARN antigo.

Fase 2: visualizar e projetar

O AWS Config reúne os dados de configuração e os armazena em buckets do Amazon Simple Storage Service (Amazon S3). Você pode usar o Amazon Athena para consultar esses dados diretamente nos buckets do S3. Com o Athena, você pode agregar esses dados no nível da organização, gerando uma visão holística das configurações dos recursos em todas as suas contas. Para definir a agregação de dados de configuração de recursos, consulte Visualizing AWS Config data using Athena and Amazon QuickSight no AWS Cloud Operations and Management Blog.

Veja a seguir um exemplo de consulta do Athena para identificar todas as funções do Lambda que usam um ARN de camada específico:

```
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%'
```

Veja os resultados da consulta:

Query results | Query stats

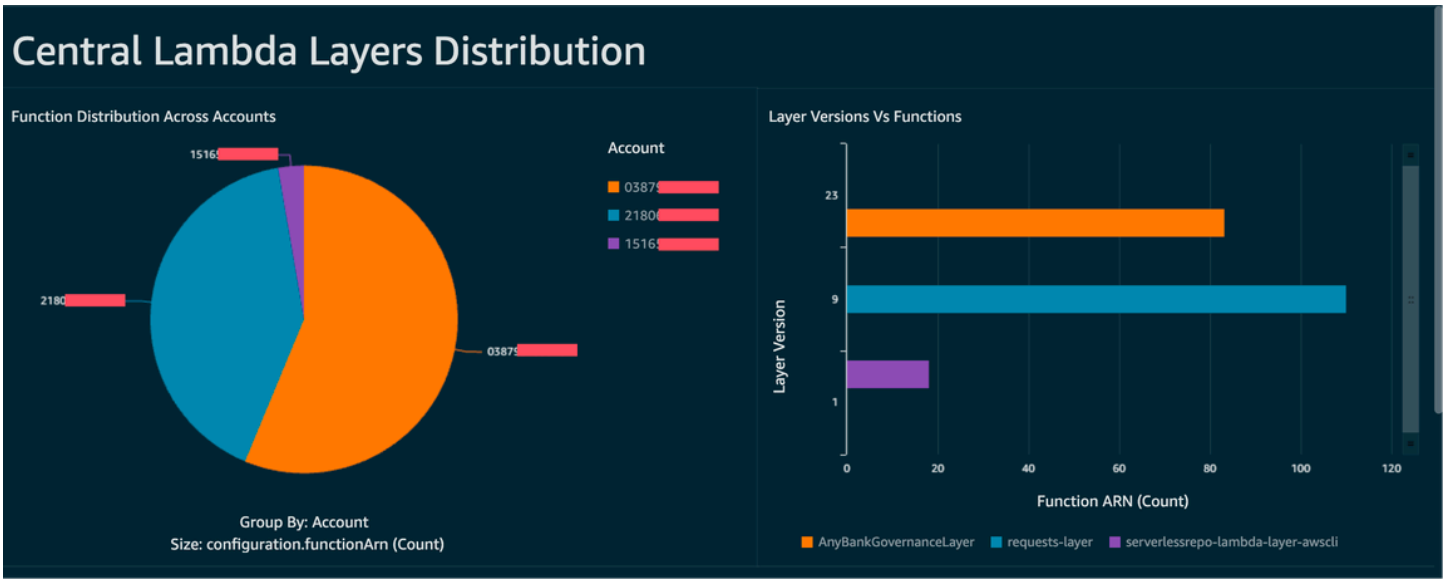
Completed — Time in queue: 127 ms — Run time: 1.803 sec — Data scanned: 239.40 KB

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

Com os dados do AWS Config agregados em toda a organização, você pode criar um painel usando o Amazon QuickSight. Ao importar os resultados do Athena para o Amazon QuickSight, você pode visualizar até que ponto as funções do Lambda aderem à regra de versão da camada. Esse painel pode destacar recursos em conformidade e fora de conformidade, o que ajuda você a determinar sua política de aplicação, conforme descrito na próxima seção. A imagem a seguir é um exemplo de painel que relata a distribuição das versões de camadas aplicadas às funções dentro da organização.



Fase 3: implementar e aplicar

Agora você pode, opcionalmente, emparelhar sua regra de versão de camada criada na fase 1 com uma ação de remediação por meio de um documento de automação do Systems Manager, que você cria como um script em Python escrito com o AWS SDK for Python (Boto3). O script chama a ação

de API UpdateFunctionConfiguration para cada função do Lambda, atualizando a configuração da função com o novo ARN da camada. Como alternativa, você pode fazer com que o script envie uma solicitação pull para o repositório do código para atualizar o ARN da camada. Dessa forma, futuras implantações de código também serão atualizadas com o ARN correto da camada.

Assinatura de código do Lambda com AWS Signer

O AWS Signer é um serviço de assinatura de código totalmente gerenciado que permite que você valide o código em uma assinatura digital para confirmar que o código está inalterado e é de um publicador confiável. O AWS Signer pode ser usado com o AWS Lambda para verificar se as funções e as camadas permanecem inalteradas antes da implantação nos ambientes da AWS. Isso protege sua organização contra agentes mal-intencionados que possam ter obtido credenciais para criar novas funções ou atualizar as existentes.

Para configurar a assinatura de código para suas funções do Lambda, comece criando um bucket do S3 com versionamento habilitado. Depois disso, crie um perfil de assinatura com AWS Signer, especifique Lambda como plataforma e, em seguida, especifique um período em dias em que o perfil de assinatura será válido. Exemplo:

```
Signer:
  Type: AWS::Signer::SigningProfile
  Properties:
    PlatformId: AWSLambda-SHA384-ECDSA
    SignatureValidityPeriod:
      Type: DAYS
      Value: !Ref pValidDays
```

Em seguida, use o perfil de assinatura e crie uma configuração de assinatura com o Lambda. Você precisa especificar o que fazer quando a configuração de assinatura vir um artefato que não corresponda à assinatura digital esperada: avisar (mas permitir a implantação) ou aplicar (e bloquear a implantação). O exemplo abaixo está configurado para aplicar e bloquear implantações.

```
SigningConfig:
  Type: AWS::Lambda::CodeSigningConfig
  Properties:
    AllowedPublishers:
      SigningProfileVersionArns:
        - !GetAtt Signer.ProfileVersionArn
    CodeSigningPolicies:
      UntrustedArtifactOnDeployment: Enforce
```

Agora, o AWS Signer está configurado com o Lambda para bloquear implantações não confiáveis. Suponhamos que você tenha concluído a codificação de uma solicitação de atributo e agora esteja prestes a implantar a função. A primeira etapa é compactar o código com as dependências

apropriadas e depois assinar o artefato usando o perfil de assinatura que você criou. Você pode fazer isso carregando o artefato zip no bucket do S3 e iniciando um trabalho de assinatura.

```
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
```

Você obtém uma saída como mostrado a seguir, em que o `jobId` é o objeto criado no bucket e no prefixo de destino e `jobOwner` é a Conta da AWS ID de 12 dígitos em que o trabalho foi executado.

```
{
    "jobId": "87a3522b-5c0b-4d7d-b4e0-4255a8e05388",
    "jobOwner": "111122223333"
  }
```

E, agora, você pode implantar a função usando o objeto S3 assinado e a configuração de assinatura de código que você criou.

```
  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
```

Como alternativa, você pode testar a implantação de uma função com o artefato zip original não assinado. A implantação deverá apresenta falha com a seguinte mensagem de erro:

```
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 você estiver criando e implantando funções usando o AWS Serverless Application Model (AWS SAM), o comando de pacote manipulará o upload do artefato zip no S3, também iniciará o trabalho de assinatura e o artefato será assinado. Você pode fazer isso usando os seguintes parâmetros e comando:

```
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
```

O AWS Signer ajuda você a verificar se os artefatos zip implantados nas contas são confiáveis para implantação. Você pode incluir o processo acima em pipelines de CI/CD e exigir que todas as funções tenham uma configuração de assinatura de código anexada usando as técnicas descritas nos tópicos anteriores. Ao usar a assinatura de código com implantações de funções do Lambda, você evita que agentes mal-intencionados, que possam ter obtido credenciais para criar ou atualizar funções, injetem código malicioso nas funções.

Automatizar as avaliações de segurança para o Lambda com o Amazon Inspector

O Amazon Inspector é um serviço de gerenciamento de vulnerabilidades que verifica continuamente as workloads em busca de vulnerabilidades de software conhecidas e exposições não intencionais da rede. O Amazon Inspector cria uma descoberta que descreve a vulnerabilidade, identifica o recurso afetado, classifica a gravidade da vulnerabilidade e fornece orientações para correção.

O suporte do Amazon Inspector fornece avaliações de vulnerabilidade de segurança contínuas e automatizadas para funções do Lambda e camadas. O Amazon Inspector fornece dois tipos de verificação para o Lambda:

- Verificação padrão do Lambda: verifica as dependências das aplicações dentro de uma função do Lambda e suas camadas em busca de vulnerabilidades de pacotes.
- Verificação de código do Lambda: verifica o código da aplicação personalizada nas funções e camadas em busca de vulnerabilidades de código. Ative o escaneamento padrão do Lambda ou ative o escaneamento padrão do Lambda junto com o escaneamento de código do Lambda.

Para habilitar o Amazon Inspector, navegue até o console do Amazon Inspector, expanda a seção Configurações e escolha Gerenciamento de contas. Na guia Contas, escolha Ativar e selecione uma das opções de verificação.

Você pode habilitar o Amazon Inspector para várias contas e delegar permissões para gerenciar o Amazon Inspector para a organização em contas específicas enquanto configura o Amazon Inspector. Durante a habilitação, você precisa conceder permissões ao Amazon Inspector ao criar a função: `AWSServiceRoleForAmazonInspector2`. O console do Amazon Inspector permite que você crie essa função usando uma opção de um clique.

Para a verificação padrão do Lambda, o Amazon Inspector inicia verificações de vulnerabilidade das funções do Lambda nas seguintes situações:

- Assim que o Amazon Inspector descobre uma função do Lambda existente.
- Quando você implanta uma nova função do Lambda.
- Ao implantar uma atualização no código do aplicativo ou nas dependências de uma função do Lambda existente ou de suas camadas.
- Sempre que o Amazon Inspector adiciona um novo item de CVEs (vulnerabilidades e exposições comuns) ao seu banco de dados, e esse CVE é relevante para sua função.

Para a verificação do código do Lambda, o Amazon Inspector avalia o código da aplicação da função do Lambda usando raciocínio automatizado e machine learning que analisam o código da aplicação para verificar a conformidade geral com a segurança. Se o Amazon Inspector detectar uma vulnerabilidade no código da aplicação da função do Lambda, o Amazon Inspector produzirá uma descoberta detalhada de Vulnerabilidade de código. Para obter uma lista de detecções possíveis, consulte a Amazon CodeGuru Detector Library.

Para visualizar as descobertas, acesse o console do Amazon Inspector. No menu Descobertas, escolha Por função do Lambda para exibir os resultados da verificação de segurança que foi executada nas funções do Lambda.

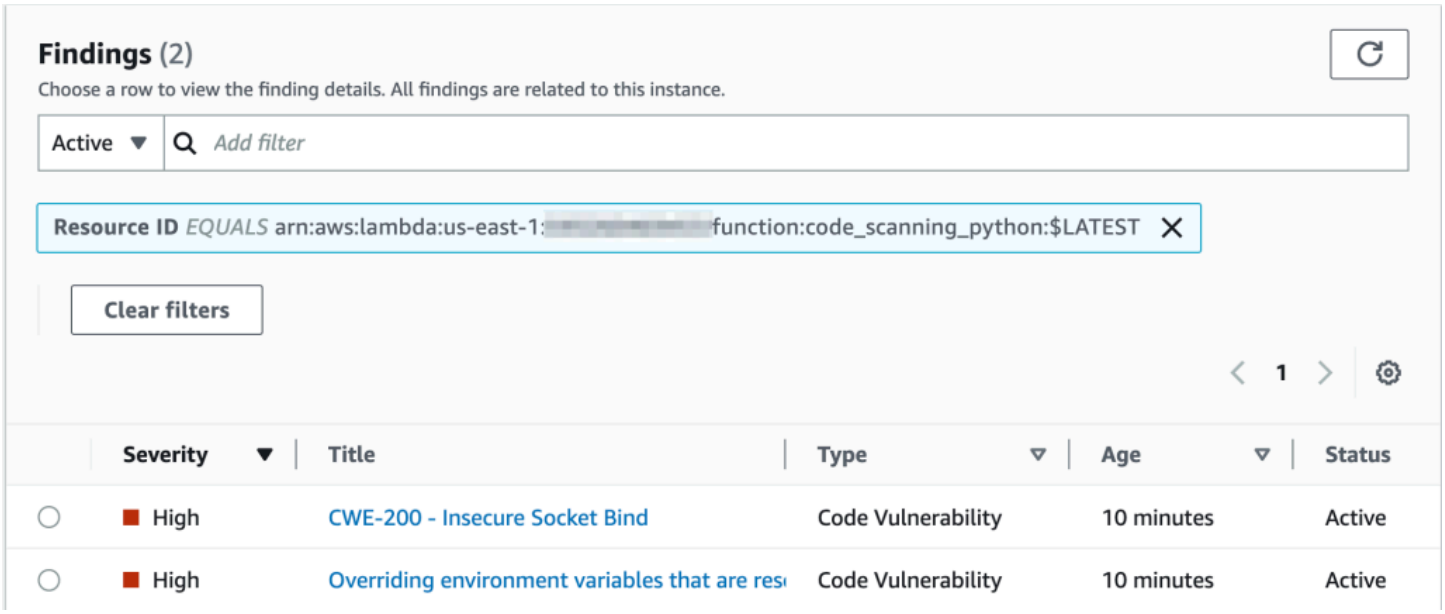
Para excluir uma função do Lambda da verificação padrão, marque a função com o seguinte par de chave/valor:

- `Key:InspectorExclusion`
- `Value:LambdaStandardScanning`

Para excluir uma função do Lambda das verificações de código, marque a função com o seguinte par de chave/valor:

- `Key:InspectorCodeExclusion`
- `Value:LambdaCodeScanning`

Por exemplo, conforme mostrado na imagem a seguir, o Amazon Inspector detecta automaticamente vulnerabilidades e categoriza as descobertas do tipo Vulnerabilidade de código, o que indica que a vulnerabilidade está no código da função e não em uma das bibliotecas dependentes do código. Você pode verificar esses detalhes para uma função específica ou várias funções ao mesmo tempo.



Você pode se aprofundar em cada uma dessas descobertas e saber como corrigir o 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.

Ao trabalhar com suas funções do Lambda, certifique-se de cumprir as convenções de nomenclatura para as funções do Lambda. Para ter mais informações, consulte Usar variáveis de ambiente no Lambda para configurar valores no código.

Você é responsável pelas sugestões de remediação que aceita. Sempre analise as sugestões de remediação antes de aceitá-las. Talvez seja necessário fazer edições nas sugestões de correção para garantir que o código faça o que você pretende.

Implementar a observabilidade para segurança e conformidade do Lambda

O AWS Config é uma ferramenta útil para encontrar e corrigir recursos da AWS com tecnologia sem servidor que não estão em conformidade. Cada alteração que você faz nos seus recursos com tecnologia sem servidor é registrada no AWS Config. Além disso, o AWS Config permite o armazenamento de dados de snapshot de configuração no S3. Você pode usar o Amazon Athena e o Amazon QuickSight para criar painéis e ver dados. AWS Config Em Detectar implantações e configurações do Lambda não compatíveis com o AWS Config, discutimos como podemos visualizar uma determinada configuração, como camadas do Lambda. Este tópico expande esses conceitos.

Visibilidade das configurações do Lambda

Você pode usar consultas para obter configurações importantes, como ID da Conta da AWS, região, configuração de rastreamento do AWS X-Ray, configuração da VPC, tamanho de memória, runtime e tags. Veja um exemplo de consulta que você pode usar para obter essas informações do 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
```

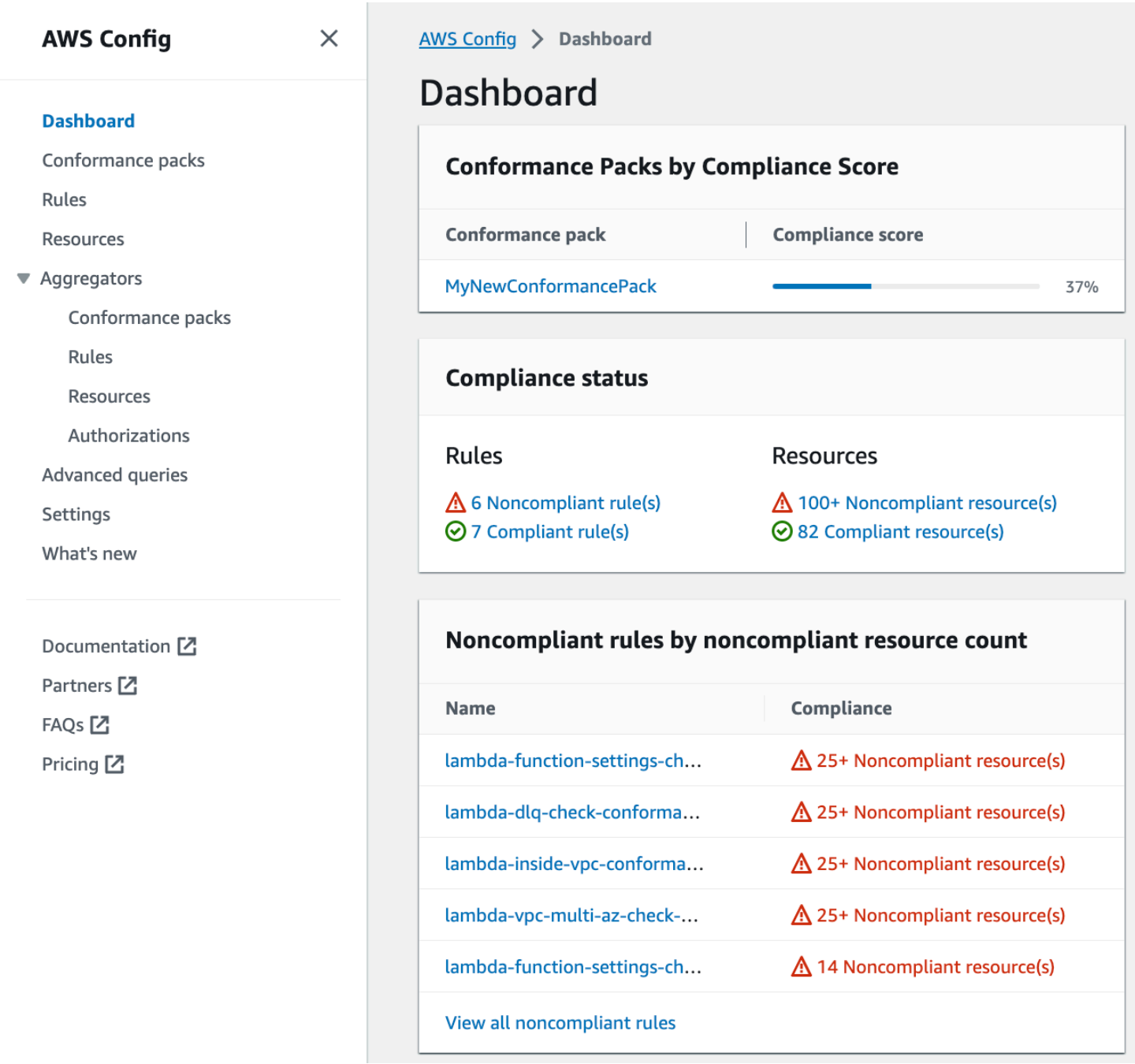
Você pode usar a consulta para criar um QuickSight painel da Amazon e visualizar os dados. Para agregar dados de configuração de AWS recursos, criar tabelas no Athena e criar painéis da QuickSight Amazon com base nos dados do Athena, consulte Visualização de dados AWS Config usando o Athena e a Amazon QuickSight no blog Cloud Operations and Management. AWS Sobretudo, essa consulta também recupera informações de tags para as funções. Isso permite insights mais profundos das workloads e ambientes, especialmente se forem empregadas tags personalizadas.

Para obter mais informações sobre as ações que você pode executar, consulte a seção Abordar as descobertas de observabilidade mais adiante neste tópico.

Visibilidade da conformidade do Lambda

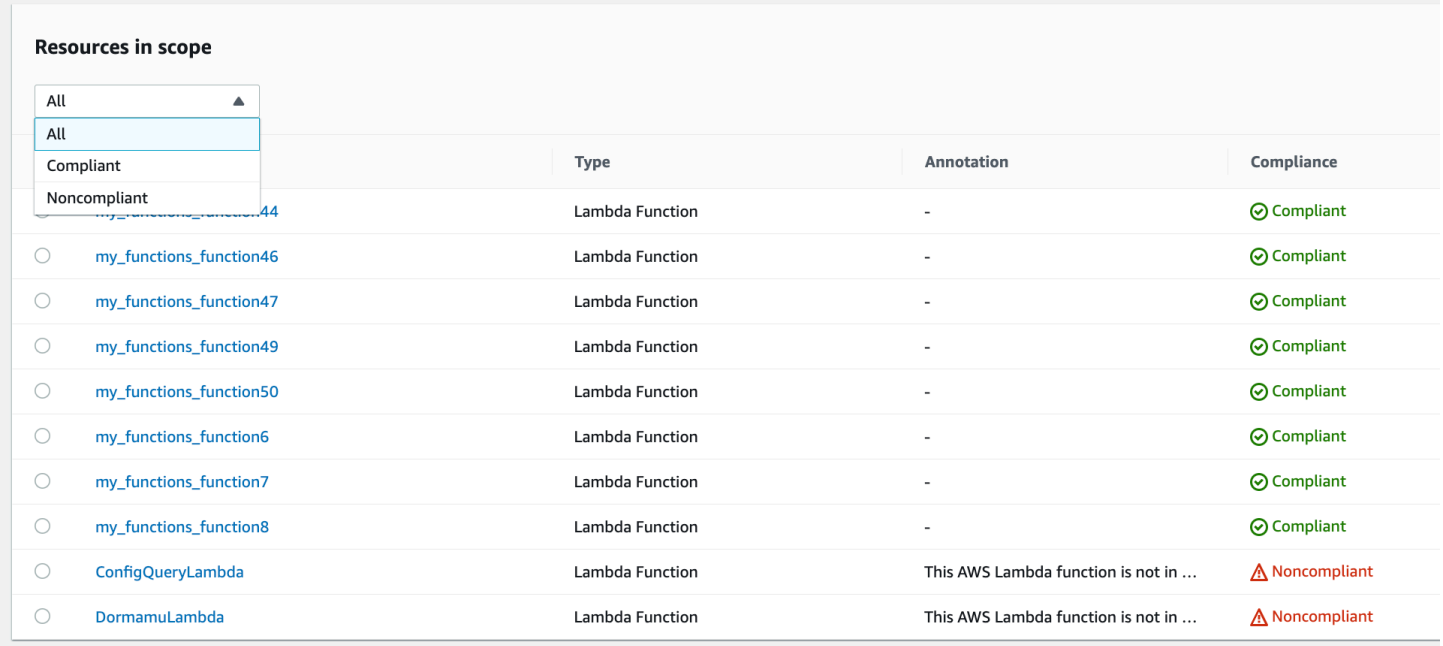
Com os dados gerados pelo AWS Config você pode criar painéis em nível organizacional para monitorar a conformidade. Isso permite o rastreamento e o monitoramento consistentes de:

- Pacotes de conformidade por pontuação de conformidade
- Regras por recursos fora de conformidade
- Compliance status (Status de conformidade)



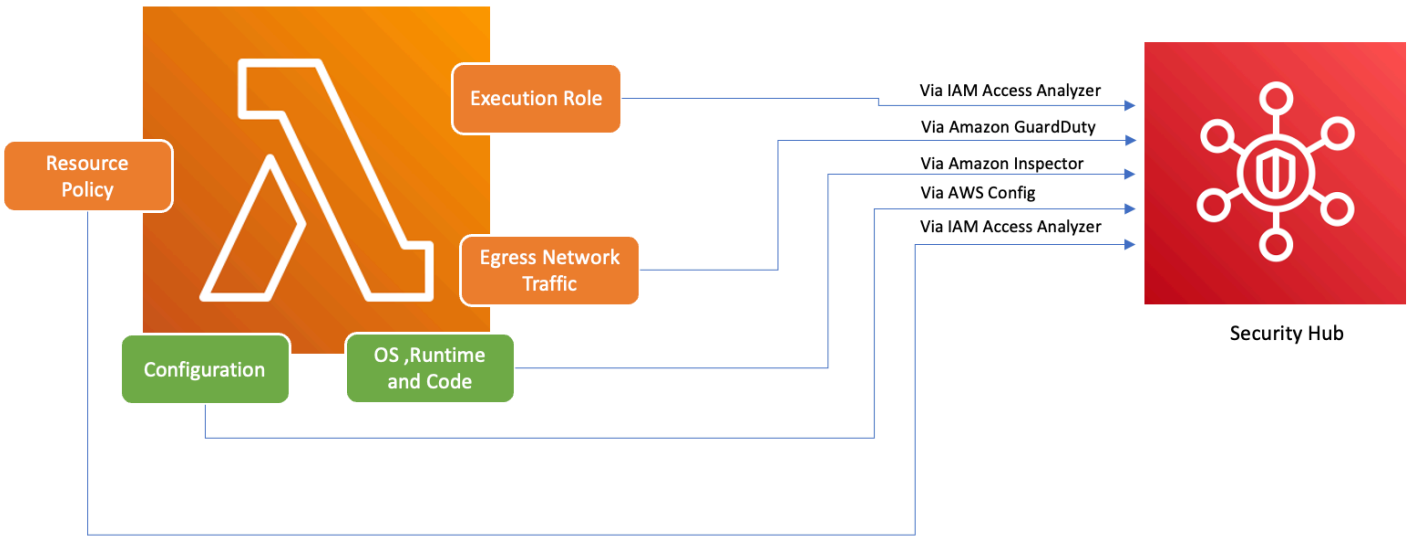
Verifique cada regra para identificar recursos fora de conformidade com esta regra. Por exemplo, se sua organização exigir que todas as funções do Lambda estejam associadas a uma VPC e se você

tiver implantado uma regra do AWS Config para identificar conformidade, você poderá selecionar a regra `lambda-inside-vpc` na lista acima.



Para obter mais informações sobre as ações que você pode executar, consulte a seção [Abordar as descobertas de observabilidade] a seguir.

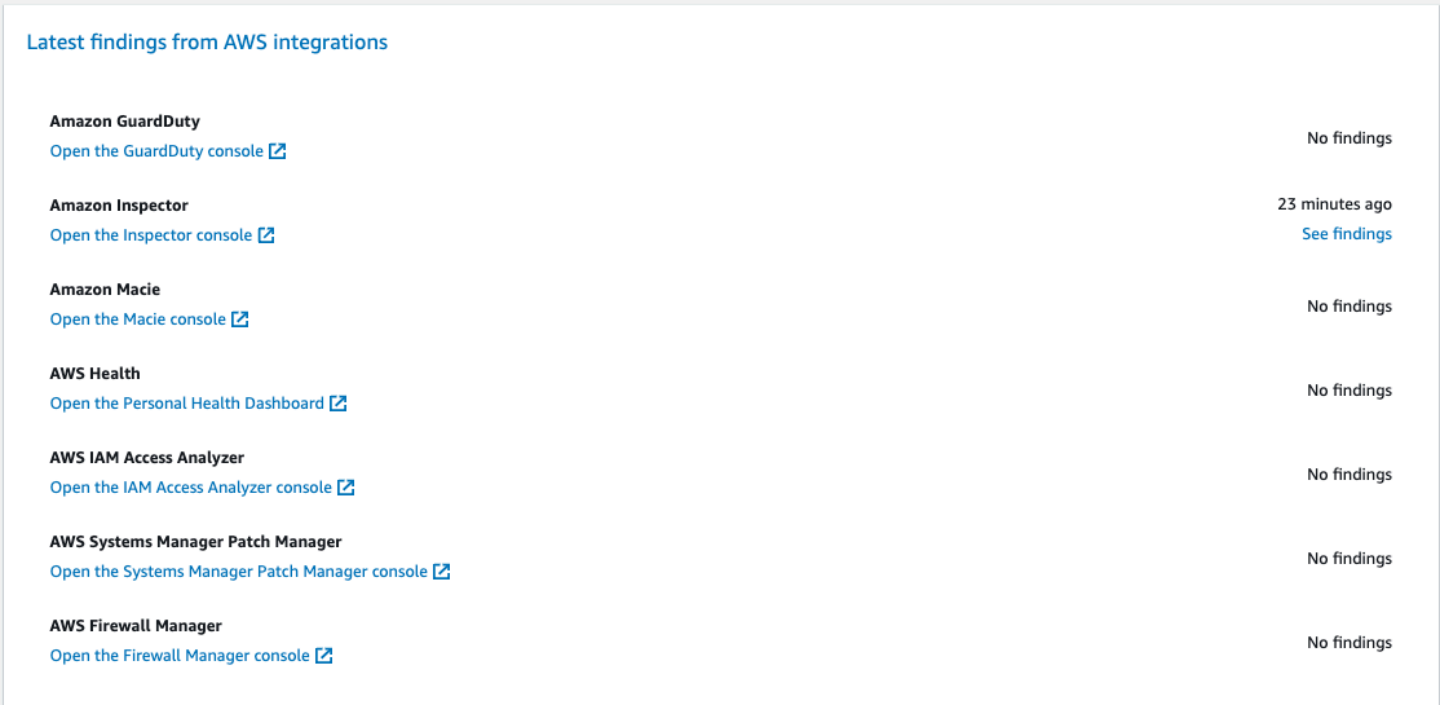
Visibilidade dos limites da função do Lambda usando o Security Hub



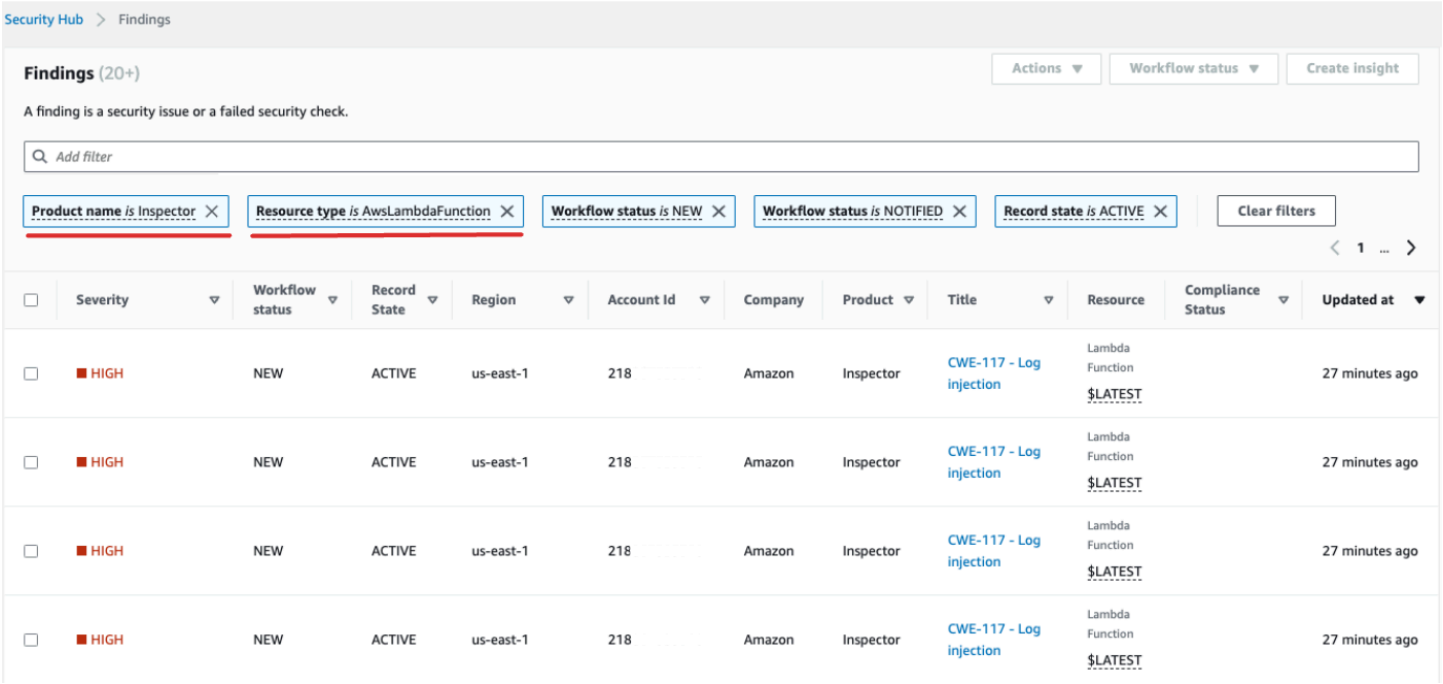
Para garantir que os serviços da AWS, incluindo o Lambda, sejam usados com segurança, a AWS introduziu as Práticas Recomendadas de Segurança Básica v1.0.0. Esse conjunto de melhores

práticas fornece diretrizes claras para proteger recursos e dados no ambiente da AWS, enfatizando a importância de manter uma postura de segurança forte. O AWS Security Hub complementa isso ao oferecer um centro unificado de segurança e conformidade. Ele agrega, organiza e prioriza as descobertas de segurança de vários serviços, como AWS Amazon Inspector e Amazon. AWS Identity and Access Management Access Analyzer GuardDuty

Se você tem o Security Hub, o Amazon Inspector, o IAM Access Analyzer e GuardDuty está habilitado em sua AWS organização, o Security Hub agrega automaticamente as descobertas desses serviços. Por exemplo, vamos considerar o Amazon Inspector. Usando o Security Hub, você pode identificar com eficiência as vulnerabilidades do código e do pacote nas funções do Lambda. No console do Security Hub, navegue até a seção inferior denominada Últimas descobertas das integrações da AWS. Aqui, você pode visualizar e analisar descobertas provenientes de vários serviços da AWS integrados.



Para ver os detalhes, escolha o link Ver descobertas na segunda coluna. É exibida uma lista de descobertas filtradas por produto, como o Amazon Inspector. Para limitar sua pesquisa às funções do Lambda, defina `ResourceType` como `AwsLambdaFunction`. Isso exibe descobertas do Amazon Inspector relacionadas às funções do Lambda.



Pois GuardDuty, você pode identificar padrões de tráfego de rede suspeitos. Essas anomalias podem sugerir a existência de código potencialmente malicioso na função do Lambda.

Com o IAM Access Analyzer, você pode verificar as políticas, especialmente aquelas com instruções de condição que concedem à função acesso a entidades externas. Além disso, o IAM Access Analyzer avalia as permissões definidas ao usar a AddPermissionoperação na API Lambda junto com um. `EventSourceToken`

Abordar as descobertas de observabilidade

Dadas as amplas configurações possíveis para as funções do Lambda e seus requisitos distintos, uma solução de automação padronizada para remediação talvez não seja adequada para todas as situações. Além disso, as mudanças são implementadas de forma diferente em vários ambientes. Se você encontrar alguma configuração que pareça fora de conformidade, considere as seguintes diretrizes:

1. Estratégia de marcação

   Recomendamos a implementação de uma estratégia abrangente de marcação. Cada função do Lambda deve ser marcada com informações essenciais, como:

   - Proprietário: a pessoa ou equipe responsável pela função.
   - Ambiente: produção, preparação, desenvolvimento ou sandbox.

- Aplicação: o contexto mais amplo ao qual essa função pertence, se aplicável.

2. Alcance do proprietário

   Em vez de automatizar as alterações significativas (como o ajuste da configuração da VPC), entre em contato proativamente com os proprietários das funções fora de conformidade (identificadas pela tag do proprietário), fornecendo a elas tempo suficiente para:

   - Ajustar configurações fora de conformidade nas funções do Lambda.
   - Fornecer uma explicação e solicitar uma exceção ou refinar os padrões de conformidade.

3. Manter um banco de dados de gerenciamento de configuração (CMDB)

   Embora as tags possam fornecer contexto imediato, manter um CMDB centralizado pode fornecer insights mais profundos. Ele pode conter informações mais granulares sobre cada função do Lambda, suas dependências e outros metadados essenciais. Um CMDB é um recurso inestimável para auditoria, verificações de conformidade e identificação de proprietários de funções.

À medida que o cenário da infraestrutura com tecnologia sem servidor evolui continuamente, é essencial adotar uma postura proativa em relação ao monitoramento. Com ferramentas, como o AWS Config, o Security Hub e o Amazon Inspector, possíveis anomalias ou configurações fora de conformidade podem ser rapidamente identificadas. No entanto, as ferramentas não podem, isoladamente, garantir total conformidade ou configurações ideais. É fundamental combinar essas ferramentas com processos e melhores práticas bem documentados.

- Ciclo de feedback: depois que as etapas de remediação forem executadas, certifique-se de que haja um ciclo de feedback. Isso significa revisitar periodicamente os recursos fora de conformidade para confirmar se eles foram atualizados ou se ainda estão em execução com os mesmos problemas.
- Documentação: sempre documente as observações, as ações executadas e quaisquer exceções concedidas. A documentação adequada não só ajuda durante as auditorias, mas também ajuda a aprimorar o processo para melhorar a conformidade e a segurança no futuro.
- Treinamento e reconhecimento: garanta que todos os investidores, especialmente os proprietários de funções do Lambda, sejam regularmente treinados e informados sobre as melhores práticas, políticas organizacionais e exigências de conformidade. Workshops, webinars ou sessões de treinamento regulares podem ajudar muito a garantir que todos estejam na mesma página quando se trata de segurança e conformidade.

Para concluir, embora ferramentas e tecnologias forneçam recursos consistentes para detectar e identificar possíveis problemas, o elemento humano, incluindo compreensão, comunicação, treinamento e documentação, continua sendo fundamental. Esses aspectos, juntos, formam uma combinação poderosa para garantir que as funções do Lambda e uma infraestrutura mais ampla permaneçam em conformidade, seguras e otimizadas para suas necessidades de negócios.

Validação de conformidade do AWS Lambda

Auditores de terceiros avaliam a segurança e a conformidade do AWS Lambda como parte de vários programas de conformidade da AWS. Isso inclui SOC, PCI, FedRAMP, HIPAA e outros.

Para obter uma lista dos serviços da AWS no escopo de programas de conformidade específicos, consulte Serviços da AWS no escopo por programa de conformidade. Para obter informações gerais, consulte Programas de conformidade da AWS.

É possível baixar relatórios de auditoria de terceiros usando o AWS Artifact. Para obter mais informações, consulte Downloading reports in AWS Artifact.

Sua responsabilidade de conformidade ao usar o Lambda é determinada pela confidencialidade dos seus dados, pelos objetivos de conformidade da sua empresa e pelas leis e regulamentos aplicáveis. Você pode implementar controles de governança para garantir que as funções do Lambda da sua empresa atendam aos seus requisitos de conformidade. Para ter mais informações, consulte Criar uma estratégia de governança para funções e camadas do Lambda.

Resiliência no AWS Lambda

A infraestrutura global da AWS é criada com base em AWS Regiões e Zonas de Disponibilidade. AWS As regiões fornecem várias zonas de disponibilidade separadas e isoladas fisicamente, as quais são conectadas com baixa latência, alto throughput e redes altamente redundantes. Com as zonas de disponibilidade, é possível projetar e operar aplicações e bancos de dados que executam o failover automaticamente entre as zonas de disponibilidade sem interrupção. As zonas de disponibilidade são mais altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de data center tradicionais.

Para obter mais informações sobre AWS Regiões e Zonas de Disponibilidade, consulte AWS global infrastructure.

Além da infraestrutura global da AWS, o Lambda oferece vários recursos para ajudar a oferecer suporte às suas necessidades de resiliência de dados e backup.

- Versionamento: você pode usar o versionamento no Lambda para salvar a configuração e o código da sua função à medida que os desenvolve. Junto com aliases, você pode usar o versionamento para executar implantações azul/verde e contínuas. Para obter detalhes, consulte Versões da função do Lambda.

- Escalabilidade— quando a função recebe uma solicitação enquanto está processando uma solicitação anterior, o Lambda executa outra instância da função para lidar com o aumento de carga. O Lambda é dimensionado automaticamente para lidar com 1.000 execuções simultâneas por região, umquotaque pode ser aumentada, se necessário. Para obter detalhes, consulte Como entender a escalabilidade da função do Lambda.
- Alta disponibilidade: o Lambda executa sua função em várias zonas de disponibilidade para garantir que ela esteja disponível para processar eventos no caso de uma interrupção do serviço em uma única zona. Se você configurar a função para se conectar a uma nuvem privada virtual (VPC) na sua conta, especifique sub-redes em várias zonas de disponibilidade para garantir uma alta disponibilidade. Para obter detalhes, consulte Conceder acesso a funções do Lambda para recursos em uma Amazon VPC.
- Simultaneidade reservada: para garantir que sua função sempre pode se dimensionar para manipular solicitações adicionais, você pode reservar simultaneidade para ela. Configurar a simultaneidade reservada para uma função garante que ela pode se dimensionar, mas não exceder, um número especificado de invocações simultâneas. Isso garante que você não perderá solicitações por causa de outras funções que estejam consumindo toda a simultaneidade disponível. Para obter detalhes, consulte Configurar a simultaneidade reservada para uma função.
- Tentativas: para invocações assíncronas e um subconjunto de invocações acionadas por outros serviços, o Lambda faz novas tentativas automaticamente sobre um erro com um atraso entre as tentativas. Outros clientes e serviços da AWS que invocam funções de forma síncrona são responsáveis por realizar novas tentativas. Para obter detalhes, consulte Compreender o comportamento de novas tentativas no Lambda.
- Fila de mensagens mortas: para invocações assíncronas, você pode configurar o Lambda para enviar solicitações para uma fila de mensagens mortas caso todas as tentativas falhem. Uma fila de mensagens mortas é um tópico do Amazon SNS ou uma fila do Amazon SQS que recebe eventos para solução de problemas ou reprocessamento. Para obter mais detalhes, consulte Filas de mensagens mortas.

Segurança da infraestrutura no AWS Lambda

Por ser um serviço gerenciado, o AWS Lambda é protegido pela segurança da rede global da AWS. Para obter informações sobre AWS serviços de segurança da AWS e como a protege a infraestrutura, consulte AWSSegurança na Nuvem . Para projetar o ambiente da AWS utilizando as práticas recomendadas de segurança de infraestrutura, consulte Proteção de infraestrutura em Pilar segurança: AWS Well-Architected Framework.

Você usa chamadas de API publicadas pela AWS para acessar o Lambda por meio da rede. Os clientes precisam oferecer suporte para:

- Transport Layer Security (TLS). Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Conjuntos de criptografia com sigilo de encaminhamento perfeito (perfect forward secrecy, ou PFS) como DHE (Ephemeral Diffie-Hellman, ou Efêmero Diffie-Hellman) ou ECDHE (Ephemeral Elliptic Curve Diffie-Hellman, ou Curva elíptica efêmera Diffie-Hellman). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Além disso, as solicitações devem ser assinadas utilizando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou você pode usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Monitoramento e solução de problemas de funções do Lambda

O AWS Lambda pode ser integrado a outros serviços da AWS para ajudar você a monitorar e solucionar problemas de suas funções do Lambda. O Lambda monitora automaticamente as funções do Lambda em seu nome e gera relatórios sobre métricas por meio do Amazon CloudWatch. Para ajudar você a monitorar seu código quando ele estiver sendo executado, o Lambda rastreia automaticamente o número de solicitações, a duração de chamadas por solicitação e o número de solicitações que resultam em erro.

Você pode usar outros serviços do AWS para solucionar problemas de suas funções do Lambda. Esta seção descreve como usar esses serviços da AWS para monitorar, rastrear, depurar e solucionar problemas de suas funções e aplicações do Lambda. Para obter detalhes sobre o registro em log de funções e erros em cada runtime, consulte as seções individuais de runtime.

Para obter mais informações sobre monitoramento de aplicações do Lambda, consulte Monitoring and observability no Serverless Land.

Seções

Monitorar funções no console do Lambda

O serviço Lambda monitora as funções em seu nome e envia métricas para a Amazon. CloudWatch O console do Lambda cria gráficos para monitoramento dessas métricas e os mostra na página Monitoring (Monitoramento) para cada função do Lambda.

O console do Lambda fornece uma visualização em painel único de métricas, logs e rastreamentos. O console fornece filtros para intervalo de horário, fuso horário e opções de atualização que se aplicam universalmente a todos os painéis. Você pode correlacionar facilmente métricas, logs e rastreamentos, reduzindo o tempo médio de recuperação (MTTR) ao solucionar erros em suas funções do Lambda.

Definição de preço

CloudWatch tem um nível gratuito perpétuo. Além do limite do nível gratuito, CloudWatch cobramos por métricas, painéis, alarmes, registros e insights. Para obter mais informações, consulte os CloudWatch preços da Amazon.

Usar o console do Lambda

Você pode monitorar as funções e as aplicações do Lambda no console do Lambda.

Como monitorar uma função

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha a guia Monitor (Monitorar).

Tipos de gráficos de monitoramento

A seção a seguir descreve os gráficos de monitoramento do console do Lambda.

Gráficos de monitoramento do Lambda

- Invocations (Invocações): o número de vezes que a função foi invocada.
- Duration (Duração): quantidade de tempo médio, mínimo e máximo que o código da função leva para processar um evento.

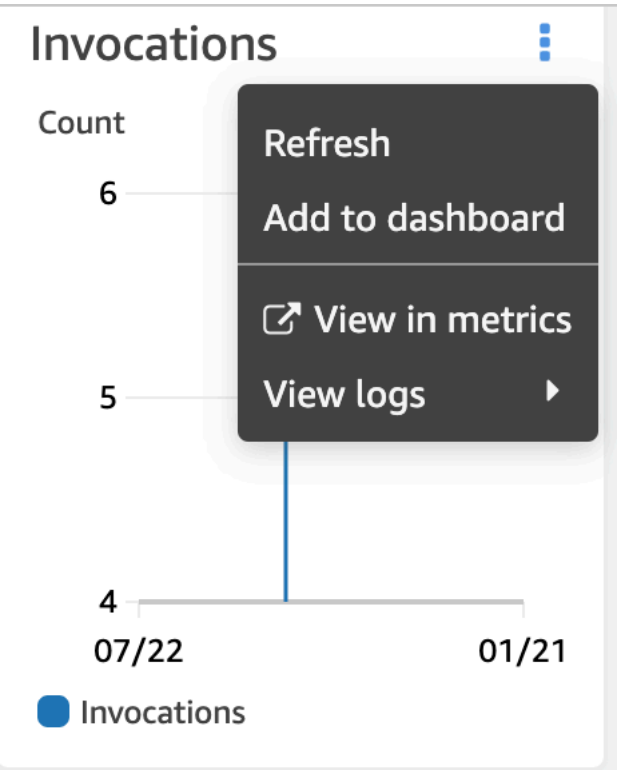
- Error count and success rate (%) (Contagem de erros e taxa de êxito [%]): o número de erros e a porcentagem de invocações concluídas sem erro.
- Throttles (Controles de utilização): o número de vezes que a execução falhou devido a limites de simultaneidade.
- IteratorAge— Para fontes de eventos de stream, a idade do último item no lote quando o Lambda o recebeu e invocou a função.
- Async delivery failures (Falhas de entrega assíncrona): o número de erros que ocorreram quando o Lambda tentou gravar em um destino ou em uma fila de mensagens mortas.
- Concurrent executions (Execuções simultâneas): o número de instâncias da função que estão processando eventos.

Visualizando gráficos no console do Lambda

A seção a seguir descreve como visualizar gráficos de CloudWatch monitoramento no console Lambda e abrir CloudWatch o painel de métricas.

Para visualizar gráficos de monitoramento de uma função

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha a guia Monitor (Monitorar).
4. Escolha entre os intervalos de tempo predefinidos ou escolha um intervalo de tempo personalizado.
5. Para ver a definição de um gráfico CloudWatch, escolha os três pontos verticais (ações do widget) e, em seguida, escolha Exibir em métricas para abrir o painel Métricas no CloudWatch console.

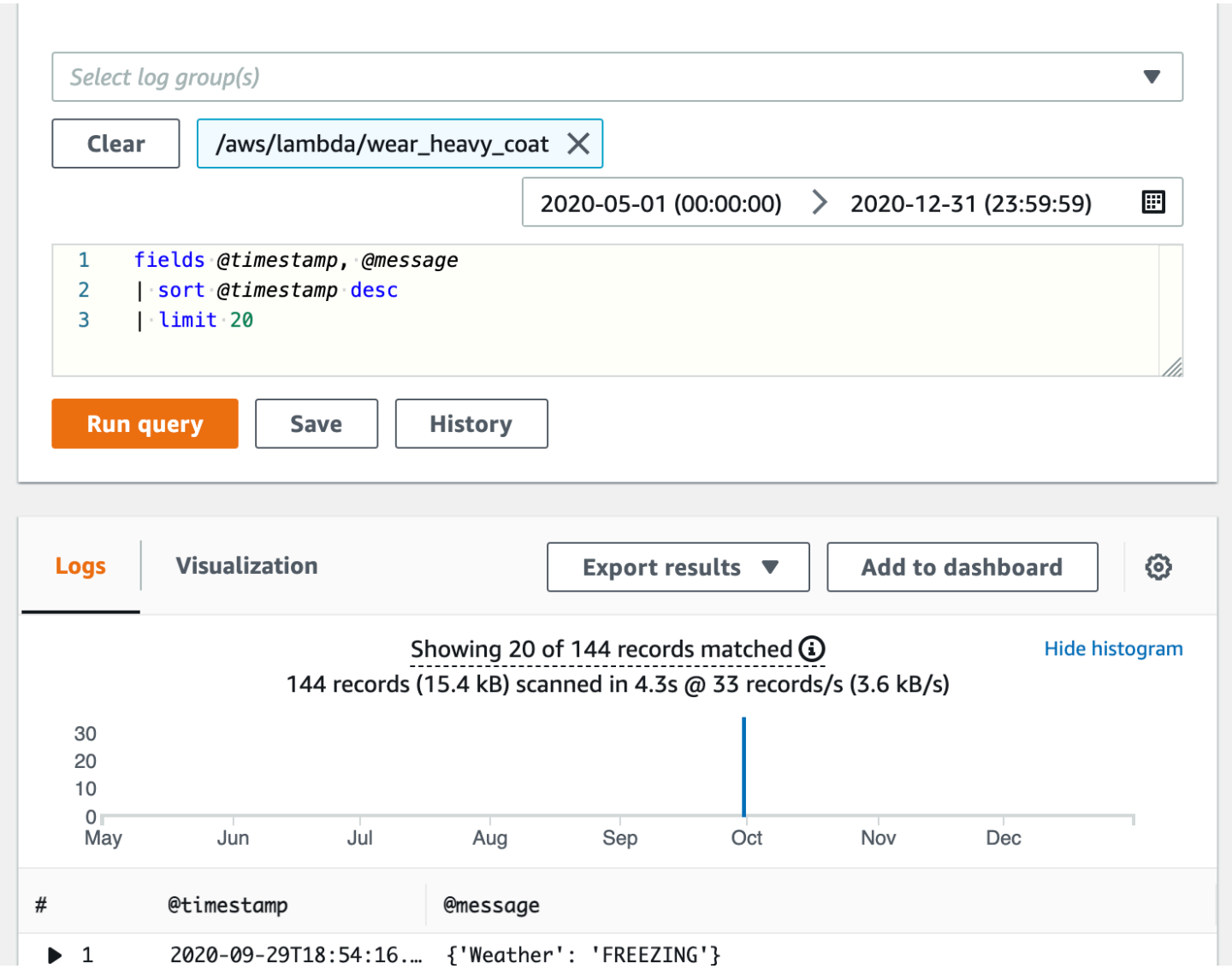


Visualização de consultas no console de CloudWatch registros

A seção a seguir descreve como visualizar e adicionar relatórios do CloudWatch Logs Insights a um painel personalizado no console do CloudWatch Logs.

Para exibir relatórios de uma função

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha a guia Monitor (Monitorar).
4. Escolha Exibir logins CloudWatch.
5. Selecione Exibir no Logs Insights.
6. Escolha entre os intervalos de tempo predefinidos ou escolha um intervalo de tempo personalizado.
7. Selecione Executar consulta.
8. (Opcional) Selecione Salvar.



Próximas etapas

- Saiba mais sobre as métricas que o Lambda registra CloudWatch e envia. Trabalhar com métricas de funções Lambda
- Saiba como usar o CloudWatch Lambda Insights para coletar e agregar métricas e logins de desempenho de tempo de execução da função Lambda. Monitorar a performance de funções com o Lambda Insights do Amazon CloudWatch

Trabalhar com métricas de funções Lambda

Quando a função do AWS Lambda termina o processamento de um evento, o Lambda envia métricas sobre a invocação para o Amazon CloudWatch. Essas métricas não são cobradas.

No console do CloudWatch, você pode criar grafos e painéis com essas métricas. É possível definir alarmes para responder a alterações na utilização, na performance ou nas taxas de erro. O Lambda envia dados de métricas ao CloudWatch em intervalos de um minuto. Para obter insights mais imediatos sobre a sua função do Lambda, você pode criar métricas personalizadas de alta resolução, conforme descrito no Serverless Land. Aplicam-se cobranças para métricas personalizadas e alarmes do CloudWatch. Para obter mais informações, consulte Amazon CloudWatch Pricing (Preços do Amazon CloudWatch).

Esta página descreve as métricas de invocação, performance e simultaneidade da função do Lambda disponíveis no console do CloudWatch.

Seções

Exibir métricas no console do CloudWatch

Você pode usar o console do CloudWatch para filtrar e classificar métricas de funções por nome, alias ou versão da função.

Para exibir métricas no console do CloudWatch

1. Abra a página Metrics (Métricas) (namespace `AWS/Lambda`) no console do CloudWatch.
2. Na guia Procurar, em Métricas, escolha qualquer uma das seguintes dimensões:
   - Por nome da função (`FunctionName`): visualize métricas agregadas para todas as versões e aliases de uma função.
   - Por recurso (`Resource`): visualize métricas para uma versão ou um alias de uma função.
   - Por versão executada (`ExecutedVersion`): visualize métricas para uma combinação de alias e versão. Use a dimensão `ExecutedVersion` para comparar taxas de erro para duas versões de uma função que são ambas destinos de um alias ponderado.

- Em todas as funções (nenhum): visualize métricas agregadas para todas as funções na Região da AWS atual.

3. Escolha uma métrica e escolha Adicionar ao gráfico ou outra opção gráfica.

Por padrão, os gráficos usam a estatística `Sum` para todas as métricas. Para escolher uma estatística diferente e personalizar o gráfico, use as opções na guia Graphed metrics (Métricas no gráfico).

> **Note**
>
> O timestamp em uma métrica reflete quando a função foi invocada. Dependendo da duração da invocação, isso pode representar vários minutos até a métrica ser emitida. Se, por exemplo, a função tiver um tempo limite de dez minutos, para obter métricas precisas, procure mais de dez minutos no passado.

Para obter mais informações sobre o CloudWatch, consulte o Guia do usuário do Amazon CloudWatch.

Tipos de métricas

A seção a seguir descreve os tipos de métricas do Lambda disponíveis no console do CloudWatch.

Métricas de invocação

As métricas de invocação são indicadores binários do resultado de uma invocação da função do Lambda. Por exemplo, se a função retornar um erro, o Lambda enviará a métrica `Errors` com um valor de 1. Para obter uma contagem do número de erros de função que ocorrem a cada minuto, visualize o `Sum` da métrica `Errors` com o período de 1 minuto.

> **Note**
>
> Visualize as métricas a seguir com a estatística `Sum`.

- `Invocations`: o número de vezes em que o código da sua função foi chamado, incluindo invocações bem-sucedidas e invocações que resultem em um erro de função. As invocações não serão registradas se a solicitação de invocação tiver controle de utilização ou se resultar em erro de invocação. O valor de `Invocations` é igual ao número de solicitações faturadas.

- `Errors`: o número de invocações que resultam em um erro de função. Os erros de função incluem exceções emitidas pelo código e exceções emitidas pelo runtime do Lambda. O runtime retorna um erro para problemas como tempos limite e erros de configuração. Para calcular a taxa de erro, divida o valor de `Errors` pelo valor de `Invocations`. Observe que o timestamp em uma métrica de erro reflete quando a função foi chamada, não quando o erro ocorreu.
- `DeadLetterErrors`: para a invocação assíncrona, o número de vezes em que o Lambda tenta enviar um evento para uma fila de mensagens não entregues (DLQ), mas falha. Os erros de mensagens não entregues podem ocorrer devido a recursos configurados incorretamente ou limites de tamanho.
- `DestinationDeliveryFailures`: para invocação assíncrona e mapeamentos da origem do evento compatíveis, o número de vezes em que o Lambda tenta enviar um evento a um destino, mas falha. Para mapeamentos da origem do evento, o Lambda oferece suporte a origens de fluxo (DynamoDB e Kinesis). Os erros de entrega podem ocorrer devido a erros de permissão, recursos configurados incorretamente ou limites de tamanho. Os erros também podem acontecer se o destino que você configurou incluir um tipo incompatível, como uma fila do Amazon SQS FIFO ou um tópico do Amazon SNS FIFO.
- `Throttles`: o número de solicitações de invocação que são limitadas. Quando todas as instâncias da função estão processando solicitações e nenhuma simultaneidade está disponível para aumento de escala vertical, o Lambda rejeita solicitações adicionais com um erro `TooManyRequestsException`. As solicitações limitadas e outros erros de invocação não contam como `Invocations` ou `Errors`.
- `OversizedRecordCount`: para origens de eventos do Amazon DocumentDB, o número de eventos que sua função recebe do stream de alterações com mais de 6 MB de tamanho. O Lambda descarta a mensagem e emite essa métrica.
- `ProvisionedConcurrencyInvocations`: o número de vezes em que o código da sua função é invocado em simultaneidade provisionada.
- `ProvisionedConcurrencySpilloverInvocations`: o número de vezes em que o código da sua função é invocado usando a simultaneidade padrão quando toda a simultaneidade provisionada está em uso.
- `RecursiveInvocationsDropped`: o número de vezes que o Lambda interrompeu a invocação da função porque detectou que ela era parte de um loop recursivo infinito. A Usar a detecção de loop recursivo do Lambda para evitar loops infinitos monitora quantas vezes uma função é invocada como parte de uma cadeia de solicitações ao rastrear metadados adicionados por AWS SDKs compatíveis. Se a função for invocada mais de 16 vezes como parte de uma cadeia de solicitações, o Lambda descartará a próxima invocação.

Métricas de performance

As métricas de performance fornecem detalhes de performance sobre uma única invocação de função. Por exemplo, a métrica `Duration` indica a quantidade de tempo, em milissegundos, que a função gasta processando um evento. Para ter uma ideia da velocidade de processamento de eventos da função, visualize essas métricas com a estatística `Average` ou `Max`.

- `Duration`: quantidade de tempo que o código da função gasta processando um evento. A duração faturada de uma invocação é o valor da `Duration` arredondado para o milissegundo mais próximo. A `Duration` não inclui o tempo de inicialização a frio.
- `PostRuntimeExtensionsDuration`: a quantidade cumulativa de tempo que o runtime gasta executando o código para extensões após a conclusão do código de função.
- `IteratorAge`: para origens de eventos do DynamoDB, Kinesis e Amazon DocumentDB, a data do último registro no evento. Essa métrica mede o tempo transcorrido entre quando um stream recebe o registro e quando o mapeamento da origem do evento envia o evento à função.
- `OffsetLag`: para origens de eventos do Apache Kafka autogerenciado e do Amazon Managed Streaming for Apache Kafka (Amazon MSK), a diferença de deslocamento entre o último registro gravado em um tópico e o último registro que o grupo de consumidores da sua função processou. Embora um tópico do Kafka possa ter várias partições, essa métrica mensura o atraso de deslocamento no nível do tópico.

`Duration` também é compatível com estatísticas de percentil (p). Use os percentis para excluir valores discrepantes que distorcem estatísticas `Average` e `Maximum`. Por exemplo, a estatística `p95` mostra a duração máxima de 95% das invocações, excluindo as 5% mais lentas. Para obter mais informações, consulte Percentis no Guia do usuário do Amazon CloudWatch.

Métricas de simultaneidade

O Lambda relata métricas de simultaneidade como uma contagem agregada do número de instâncias que estão processando eventos em uma função, uma versão, um alias ou uma Região da AWS. Para ver se você está próximo de atingir os limites de simultaneidade, visualize essas métricas com a estatística `Max`.

- `ConcurrentExecutions`: o número de instâncias da função que estão processando eventos. Se esse número atingir a cota de execuções simultâneas para a região ou o limite de simultaneidade reservada na função, o Lambda controlará a utilização de solicitações de invocação adicionais.

- `ProvisionedConcurrentExecutions`: o número de instâncias da função que estão processando eventos usando a simultaneidade provisionada. Para cada invocação de um alias ou versão com simultaneidade provisionada, o Lambda emite a contagem atual.
- `ProvisionedConcurrencyUtilization`: para uma versão ou um alias, o valor de `ProvisionedConcurrentExecutions` dividido pela quantidade total de simultaneidade provisionada configurada. Por exemplo, se você configurar uma simultaneidade provisionada de 10 para sua função e seu `ProvisionedConcurrentExecutions` for 7, seu `ProvisionedConcurrencyUtilization` será 0,7.
- `UnreservedConcurrentExecutions`: para uma região, o número de eventos que estão sendo processados por funções que não tenham simultaneidade reservada.
- `ClaimedAccountConcurrency`: para uma região, quantidade de simultaneidade que não está disponível para invocações sob demanda. `ClaimedAccountConcurrency` é igual a `UnreservedConcurrentExecutions` mais a quantidade de simultaneidade alocada (ou seja, a simultaneidade total reservada mais a simultaneidade total provisionada). Para ter mais informações, consulte Trabalhar com a métrica `ClaimedAccountConcurrency`.

Métricas de invocação assíncrona

As métricas de invocação assíncrona fornecem detalhes sobre invocações assíncronas de fontes de eventos e invocações diretas. Você pode definir limites e alarmes para notificá-lo sobre determinadas alterações. Por exemplo, quando há um aumento indesejado no número de eventos enfileirados para processamento (`AsyncEventsReceived`). Ou quando um evento está esperando há muito tempo para ser processado (`AsyncEventAge`).

- `AsyncEventsReceived`: o número de eventos que o Lambda enfileira com sucesso para processamento. Esta métrica fornece informações sobre o número de eventos que uma função do Lambda recebe. Monitore essa métrica e defina alarmes de limites para verificar problemas. Por exemplo, para detectar um número indesejável de eventos enviados ao Lambda e diagnosticar rapidamente problemas resultantes de configurações incorretas de gatilhos ou funções. As incompatibilidades entre os `AsyncEventsReceived` e as `Invocations` podem indicar uma disparidade no processamento, a eliminação de eventos ou um possível backlog na fila.
- `AsyncEventAge`: o tempo entre o momento em que o Lambda enfileira com sucesso o evento e o momento em que a função é invocada. O valor dessa métrica aumenta quando os eventos estão sendo repetidos devido a falhas de invocação ou no controle de utilização. Monitore esta métrica e defina alarmes para limites em estatísticas diferentes para quando ocorrer um acúmulo de fila.

Para solucionar um aumento nessa métrica, examine a métrica `Errors` para identificar erros de função e a métrica `Throttles` para identificar problemas de simultaneidade.

- `AsyncEventsDropped`: o número de eventos que são eliminados sem executar a função com êxito. Se você configurar uma fila de mensagens não entregues (DLQ) ou um destino `OnFailure`, os eventos serão enviados para lá antes de serem descartados. Os eventos são encerrados por vários motivos. Por exemplo, os eventos podem exceder a idade máxima do evento, esgotar o máximo de tentativas ou a simultaneidade reservada pode ser definida como 0. Para entender por que os eventos são encerrados, confira a métrica `Errors` para identificar erros de função e a métrica `Throttles` para identificar problemas de simultaneidade.

Usar logs do Amazon CloudWatch com o AWS Lambda

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome para ajudar a solucionar falhas nas funções. Desde que o perfil de execução da função tenha as permissões necessárias, o Lambda captura os logs de todas as solicitações tratadas pela função e os envia para o Amazon CloudWatch Logs.

Você pode inserir instruções de registro em log no seu código para ajudá-lo a validar se o seu código está funcionando conforme o esperado. O Lambda se integra automaticamente com o CloudWatch Logs e envia todos os logs do seu código a um grupo de logs do CloudWatch associado a uma função do Lambda.

Por padrão, o Lambda envia logs a um grupo de logs denominado `/aws/lambda/<function name>`. Caso queira que a função envie logs para outro grupo, você precisará configurar isso usando o console do Lambda, a AWS Command Line Interface (AWS CLI) ou a API do Lambda. Para saber mais, consulte the section called “Configurar grupos de logs do CloudWatch”.

Você pode exibir registros de funções do Lambda usando o console do Lambda, o console do CloudWatch, oAWS Command Line Interface(AWS CLI) ou a API do CloudWatch.

> **Note**
>
> Pode levar de 5 a 10 minutos para que os logs apareçam após uma invocação de função.

Seção

Pré-requisitos

Sua função de execução precisa de permissão para fazer o upload de registros no CloudWatch Logs. Você pode adicionar permissões do CloudWatch Logs usando uma política gerenciada da `AWSLambdaBasicExecutionRole` AWS fornecida pelo Lambda. Para adicionar essa política à sua função, execute o seguinte comando:

```
aws iam attach-role-policy --role-name your-role --policy-arn arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole
```

Para ter mais informações, consulte the section called “Políticas gerenciadas AWS”.

Definição de preço

Não há nenhuma cobrança adicional para usar os logs do Lambda, porém, a cobrança padrão do CloudWatch Logs é aplicável. Para obter mais informações, consulte Preço do CloudWatch.

Configurar controles avançados de registro em log para a função do Lambda

Para dar mais controle sobre como os logs das funções são capturados, processados e consumidos, o Lambda oferece as seguintes opções de configuração de log:

- Formato do log: selecione entre texto simples e formato JSON estruturado para os logs da sua função.
- Nível de log: para logs JSON estruturados, escolha o nível de detalhe dos logs enviados pelo Lambda para o CloudWatch, como ERROR, DEBUG ou INFO.
- Grupo de logs: escolha o grupo de logs do CloudWatch para o qual sua função envia logs.

Configurar logs em formato de texto simples e JSON

Capturar as saídas de log como pares de valores-chave JSON facilita a pesquisa e a filtragem ao depurar suas funções. Com os logs em formato JSON, você também pode adicionar tags e informações contextuais aos logs. Esse recurso ajuda a realizar análises automatizadas de grandes volumes de dados de log. A menos que o fluxo de trabalho de desenvolvimento dependa de ferramentas existentes que consumam logs do Lambda em texto simples, recomendamos que você selecione JSON para o formato de log.

Para todos os runtimes gerenciados pelo Lambda, você pode escolher se os logs do sistema da função são enviados para o CloudWatch Logs em texto simples não estruturado ou no formato JSON. Os logs do sistema são os logs que o Lambda gera e, às vezes, são conhecidos como logs de eventos da plataforma.

Para runtimes compatíveis, quando você usa um dos métodos de registro em log integrado compatível, o Lambda também pode gerar os logs de aplicações da função (os logs gerados pelo código da função) no formato JSON estruturado. Quando você configura o formato de log da função para esses runtimes, a configuração escolhida se aplica aos logs do sistema e de aplicações.

Para runtimes compatíveis, se a função usar uma biblioteca ou um método de registro em log compatível, você não precisa fazer nenhuma alteração no código existente para que o Lambda capture registros em JSON estruturado.

> **Note**
>
> O uso de log no formato JSON adiciona metadados e codifica mensagens de log como objetos JSON contendo uma série de pares de valores-chave. Por causa disso, o tamanho das mensagens de log da função pode aumentar.

Runtimes e métodos de registro em log compatíveis

Atualmente, o Lambda é compatível com a opção de gerar logs de aplicações em JSON estruturado para os runtimes a seguir.

Runtime	Versões compatíveis
Java	Todos os runtimes do Java, exceto Java 8 no Amazon Linux 1
Node.js	Node.js 16 e posterior
Python	Python 3.7 e posterior

Para que o Lambda envie os logs de aplicações da função para o CloudWatch no formato JSON estruturado, sua função precisa usar as seguintes ferramentas de registro em log integradas para gerar logs:

- Java: o logger `LambdaLogger` ou Log4j2.
- Node.js: os métodos do console `console.trace`, `console.debug`, `console.log`, `console.info`, `console.error` e `console.warn`.
- Python: a biblioteca `logging` padrão do Python

Para obter mais informações sobre o uso de controles avançados de registro em log com runtimes compatíveis, consulte the section called “Registro em log”, the section called “Registro em log” e the section called “Registro em log”.

Para outros runtimes gerenciados do Lambda, ele atualmente só tem compatibilidade nativa com a captura de logs do sistema no formato JSON estruturado. No entanto, você ainda pode capturar logs de aplicações no formato JSON estruturado em qualquer runtime usando ferramentas de registro em log, como o Powertools para AWS Lambda, que gera logs no formato JSON.

Formatos de log padrão

Atualmente, o formato de log padrão para todos os runtimes do Lambda é texto simples.

Se você já usa bibliotecas de registro em log, como o Powertools para AWS Lambda, para gerar logs de funções no formato JSON estruturado, não precisará alterar o código se selecionar log no formato JSON. O Lambda não codifica duas vezes nenhum log que já esteja codificado em JSON. Então, os logs de aplicações da função continuarão sendo capturados como antes.

Formato JSON para logs do sistema

Quando você configura o formato de log da função como JSON, cada item de log do sistema (evento de plataforma) é capturado como um objeto JSON que contém pares de valores-chave com as seguintes chaves:

- `"time"`: o horário em que a mensagem de log foi gerada.
- `"type"`: o tipo de evento que está sendo registrado em log
- `"record"`: o conteúdo do log gerado

O formato do valor `"record"` varia de acordo com o tipo de evento que está sendo registrado em log. Para ter mais informações, consulte the section called “Tipos de objeto `Event` da API de telemetria”. Para obter mais informações sobre os níveis de log atribuídos aos eventos de log do sistema, consulte the section called “Mapeamento de eventos no nível de log do sistema”.

Para comparação, os dois exemplos a seguir mostram a mesma saída de log nos formatos de texto simples e JSON estruturado. Observe que, na maioria dos casos, os eventos de log do sistema contêm mais informações quando enviados no formato JSON do que quando enviados em texto simples.

Example texto simples:

```
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 estruturado:

```
{
  "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**
>
> the section called “API de telemetria” sempre emite eventos de plataforma, como `START` e `REPORT` no formato JSON. Configurar o formato dos logs do sistema enviados pelo Lambda para o CloudWatch não afeta o comportamento da API de Telemetria do Lambda.

Formato JSON para logs de aplicações

Quando você configura o formato de log da função como JSON, as saídas de logs de aplicações gravadas usando bibliotecas e métodos de registro em log compatíveis são capturadas como um objeto JSON que contém pares de valores de chave com as chaves a seguir.

- `"timestamp"`: o horário em que a mensagem de log foi gerada.
- `"level"`: o nível de log atribuído à mensagem.

- `"message"`: o conteúdo da mensagem de log.
- `"requestId"` (Python e Node.js) ou `"AWSrequestId"` (Java): o ID de solicitação exclusivo para a invocação da função

Dependendo do runtime e do método de registro em log que a função usa, esse objeto JSON também pode conter pares de chaves adicionais. Por exemplo, em Node.js, se a função usa métodos do `console` para registrar em log objetos de erro usando vários argumentos, o objeto JSON conterá pares adicionais de valores de chave com as chaves `errorMessage`, `errorType` e `stackTrace`. Para saber mais sobre logs formatados em JSON em diferentes runtimes do Lambda, consulte the section called “Registro em log”, the section called “Registro em log” e the section called “Registro em log”.

> **Note**
>
> A chave usada pelo Lambda para o valor do carimbo de data/hora é diferente para logs do sistema e de aplicações. Para logs do sistema, o Lambda usa a chave `"time"` para manter a consistência com a API de telemetria. Para logs de aplicações, o Lambda segue as convenções dos runtimes compatíveis e usa `"timestamp"`.

Para comparação, os dois exemplos a seguir mostram a mesma saída de log nos formatos de texto simples e JSON estruturado.

Example texto simples:

```
2023-10-27T19:17:45.586Z 79b4f56e-95b1-4643-9700-2807f4e68189 INFO some log message
```

Example JSON estruturado:

```
{
    "timestamp":"2023-10-27T19:17:45.586Z",
    "level":"INFO",
    "message":"some log message",
    "requestId":"79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Configurar o formato de log da função

Para configurar o formato de log da função, você pode usar o console do Lambda ou a AWS Command Line Interface (AWS CLI). Você também pode configurar o formato de log de uma função usando os comandos CreateFunction e UpdateFunctionConfiguration da API do Lambda, o recurso AWS::Serverless::Function do AWS Serverless Application Model (AWS SAM) e o recurso AWS::Lambda::Function do AWS CloudFormation.

Alterar o formato de log da função não afeta os logs existentes armazenados no CloudWatch Logs. Somente os novos logs usarão o formato atualizado.

Se você alterar o formato de log da função para JSON e não definir o nível do log, o Lambda definirá automaticamente o nível do log da aplicação e do sistema da função como INFO. Isso significa que o Lambda só envia saídas de log de nível WARN e inferiores para o CloudWatch Logs. Para saber mais sobre a filtragem em nível de log da aplicação e do sistema, consulte the section called "Filtragem em nível de log"

> **Note**
>
> Em runtimes do Python, quando o formato de log da função é definido como texto sem formatação, a configuração padrão de nível de log é WARN. Isso significa que o Lambda só envia saídas de log de nível WARN e inferior para o CloudWatch Logs. Alterar o formato de log da função para JSON altera esse comportamento padrão. Para saber mais sobre registro em log no Python, consulte the section called "Registro em log".

Para funções do Node.js que emitem logs de formato de métricas incorporadas (EMF), alterar o formato de log da função para JSON pode fazer com que o CloudWatch não consiga reconhecer as métricas.

> **Important**
>
> Se a função usa o Powertools para AWS Lambda (TypeScript) ou bibliotecas cliente EMF de código aberto para emitir logs em EMF, atualize as bibliotecas do Powertools e EMF para as versões mais recentes a fim de garantir que o CloudWatch possa continuar analisando os logs corretamente. Se você mudar para logs em formato JSON, também recomendamos que você realize testes para garantir a compatibilidade com as métricas incorporadas da sua função. Para obter mais informações sobre as funções do node.js que emitem logs em

EMF, consulte the section called “Usar bibliotecas cliente com logs de formato de métricas incorporadas (EMF) em JSON estruturado”.

Configurar o formato em log da função (console)

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Na página de configuração da função, escolha Ferramentas de monitoramento e operações.
4. No painel Configuração de registro em log, escolha Editar.
5. Em Conteúdo do log, em Formato do log, selecione Texto ou JSON.
6. Escolha Salvar.

Alterar o formato de log de uma função existente (AWS CLI)

- Para alterar o formato de log de uma função existente, use o comando `update-function-configuration`. Defina a opção `LogFormat` em `LoggingConfig` como `JSON` ou `Text`.

```
aws lambda update-function-configuration \
--function-name myFunction --logging-config LogFormat=JSON
```

Definir o formato do log ao criar uma função (AWS CLI)

- Para configurar o formato do log ao criar uma nova função, use a opção `--logging-config` no comando `create-function`. Defina `LogFormat` como `JSON` ou `Text`. O exemplo de comando a seguir cria uma função usando o runtime do Node.js 18, que gera logs em JSON estruturado.

  Se você não especificar um formato de log ao criar uma função, o Lambda usará o formato de log padrão para a versão de runtime selecionada. Para obter informações sobre o formato do log, consulte the section called “Formatos de log padrão”.

```
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
```

Filtragem em nível de log

O Lambda pode filtrar os logs da função para que somente logs de um determinado nível de detalhes, ou de um nível inferior, sejam enviados para o CloudWatch Logs. Você pode configurar a filtragem em nível de logs separadamente para os logs do sistema da função (logs gerados pelo Lambda) e os logs de aplicações (logs gerados pelo código da função).

Para the section called “Runtimes e métodos de registro em log compatíveis”, você não precisa fazer nenhuma alteração no código da função do Lambda para filtrar logs de aplicações da função.

Para todos os outros runtimes e métodos de registro em log, o código da função deve gerar eventos de log para `stdout` ou `stderr` como objetos formatados em JSON que contenham um par de valores-chave com a chave `"level"`. Por exemplo, o Lambda interpreta a saída a seguir `stdout` como um log de nível DEBUG.

```
print('{"level": "debug", "msg": "my debug log", "timestamp":
 "2023-11-02T16:51:31.587199Z"}')
```

Se o campo de valor `"level"` for inválido ou estiver ausente, o Lambda atribuirá à saída do log o nível INFO. Para que o Lambda use o campo de carimbo de data/hora, você precisa especificar a hora em um formato RFC 3339 válido de carimbo de data/hora. Se você não fornecer um carimbo de data/hora válido, o Lambda atribuirá ao log o nível INFO e adicionará um carimbo de data/hora.

Ao denominar a chave de carimbo de data/hora, siga as convenções do runtime que estiver usando. O Lambda é compatível com as convenções de nomenclatura mais comuns usadas pelos runtimes gerenciados. Por exemplo, em funções que usam o runtime do.NET, o Lambda reconhece a chave `"Timestamp"`.

> **Note**
>
> Para usar a filtragem em nível de log, sua função precisar estar configurada para usar logs em formato JSON. O formato de log padrão para todos os runtimes gerenciados pelo Lambda atualmente é texto simples. Para saber como configurar logs em formato JSON para a função, consulte the section called “Configurar o formato de log da função”.

Para logs de aplicações (logs gerados pelo código da função), você pode escolher entre os níveis de log a seguir.

Nível de log	Uso padrão
TRACE (mais detalhes)	As informações mais detalhadas usadas para rastrear o caminho da execução do código
DEBUG	Informações detalhadas para depuração do sistema
INFO	Mensagens que registram a operação normal da função
WARN	Mensagens sobre possíveis erros que podem levar a um comportamento inesperado se não forem corrigidos
ERRO	Mensagens sobre problemas que impedem que o código funcione conforme o esperado
FATAL (menos detalhes)	Mensagens sobre erros graves que fazem a aplicação parar de funcionar

Quando você seleciona um nível de log, o Lambda envia logs desse nível, e de níveis inferiores, para o CloudWatch Logs. Por exemplo, se você definir o nível de log de aplicações de uma função como WARN, o Lambda não enviará saídas de log nos níveis INFO e DEBUG. O nível padrão de logs de aplicações para a filtragem de logs é INFO.

Quando o Lambda filtra os logs de aplicações da função, as mensagens de log sem nível receberão o nível INFO.

Para logs do sistema (logs gerados pelo serviço Lambda), você pode escolher entre os níveis de log a seguir.

Nível de log	Uso
DEBUG (mais detalhes)	Informações detalhadas para depuração do sistema

Nível de log	Uso
INFO	Mensagens que registram a operação normal da função
WARN (menos detalhes)	Mensagens sobre possíveis erros que podem levar a um comportamento inesperado se não forem corrigidos

Quando você seleciona um nível de log, o Lambda envia logs desse nível, e de níveis inferiores. Por exemplo, se você definir o nível de log do sistema de uma função como INFO, o Lambda não enviará saídas de log no nível DEBUG.

Por padrão, o Lambda define o nível de log do sistema como INFO. Com essa configuração, o Lambda envia automaticamente as mensagens de log `"start"` e `"report"` para o CloudWatch. Para receber logs do sistema mais ou menos detalhados, altere o nível de log para DEBUG ou WARN. Para ver uma lista dos níveis de log para os quais o Lambda mapeia diferentes eventos de log do sistema, consulte the section called “Mapeamento de eventos no nível de log do sistema”.

Configurar a filtragem em nível de log

Para configurar a filtragem em nível de log de aplicações e do sistema para a função, você pode usar o console do Lambda ou a AWS Command Line Interface (AWS CLI). Você também pode configurar o nível de log de uma função usando os comandos CreateFunction e UpdateFunctionConfiguration da API do Lambda, o recurso AWS::Serverless::Function do AWS Serverless Application Model (AWS SAM) e o recurso AWS::Lambda::Function do AWS CloudFormation.

Observe que, se você definir o nível de log da função no código, essa definição terá precedência sobre qualquer outra configuração de nível de log que você definir. Por exemplo, se você usar o método `logging setLevel()` do Python para definir o nível de registro em log da sua função como INFO, essa definição terá precedência sobre uma configuração de WARN que você definir usando o console do Lambda.

Configurar o nível de log de aplicações ou do sistema de uma função existente (console)

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Na página de configuração da função, escolha Ferramentas de monitoramento e operações.

4. No painel Configuração de registro em log, escolha Editar.
5. Em Conteúdo do log, em Formato do log, certifique-se de que a opção JSON esteja selecionada.
6. Use os botões de opção para selecionar o Nível de log da aplicação e o Nível de log do sistema desejados para a função.
7. Escolha Salvar.

Configurar o nível de logs de aplicações ou do sistema de uma função existente (AWS CLI)

- Para alterar o nível de logs de aplicações ou do sistema de uma função existente, use o comando `update-function-configuration`. Defina `--system-log-level` como `DEBUG`, `INFO` ou `WARN`. Defina `--application-log-level` como `DEBUG`, `INFO`, `WARN`, `ERROR` ou `FATAL`.

```
aws lambda update-function-configuration \
--function-name myFunction --system-log-level WARN \
--application-log-level ERROR
```

Configurar a filtragem em nível de log ao criar uma função

- Para configurar a filtragem em nível de log ao criar uma nova função, use as opções `--system-log-level` e `--application-log-level` no comando create-function. Defina `--system-log-level` como `DEBUG`, `INFO` ou `WARN`. Defina `--application-log-level` como `DEBUG`, `INFO`, `WARN`, `WARN` ou `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
```

Mapeamento de eventos no nível de log do sistema

Para eventos de log no nível do sistema gerados pelo Lambda, a tabela a seguir define o nível de log atribuído a cada evento. Para saber mais sobre os eventos listados na tabela, consulte the section called “Referência de esquema para `Event`”.

Nome do evento	Condição	Nível de log atribuído
initStart	runtimeVersion está definida	INFO
initStart	runtimeVersion não está definida	DEBUG
initRuntimeDone	status=success	DEBUG
initRuntimeDone	status!=success	WARN
initReport	initializationType=snapstart	INFO
initReport	initializationType!=snapstart	DEBUG
initReport	status!=success	WARN
restoreStart	runtimeVersion está definida	INFO
restoreStart	runtimeVersion não está definida	DEBUG
restoreRuntimeDone	status=success	DEBUG
restoreRuntimeDone	status!=success	WARN
restoreReport	status=success	INFO
restoreReport	status!=success	WARN
rápido	-	INFO
runtimeDone	status=success	DEBUG
runtimeDone	status!=success	WARN
relatório	status=success	INFO
relatório	status!=success	WARN
extensão	state=success	INFO

Nome do evento	Condição	Nível de log atribuído
extensão	state!=success	WARN
logSubscription	-	INFO
telemetrySubscription	-	INFO
logsDropped	-	WARN

> **Note**
>
> the section called “API de telemetria” sempre emite o conjunto completo de eventos da plataforma. Configurar o nível dos logs do sistema enviados pelo Lambda para o CloudWatch não afeta o comportamento da API de Telemetria do Lambda.

Filtragem em nível de log de aplicações com runtimes personalizados

Quando você configura a filtragem em nível de log de aplicações para a função, o Lambda define o nível de log de aplicações em segundo plano no runtime usando a variável de ambiente `AWS_LAMBDA_LOG_LEVEL`. O Lambda também define o formato de log da função usando a variável de ambiente `AWS_LAMBDA_LOG_FORMAT`. Você pode usar essas variáveis para integrar os controles avançados de registro em log do Lambda em um runtime personalizado.

Para definir as configurações de registro em log de uma função usando um runtime personalizado com o console, a AWS CLI e as APIs do Lambda, configure o runtime personalizado para verificar o valor dessas variáveis de ambiente. Depois disso, você pode configurar os loggers do runtime de acordo com o formato e os níveis de log selecionados.

Configurar grupos de logs do CloudWatch

Por padrão, o CloudWatch cria automaticamente um grupo de logs denominado `/aws/lambda/<function name>` para a função quando ela é invocada pela primeira vez. Para configurar a função para enviar logs a um grupo de logs existente ou para criar um novo grupo de logs para a função, você pode usar a AWS CLI ou o console do Lambda. Você também pode configurar grupos de logs personalizados usando os comandos CreateFunction e

UpdateFunctionConfiguration da API do Lambda e o recurso do AWS Serverless Application Model (AWS SAM) AWS::Serverless::Function.

Você pode configurar várias funções do Lambda para enviar logs ao mesmo grupo de logs do CloudWatch. Por exemplo, é possível usar um único grupo de logs para armazenar os logs de todas as funções do Lambda que fizerem parte de uma aplicação específica. Quando você usa um grupo de logs personalizado para uma função do Lambda, os fluxos de log criados pelo Lambda incluem o nome e a versão da função. Isso garante que o mapeamento entre mensagens de log e funções seja preservado, mesmo se você usar o mesmo grupo de logs para várias funções.

O formato de nomenclatura de fluxos de logs para grupos de logs personalizados segue esta convenção:

```
YYYY/MM/DD/<function_name>[<function_version>][<execution_environment_GUID>]
```

Observe que, ao configurar um grupo de logs personalizado, o nome selecionado para o grupo de logs deve seguir as regras de nomenclatura do CloudWatch Logs. Além disso, nomes de grupos de logs personalizados não devem começar com a string `aws/`. Se você criar um grupo de logs personalizado começando com `aws/`, o Lambda não conseguirá criar o grupo de logs. Como resultado, os logs da função não serão enviados para o CloudWatch.

Alterar o grupo de logs de uma função (console)

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Na página de configuração da função, escolha Ferramentas de monitoramento e operações.
4. No painel Configuração de registro em log, escolha Editar.
5. No painel Grupo de logs do grupo de logs do CloudWatch, escolha Personalizado.
6. Em Grupo de logs personalizado, insira o nome do grupo de logs do CloudWatch para o qual você deseja que sua função envie logs. Se você inserir o nome de um grupo de logs existente, sua função usará esse grupo. Se não existir nenhum grupo de logs com o nome inserido, o Lambda criará um novo grupo de logs para a função com esse nome.

Alterar o grupo de logs de uma função (AWS CLI)

- Para alterar o grupo de logs de uma função existente, use o comando `update-function-configuration`. Se você especificar o nome de um grupo de logs existente, sua função usará

esse grupo. Se não existir nenhum grupo de logs com o nome especificado, o Lambda criará um novo grupo de logs para a função com esse nome.

```
aws lambda update-function-configuration \
--function-name myFunction --log-group myLogGroup
```

Especificar um grupo de logs personalizado ao criar uma função (AWS CLI)

- Para especificar um grupo de logs personalizado ao criar uma nova função do Lambda usando a AWS CLI, use a opção `--log-group`. Se você especificar o nome de um grupo de logs existente, sua função usará esse grupo. Se não existir nenhum grupo de logs com o nome especificado, o Lambda criará um novo grupo de logs para a função com esse nome.

  O comando de exemplo a seguir cria uma função do Lambda para Node.js que envia logs para um grupo de logs denominado `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
  ```

Permissões da função de execução

Para que sua função envie logs para o CloudWatch Logs, ela precisa ter a permissão `logs:PutLogEvents`. Quando você configura o grupo de logs da função usando o console do Lambda, se a função não tiver essa permissão, o Lambda a adiciona por padrão ao perfil de execução da função. Quando o Lambda adiciona essa permissão, ele dá à função permissão para enviar logs para qualquer grupo de logs do CloudWatch Logs.

Para evitar que o Lambda atualize automaticamente o perfil de execução da função e o edite manualmente, expanda Permissões e desmarque Adicionar permissões necessárias.

Quando você configura o grupo de logs da função usando a AWS CLI, o Lambda não adiciona automaticamente a permissão `logs:PutLogEvents`. Adicione a permissão ao perfil de execução da função, caso isso ainda não tenha sido feito. Essa permissão está incluída na política gerenciada AWSLambdaBasicExecutionRole.

Acesso a logs com o console do Lambda

Para visualizar os logs usando o console do Lambda

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Monitor.
4. Escolha View logs in CloudWatch (Exibir logs no CloudWatch).

Acesso a logs com a AWS CLI

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- AWS Command Line Interface (AWS CLI) versão 2
- AWS CLI – Configuração rápida com `aws configure`

Você pode usar a AWS CLI para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

A seguinte saída deverá ser mostrada:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `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
```

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 2.

A seguinte saída deverá ser mostrada:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e Ubuntu no Windows. Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa `sed` para remover as aspas do arquivo de saída e fica inativo por 15 segundos para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

A opção cli-binary-format será necessária se você estiver usando a AWS CLI versão 2. Para que essa seja a configuração padrão, execute `aws configure set cli-binary-format raw-in-base64-out`. Para obter mais informações, consulte A AWS CLI comporta opções de linha de comando globais no Guia do usuário da AWS Command Line Interface versão 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 macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod -R 755 get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

A seguinte saída deverá ser mostrada:

```
{
    "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"
}
```


Registro em log da função de runtime

Para depurar e validar que seu código está funcionando conforme o esperado, é possível gerar logs com a funcionalidade de registro em log padrão para sua linguagem de programação. O runtime do Lambda faz upload da saída de log da sua função para o CloudWatch Logs. Para obter instruções específicas de linguagem, consulte os tópicos a seguir:

- Registro em log da função do AWS Lambda em Node.js
- Registro em log da função do AWS Lambda em Python
- Registro em log da função do AWS Lambda em Ruby
- Registro em log da função do AWS Lambda em Java
- Registro em log da função do AWS Lambda em Go
- Registro em log da função do Lambda em C#
- Registro em log da função do AWS Lambda no PowerShell

Próximas etapas

- Saiba mais sobre os grupos de logs e como acessá-los no console do CloudWatch em Monitoring system, application, and custom log files no Manual do usuário do Amazon CloudWatch.

Registrar em log chamadas de API do AWS Lambda usando o AWS CloudTrail

O AWS Lambda é integrado ao AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, um perfil ou um AWS service (Serviço da AWS). O CloudTrail captura chamadas de API do Lambda como eventos. As chamadas capturadas incluem as chamadas de código do console do Lambda e as chamadas para as operações da API do Lambda. Ao fazer uso das informações coletadas pelo CloudTrail, é possível determinar a solicitação feita ao Lambda, o endereço IP no qual a solicitação foi feita, quando a solicitação foi feita e detalhes adicionais.

Cada entrada de log ou evento contém informações sobre quem gerou a solicitação. As informações de identidade ajudam a determinar:

- Se a solicitação foi feita com credenciais de usuário raiz ou credenciais de usuário.
- Se a solicitação foi feita em nome de um usuário do Centro de Identidade do IAM.
- Se a solicitação foi feita com credenciais de segurança temporárias de uma função ou de um usuário federado.
- Se a solicitação foi feita por outro AWS service (Serviço da AWS).

O CloudTrail está ativo em sua Conta da AWS e você tem acesso automático ao Histórico de eventos do CloudTrail. O Histórico de eventos do CloudTrail fornece um registro visualizável, pesquisável, baixável e imutável dos últimos 90 dias de eventos de gerenciamento gravados em uma Região da AWS. Para obter mais informações, consulte Trabalhar com histórico de eventos do CloudTrail no Guia do usuário do AWS CloudTrail. Não há cobranças do CloudTrail pela visualização do Histórico de eventos.

Para obter um registro contínuo de eventos em sua Conta da AWS nos últimos 90 dias, crie uma trilha ou um um armazenamento de dados de eventos do CloudTrail Lake.

Trilhas do CloudTrail

> Uma trilha permite que o CloudTrail entregue arquivos de log a um bucket Amazon S3. As trilhas criadas usando o AWS Management Console são de várias regiões. Só é possível criar uma trilha de região única ou de várias regiões usando a AWS CLI. Criar uma trilha de várias regiões é uma prática recomendada, pois você captura atividades em todas as Regiões da AWS da conta. Se você criar uma trilha de região única, poderá visualizar somente os eventos registrados na Região

da AWS da trilha. Para obter mais informações sobre trilhas, consulte Criar uma trilha para a Conta da AWS e Criar uma trilha para uma organização no Guia do usuário do AWS CloudTrail.

Uma cópia dos seus eventos de gerenciamento em andamento pode ser entregue no bucket do Amazon S3 sem nenhum custo via CloudTrail com a criação de uma trilha. No entanto, há cobranças de armazenamento do Amazon S3. Para obter mais informações sobre os preços do CloudTrail, consulte Preços do AWS CloudTrail. Para receber informações sobre a definição de preço do Amazon S3, consulte Definição de preço do Amazon S3.

Armazenamentos de dados de eventos do CloudTrail Lake

O CloudTrail Lake permite executar consultas baseadas em SQL em seus eventos. O CloudTrail Lake converte eventos existentes em formato JSON baseado em linhas para o formato Apache ORC. O ORC é um formato colunar de armazenamento otimizado para recuperação rápida de dados. Os eventos são agregados em armazenamentos de dados de eventos, que são coleções imutáveis de eventos baseados nos critérios selecionados com a aplicação de seletores de eventos avançados. Os seletores que você aplica a um armazenamento de dados de eventos controlam quais eventos persistem e estão disponíveis para você consultar. Para obter mais informações sobre o CloudTrail Lake, consulte Trabalhar com o AWS CloudTrail Lake, no Guia do usuário do AWS CloudTrail.

Os armazenamentos de dados de eventos e consultas do CloudTrail Lake incorrem em custos. Ao criar um armazenamento de dados de eventos, você escolhe a opção de preço que deseja usar para ele. A opção de preço determina o custo para a ingestão e para o armazenamento de eventos, e o período de retenção padrão e máximo para o armazenamento de dados de eventos. Para obter mais informações sobre os preços do CloudTrail, consulte Preços do AWS CloudTrail.

Eventos de dados do Lambda no CloudTrail

Os eventos de dados fornecem informações sobre as operações de recursos realizadas em um recurso (por exemplo, leitura ou gravação em um objeto do Amazon S3). Elas também são conhecidas como operações de plano de dados. Eventos de dados geralmente são atividades de alto volume. Por padrão, o CloudTrail não registra em log a maioria dos eventos de dados, e o Histórico de eventos do CloudTrail não os registra.

Um evento de dados do CloudTrail que é registrado em log por padrão para serviços compatíveis é `LambdaESMDisabled`. Para saber mais sobre o uso desse evento para ajudar a solucionar problemas de mapeamentos das origens dos eventos do Lambda, consulte the section called “Usar o CloudTrail para solucionar problemas de origens de eventos do Lambda desabilitadas”.

Há cobranças adicionais para eventos de dados. Para obter mais informações sobre os preços do CloudTrail, consulte Preços do AWS CloudTrail.

É possível registrar em log eventos de dados para o tipo de recurso `AWS::Lambda::Function` usando o console do CloudTrail, a AWS CLI ou operações da API do CloudTrail. Para obter mais informações sobre como registrar eventos de dados em log, consulte Registrar eventos de dados com o AWS Management Console e Registrar eventos de dados com a AWS Command Line Interface no Guia do usuário do AWS CloudTrail.

A tabela a seguir lista o tipo de recurso do Lambda para o qual você pode registrar eventos de dados em log. A coluna Tipo de evento de dados (console) mostra o valor a ser escolhido na lista Tipo de evento de dados no console do CloudTrail. A coluna do valor resources.type mostra o valor de `resources.type` que você especificaria ao configurar seletores de eventos avançados usando a AWS CLI ou as APIs do CloudTrail. A coluna APIs de dados registradas no CloudTrail mostra as chamadas de API registradas no CloudTrail para o tipo de recurso.

Tipo de evento de dados (console)	valor resources.type	APIs de dados registradas no CloudTrail
Lambda	`AWS::Lambda::Function`	Invoke

É possível configurar seletores de eventos avançados para filtrar os campos `eventName`, `readOnly` e `resources.ARN` para registrar em log somente os eventos que são importantes para você. O exemplo a seguir é a visualização JSON de uma configuração de evento de dados que registra eventos somente para uma função específica. Para obter mais informações sobre esses campos, consulte AdvancedFieldSelector na Referência de API do AWS CloudTrail.

```
[
  {
    "name": "function-invokes",
    "fieldSelectors": [
      {
        "field": "eventCategory",
        "equals": [
          "Data"
        ]
      },
```

```
      {
        "field": "resources.type",
        "equals": [
          "AWS::Lambda::Function"
        ]
      },
      {
        "field": "resources.ARN",
        "equals": [
          "arn:aws:lambda:us-east-1:111122223333:function:hello-world"
        ]
      }
    ]
  }
]
```

Eventos de gerenciamento do Lambda no CloudTrail

Os Eventos de gerenciamento fornecem informações sobre operações de gerenciamento executadas em recursos na sua Conta da AWS. Elas também são conhecidas como operações de plano de controle. Por padrão, o CloudTrail registra eventos de gerenciamento em logs.

O Lambda oferece suporte ao registro das seguintes ações como eventos de gerenciamento nos arquivos de log do CloudTrail.

> **Note**
>
> No arquivo de log do CloudTrail, o `eventName` pode incluir informações como data e versão, mas ainda está se referindo à mesma ação de API. Por exemplo, a ação `GetFunction` aparece como `GetFunction20150331v2`. A lista a seguir especifica quando o nome do evento é diferente do nome da ação da API.

- AddLayerVersionPermission
- AddPermission (nome do evento: `AddPermission20150331v2`)
- CreateAlias (nome do evento: `CreateAlias20150331`)
- CreateEventSourceMapping (nome do evento: `CreateEventSourceMapping20150331`)
- CreateFunction (nome do evento: `CreateFunction20150331`)

(Os parâmetros `Environment` e `ZipFile` são omitidos dos logs do CloudTrail para `CreateFunction`.)

- CreateFunctionUrlConfig
- DeleteAlias (nome do evento: `DeleteAlias20150331`)
- DeleteCodeSigningConfig
- DeleteEventSourceMapping (nome do evento: `DeleteEventSourceMapping20150331`)
- DeleteFunction (nome do evento: `DeleteFunction20150331`)
- DeleteFunctionConcurrency (nome do evento: `DeleteFunctionConcurrency20171031`)
- DeleteFunctionUrlConfig
- DeleteProvisionedConcurrencyConfig
- GetAlias (nome do evento: `GetAlias20150331`)
- GetEventSourceMapping
- GetFunction
- GetFunctionUrlConfig
- GetFunctionConfiguration
- GetLayerVersionPolicy
- GetPolicy
- ListEventSourceMappings
- ListFunctions
- ListFunctionUrlConfigs
- PublishLayerVersion (nome do evento: `PublishLayerVersion20181031`)

  (O parâmetro `ZipFile` é omitido dos logs do CloudTrail para `PublishLayerVersion`.)
- PublishVersion (nome do evento: `PublishVersion20150331`)
- PutFunctionConcurrency (nome do evento: `PutFunctionConcurrency20171031`)
- PutFunctionCodeSigningConfig
- PutFunctionEventInvokeConfig
- PutProvisionedConcurrencyConfig
- PutRuntimeManagementConfig
- RemovePermission (nome do evento: `RemovePermission20150331v2`)
- TagResource (nome do evento: `TagResource20170331v2`)

- UntagResource (nome do evento: `UntagResource20170331v2`)
- UpdateAlias (nome do evento: `UpdateAlias20150331`)
- UpdateCodeSigningConfig
- UpdateEventSourceMapping (nome do evento: `UpdateEventSourceMapping20150331`)
- UpdateFunctionCode (nome do evento: `UpdateFunctionCode20150331v2`)

  (O parâmetro `ZipFile` é omitido dos logs do CloudTrail para `UpdateFunctionCode`.)
- UpdateFunctionConfiguration (nome do evento: `UpdateFunctionConfiguration20150331v2`)

  (O parâmetro `Environment` é omitido dos logs do CloudTrail para `UpdateFunctionConfiguration`.)
- UpdateFunctionEventInvokeConfig
- UpdateFunctionUrlConfig

Usar o CloudTrail para solucionar problemas de origens de eventos do Lambda desabilitadas

Quando você altera o estado de um mapeamento da origem do evento usando a ação de API UpdateEventSourceMapping, a chamada de API é registrada em log como um evento de gerenciamento do CloudTrail. Os mapeamentos de origens de eventos também podem passar diretamente para o estado `Disabled` devido a erros.

Para os seguintes serviços, o Lambda publica o evento de dados `LambdaESMDisabled` no CloudTrail quando a origem dos eventos passa para o estado Desativado:

- Amazon Simple Queue Service (Amazon SQS)
- Amazon DynamoDB
- Amazon Kinesis

O Lambda não é compatível com esse evento para nenhum outro tipo de mapeamento da origem do evento.

Para receber alertas quando os mapeamentos das origens dos eventos dos serviços compatíveis passarem para o estado `Disabled`, configure um alarme no Amazon CloudWatch usando o evento `LambdaESMDisabled` do CloudTrail. Para obter mais informações sobre a configuração de um alarme do CloudWatch, consulte Creating CloudWatch alarms for CloudTrail events: examples.

A entidade `serviceEventDetails` na mensagem do evento `LambdaESMDisabled` contém um dos códigos de erro a seguir.

`RESOURCE_NOT_FOUND`

O recurso especificado na solicitação não existe.

`FUNCTION_NOT_FOUND`

A função anexada à fonte de evento não existe.

`REGION_NAME_NOT_VALID`

O nome de uma região fornecido para a função ou para a fonte de evento é inválido.

`AUTHORIZATION_ERROR`

As permissões não foram definidas ou estão configuradas incorretamente.

`FUNCTION_IN_FAILED_STATE`

O código da função não compila, encontrou uma exceção irrecuperável ou ocorreu uma implantação incorreta.

Exemplos de eventos do Lambda

Um evento representa uma única solicitação de qualquer origem e inclui informações sobre a operação solicitada, a data e a hora da operação da API, os parâmetros de solicitação etc. Os arquivos de log do CloudTrail não são um rastreamento de pilha ordenada de chamadas de API pública, portanto não são exibidos em uma ordem específica.

O exemplo a seguir mostra entradas de log do CloudTrail que demonstram as ações `GetFunction` e `DeleteFunction`.

> **Note**
>
> O `eventName` pode incluir informações como data e versão, como `"GetFunction20150331"`, mas ainda está se referindo à mesma API pública.

```
{
  "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"
    }
  ]
}
```

Para obter informações sobre o conteúdo dos registros do CloudTrail, consulte Conteúdo dos registros do CloudTrail no Guia do usuário do AWS CloudTrail.

Visualizar as invocações da função do Lambda usando o AWS X-Ray

Você pode usar o AWS X-Ray para visualizar os componentes do aplicativo, identificar gargalos de performance e solucionar problemas de solicitações que resultaram em um erro. Suas funções do Lambda enviam dados de rastreamento para o X-Ray, e o X-Ray processa os dados para gerar um mapa de serviço e resumos de rastreamento pesquisáveis.

Se você tiver habilitado o rastreamento do X-Ray em um serviço que invoca sua função, o Lambda enviará rastreamentos para o X-Ray automaticamente. O serviço upstream, como Amazon API Gateway, ou uma aplicação hospedada no Amazon EC2 que é instrumentada com o X-Ray SDK, analisa solicitações de entrada e adiciona um cabeçalho de rastreamento que informa ao Lambda para enviar rastreamentos ou não. Os rastreamentos de produtores de mensagens upstream, como o Amazon SQS, são vinculados automaticamente aos rastreamentos de funções do Lambda downstream, o que cria uma visão completa de toda a aplicação. Para obter mais informações, consulte Tracing event-driven applications (Rastreamento de aplicações orientadas a eventos) no Guia do desenvolvedor do AWS X-Ray.

> **Note**
>
> Atualmente, o rastreamento do X-Ray não oferece suporte a funções do Lambda com o Amazon Managed Streaming for Apache Kafka (Amazon MSK), Apache Kafka autogerenciado, Amazon MQ com ActiveMQ e RabbitMQ ou mapeamentos da origem do evento do Amazon DocumentDB.

Para alternar o rastreamento ativo na sua função do Lambda usando o console, siga as etapas abaixo:

Para ativar o rastreamento ativo

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e depois Monitoring and operations tools (Ferramentas de monitoramento e operações).
4. Selecione a opção Editar.
5. Em X-Ray, ative a opção Active tracing (Rastreamento ativo).

6. Escolha Salvar.

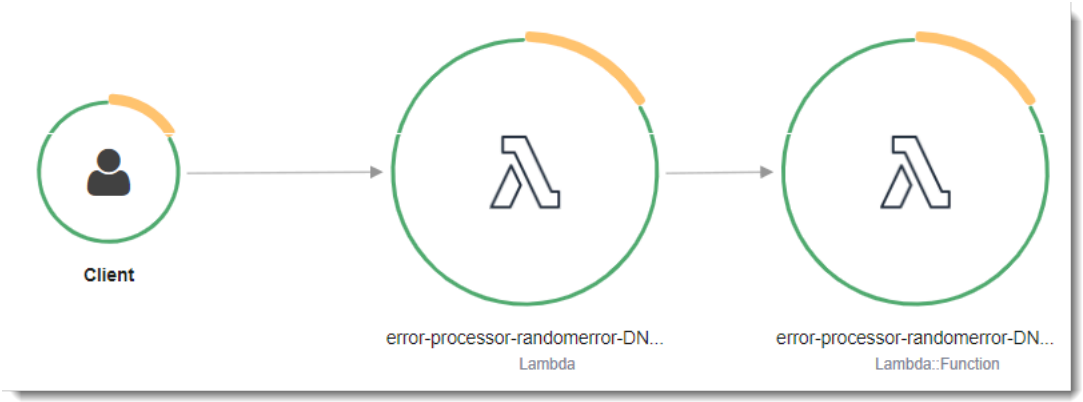
> **Definição de preço**
>
> Você pode usar o rastreamento do X-Ray gratuitamente todos os meses até determinado limite como parte do nível gratuito da AWS. Além do limite, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter mais informações, consulte Preços do AWS X-Ray.

Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você ativa o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à função de execução da função. Caso contrário, adicione a política AWSXRayDaemonWriteAccess à função de execução.

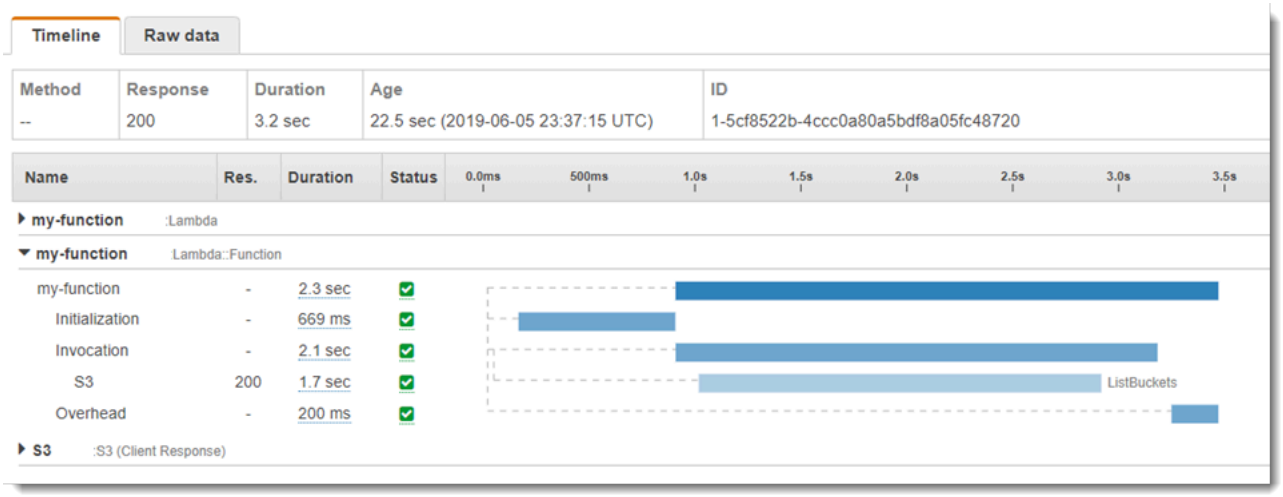
O X-Ray não rastreia todas as solicitações para sua aplicação. O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento seja eficiente, enquanto ainda fornece uma amostra representativa das solicitações. A taxa de amostragem é uma solicitação por segundo e 5% de solicitações adicionais.

> **Note**
>
> Não é possível configurar a taxa de amostragem do X-Ray para suas funções.

No X-Ray, um rastreamento registra informações sobre uma solicitação que é processada por um ou mais serviços. O Lambda registra dois segmentos por rastreamento, o que cria dois nós no gráfico de serviços. A imagem a seguir destaca esses dois nós:



O primeiro nó à esquerda representa o serviço do Lambda, que recebe a solicitação de invocação. O segundo nó representa a sua função do Lambda específica. O exemplo a seguir mostra um rastreamento com esses dois segmentos. Ambos têm o nome my-function, mas um tem a origem `AWS::Lambda` e o outro, a origem `AWS::Lambda::Function`. Se o segmento `AWS::Lambda` mostrar um erro, o serviço Lambda teve um problema. Se o segmento `AWS::Lambda::Function` mostrar um erro, sua função teve um problema.



O segmento de função (`AWS::Lambda::Function`) vem com subsegmentos para `Initialization`, `Invocation`, `Restore` (Lambda SnapStart somente) e `Overhead`. Para obter mais informações, consulte Ciclo de vida do ambiente de execução do Lambda.

> **Note**
>
> O X-Ray trata as exceções não processadas em sua função do Lambda, como o status `Error`. O X-Ray registra os status `Fault` somente quando o Lambda apresenta erros internos do servidor. Para obter mais informações, consulte Errors, faults, and exceptions (Erros, falhas e exceções) no Guia do desenvolvedor do X-Ray.

O subsegmento `Initialization` representa a fase inicial do ciclo de vida do ambiente de execução do Lambda. Durante essa fase, Lambda cria ou descongela um ambiente de execução com os recursos configurados, faz download do código da função e de todas as camadas, inicializa extensões, inicializa o runtime e executa o código de inicialização da função.

O subsegmento `Invocation` representa a fase de chamada em que o Lambda chama o manipulador de função. Isso começa com o registro do runtime e da extensão e termina quando o runtime está pronto para enviar a resposta.

(Somente Lambda SnapStart) O subsegmento `Restore` mostra o tempo necessário para o Lambda restaurar um snapshot, carregar o runtime (JVM) e executar qualquer hook de runtime de `afterRestore`. O processo de restauração de snapshots pode incluir o tempo gasto em atividades fora da MicroVM. Esse tempo é relatado no subsegmento `Restore`. Você não é cobrado pelo tempo gasto fora da microVM para restaurar um snapshot.

O subsegmento `Overhead` representa a fase que ocorre entre o momento em que o runtime envia a resposta e o sinal para a próxima chamada. Durante esse período, o runtime termina todas as tarefas relacionadas a uma chamada e se prepara para congelar o sandbox.

> **Note**
>
> Ocasionalmente, é possível notar um grande intervalo entre as fases de inicialização e de invocação da função em seus rastreamentos do X-Ray. Para as funções que usam a simultaneidade provisionada, isso ocorre pelo Lambda inicializar as instâncias de função com bastante antecedência à invocação. Para as funções que usam a simultaneidade não reservada (sob demanda), o Lambda pode inicializar proativamente uma instância de função, mesmo que não haja invocação. Visualmente, ambos os casos apresentam um intervalo de tempo entre as fases de inicialização e de invocação.

> **Important**
>
> No Lambda, é possível usar o X-Ray SDK para estender o subsegmento `Invocation` com subsegmentos adicionais para anotações, metadados e chamadas downstream. Não é possível acessar o segmento de função diretamente nem registrar trabalhos feitos fora do escopo de chamada do manipulador.

Consulte os tópicos a seguir para obter uma introdução específica de uma linguagem ao rastreamento no Lambda:

- Instrumentação do código Node.js no AWS Lambda
- Instrumentação do código Python no AWS Lambda
- Instrumentar o código Ruby no AWS Lambda
- Instrumentação do código Java no AWS Lambda
- Instrumentação do código Go no AWS Lambda

- Instrumentar o código C # no AWS Lambda

Para obter uma lista completa de serviços que oferecem suporte à instrumentação ativa, consulte Serviços da AWS compatíveis no Guia do desenvolvedor do AWS X-Ray.

Seções

- Permissões da função de execução
- O daemon do AWS X-Ray
- Habilitar o rastreamento ativo com a API do Lambda
- Habilitar o rastreamento ativo com o AWS CloudFormation

Permissões da função de execução

O Lambda precisa das permissões a seguir para enviar dados de rastreamento para o X-Ray. Adicione-as à função de execução da sua função.

- xray:PutTraceSegments
- xray:PutTelemetryRecords

Essas permissões estão incluídas na política gerenciada AWSXRayDaemonWriteAccess.

O daemon do AWS X-Ray

Em vez de enviar dados de rastreamento diretamente para a API do X-Ray, o X-Ray SDK usa um processo do daemon. O daemon do AWS X-Ray é uma aplicação que é executada no ambiente do Lambda e escuta o tráfego UDP que contém segmentos e subsegmentos. Ele armazena em buffer os dados recebidos e os grava no X-Ray em lotes, reduzindo a sobrecarga de processamento e memória necessária para rastrear invocações.

O runtime do Lambda permite ao daemon até 3% da memória configurada da sua função ou 16 MB, o que for maior. Se sua função ficar sem memória durante a invocação, o runtime encerrará o processo do daemon primeiro para liberar memória.

O processo do daemon é totalmente gerenciado pelo Lambda e não pode ser configurado pelo usuário. Todos os segmentos gerados por invocações de função são registrados na mesma conta que a função do Lambda. O daemon não pode ser configurado para redirecioná-los para nenhuma outra conta.

Para obter mais informações, consulte The X-Ray daemon no Guia do desenvolvedor do X-Ray.

Habilitar o rastreamento ativo com a API do Lambda

Para gerenciar a configuração de rastreamento com a AWS CLI ou com o AWS SDK, use as seguintes operações de API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration \
--function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Habilitar o rastreamento ativo com o AWS CloudFormation

Para ativar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example function-inline.yml: configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example template.yml: configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Monitorar a performance de funções com o Lambda Insights do Amazon CloudWatch

O Lambda Insights do Amazon CloudWatch coleta e agrega métricas e logs de performance do tempo de execução da função do Lambda para as aplicações sem servidor. Esta página descreve como ativar e usar o Lambda Insights para diagnosticar problemas em suas funções do Lambda.

Seções

Como o Lambda Insights monitora aplicações sem servidor

O Lambda Insights do CloudWatch Lambda é uma solução de monitoramento e solução de problemas para aplicações sem servidor em execução no AWS Lambda. A solução coleta, agrega e resume métricas no nível do sistema, incluindo tempo da CPU, memória, disco e uso da rede. Ele também coleta, agrega e resume informações de diagnóstico, como inicializações a frio e desligamentos do operador do Lambda para ajudar a isolar problemas com as funções do Lambda e resolvê-los rapidamente.

Lambda Insights usa um novo CloudWatch Lambda Insightsextensão, que é fornecido como umLambda. Quando você habilita essa extensão em uma função do Lambda para um runtime compatível, ela coleta métricas no nível do sistema e emite um único evento de log de performance para cada invocação dessa função do Lambda. O CloudWatch usa formatação métrica incorporada para extrair métricas dos eventos de log. Para obter mais informações, consulte Usar extensões do AWS Lambda.

A camada do Lambda Insights estende `CreateLogStream` e `PutLogEvents` para o grupo de logs `/aws/lambda-insights/`.

Definição de preço

Quando você habilita o Lambda Insights para sua função do Lambda, o Lambda Insights relata 8 métricas por função e cada invocação de função envia cerca de 1 KB de dados de log para o CloudWatch. Você paga apenas pelas métricas e logs relatados para sua função pelo Lambda Insights. Não há tarifas mínimas nem políticas de uso obrigatório do serviço. Você não paga pelo Lambda Insights se a função não for invocada. Para obter um exemplo de preço, consulte Preço do Amazon CloudWatch.

Tempos de execução compatíveis

Você pode usar o Lambda Insights com qualquer um dos tempos de execução que oferecem suporte para extensões do Lambda.

Ativar o Lambda Insights no console do Lambda

É possível habilitar o monitoramento aprimorado do Lambda Insights em funções do Lambda novas e existentes. Quando você ativa o Lambda Insights em uma função no console do Lambda para um runtime compatível, o Lambda adiciona a extensão do Lambda Insights à sua função como uma camada e verifica ou tenta associar a política `CloudWatchLambdaInsightsExecutionRolePolicy` à função de execução da função.

Para habilitar o Lambda Insights no console do Lambda

1. Abra a página Funções do console do Lambda.
2. Escolha a função.
3. Escolha a guia Configuração.
4. No menu à esquerda, escolha Ferramentas de monitoramento e operações.
5. No painel Ferramentas de monitoramento adicionais, selecione Editar.
6. Em Lambda Insights do CloudWatch, ative o Monitoramento avançado.
7. Escolha Salvar.

Ativação do Lambda Insights por programação

Também é possível habilitar o Lambda Insights usando a AWS Command Line Interface (AWS CLI), a CLI do AWS Serverless Application Model (SAM), o AWS CloudFormation ou o AWS Cloud Development Kit (AWS CDK). Quando você habilita o Lambda Insights programaticamente em uma função para um runtime compatível, o CloudWatch associa a política `CloudWatchLambdaInsightsExecutionRolePolicy` à função de execução da sua função.

Para obter mais informações, consulte Conceitos básicos do Lambda Insights no Manual do usuário do Amazon CloudWatch.

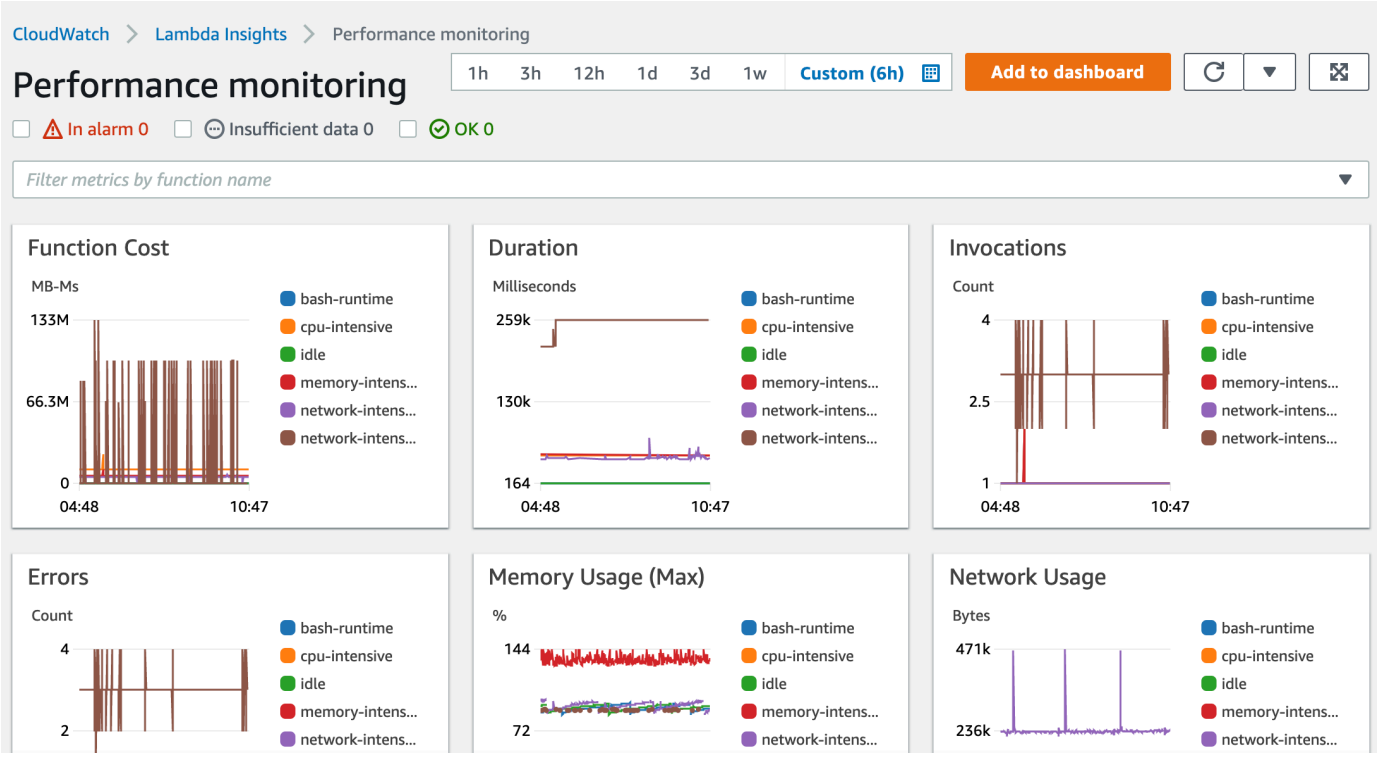
Usando o painel do Lambda Insights

O painel do Lambda Insights tem duas exibições no console do CloudWatch: a visão geral de várias funções e a exibição de função única. A visão geral de várias funções agrega as métricas de tempo de execução para as funções do Lambda na conta e na região atual da AWS. A visualização de função única mostra as métricas de tempo de execução disponíveis para uma única função do Lambda.

É possível usar a visão geral multifuncional do painel do Lambda Insights no console do CloudWatch para identificar funções do Lambda usadas em excesso e subutilizadas. É possível usar a visualização de função única do painel do Lambda Insights no console do CloudWatch para solucionar problemas de solicitações individuais.

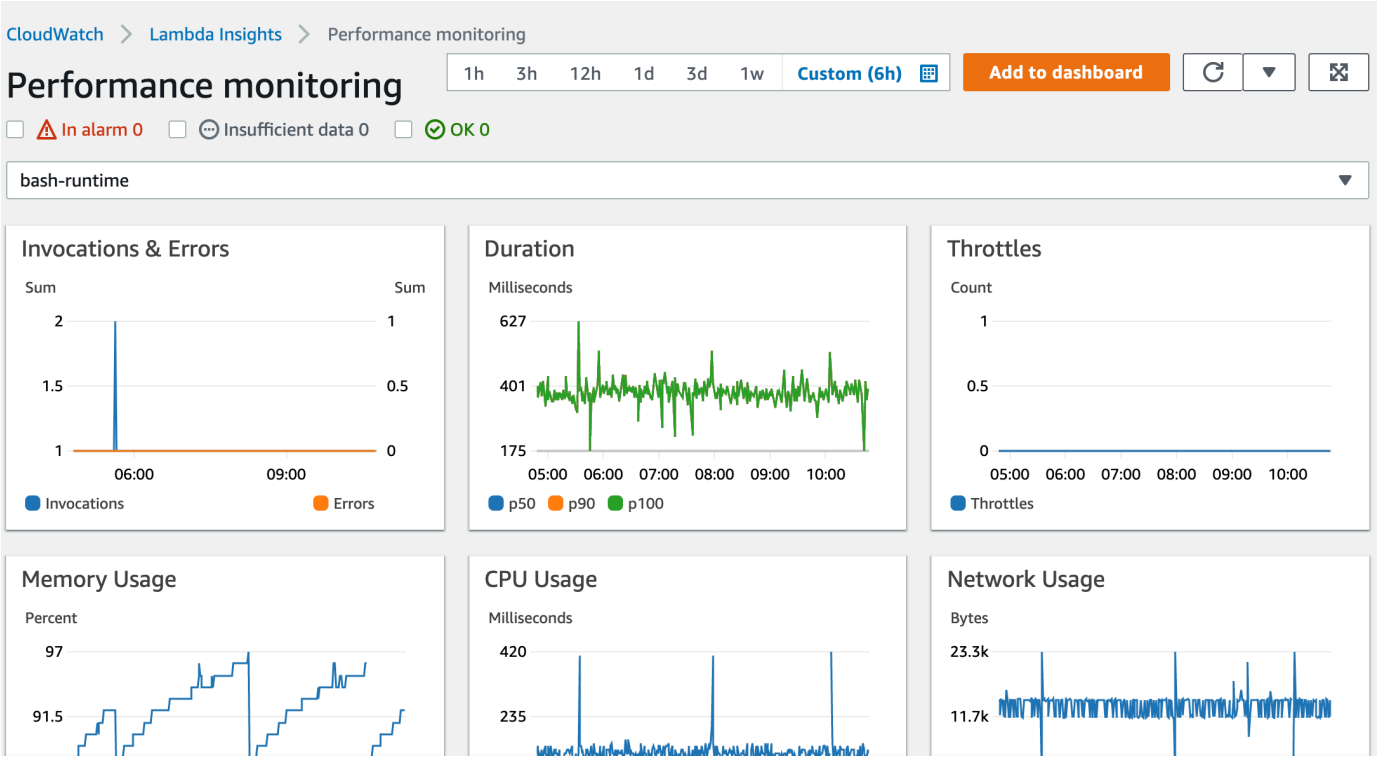
Como visualizar as métricas de tempo de execução de todas as funções

1. Abra a página Multi-funcion (Várias funções) no console do CloudWatch.
2. Escolha entre os intervalos de tempo predefinidos ou escolha um intervalo de tempo personalizado.
3. (Opcional) Selecione Add to dashboard (Adicionar ao painel) para adicionar os widgets ao painel do CloudWatch.



Como visualizar as métricas de tempo de execução de uma única função

1. Abra a página Single-funcion (Função única) no console do CloudWatch.
2. Escolha entre os intervalos de tempo predefinidos ou escolha um intervalo de tempo personalizado.
3. (Opcional) Selecione Add to dashboard (Adicionar ao painel) para adicionar os widgets ao painel do CloudWatch.



Para obter mais informações, consulte Criar e trabalhar com widgets em painéis do CloudWatch.

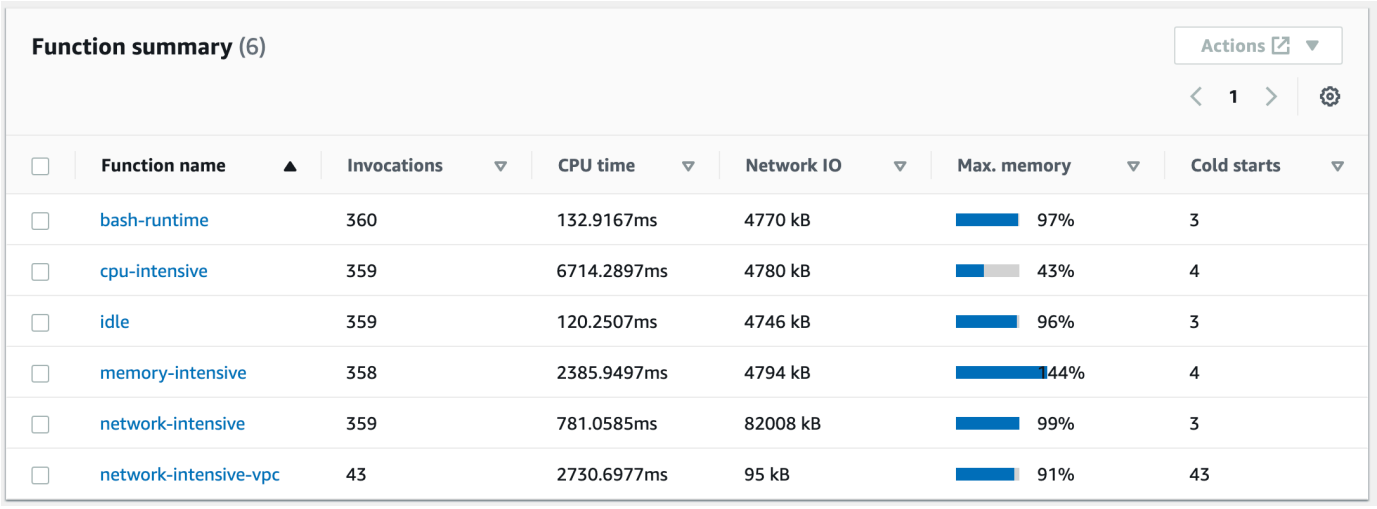
Exemplo de fluxo de trabalho para detectar anomalias de função

É possível usar a visão geral de várias funções no painel Lambda Insights para identificar e detectar anomalias de memória computacional com a função. Por exemplo, se a visão geral de várias funções indicar que uma função está usando uma grande quantidade de memória, você poderá visualizar métricas detalhadas de utilização da memória no painel Memory Usage (Uso de memória). Depois, você pode acessar o painel de métricas para habilitar a detecção de anomalias ou criar um alarme.

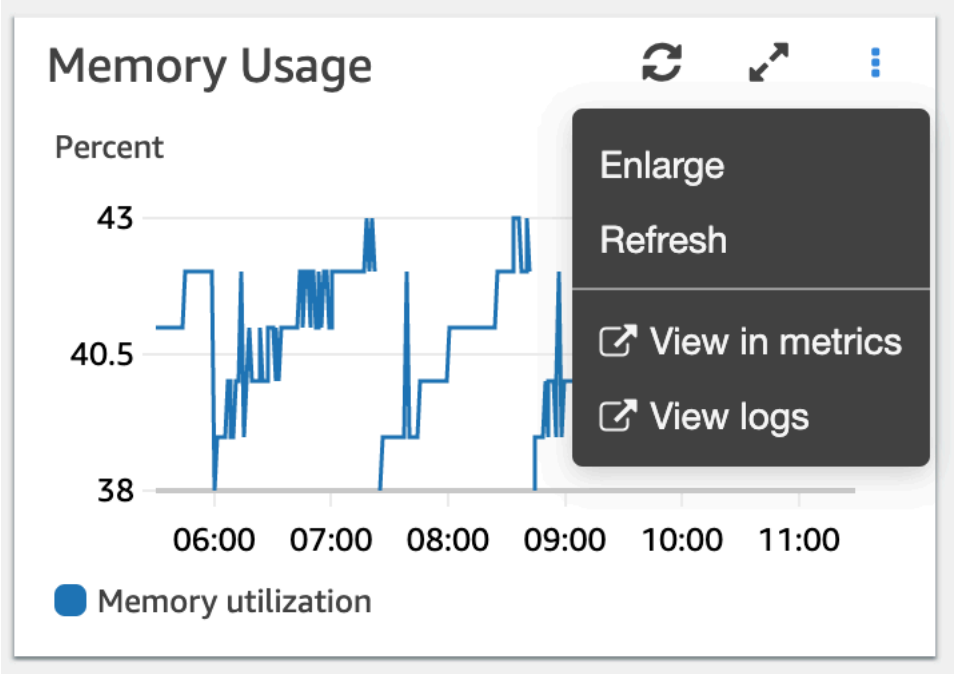
Como habilitar a detecção de anomalias para uma função

1. Abra a página Multi-funcion (Várias funções) no console do CloudWatch.
2. Em Function summary (Resumo da função), escolha o nome da função.

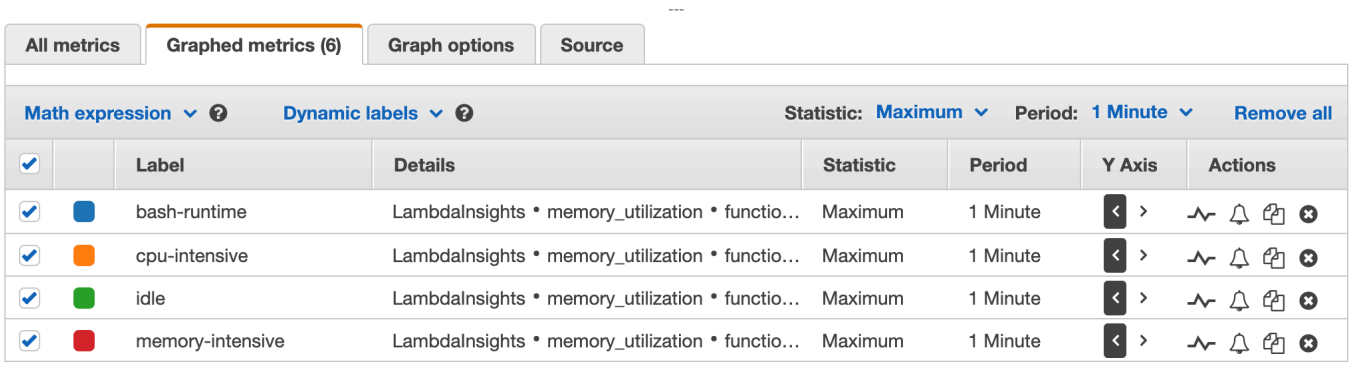
   A visualização de função única é aberta com as métricas de tempo de execução da função.



3. No painel Memory Usage (Uso de memória), selecione os três pontos na vertical e selecione View in metrics (Visualizar nas métricas) para abrir o painel Metrics (Métricas).



4. Na guia Graphed metrics (Métricas em gráficos), na coluna Actions (Ações), selecione o primeiro ícone para habilitar a detecção de anomalias da função.



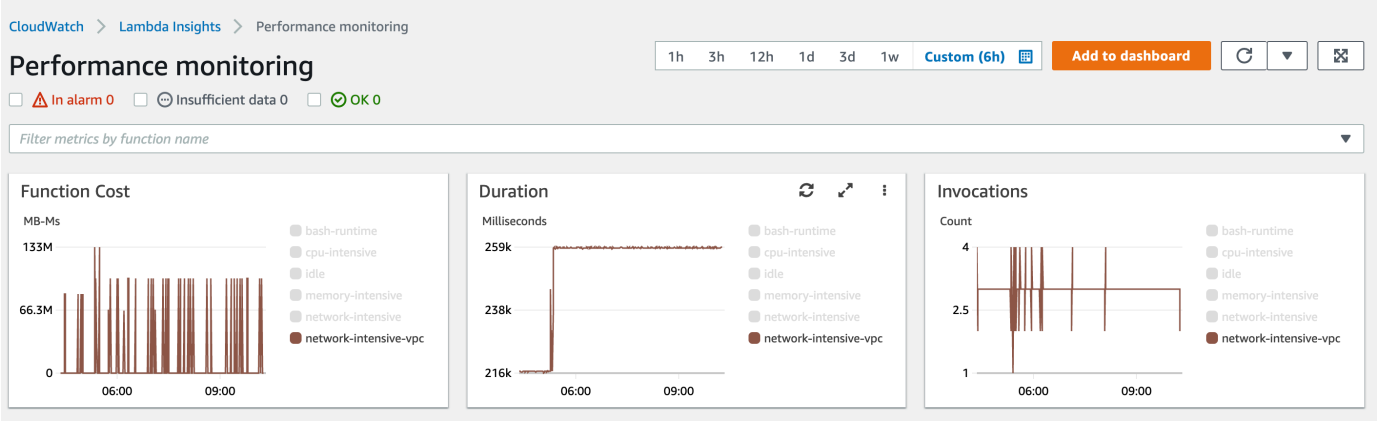
Para obter mais informações, consulte Usar a detecção de anomalias do CloudWatch.

Exemplo de fluxo de trabalho usando consultas para solucionar problemas de uma função

É possível usar a visualização de função única no painel Lambda Insights para identificar a causa raiz de um pico na duração da função. Por exemplo, se a visão geral de várias funções indicar um grande aumento na duração da função, você poderá pausar ou escolher cada função no painel Duration (Duração) para determinar qual função está causando o aumento. Depois, você pode acessar a visualização de função única e revisar os Application logs (Logs de aplicações) para determinar a causa raiz.

Como executar consultas em uma função

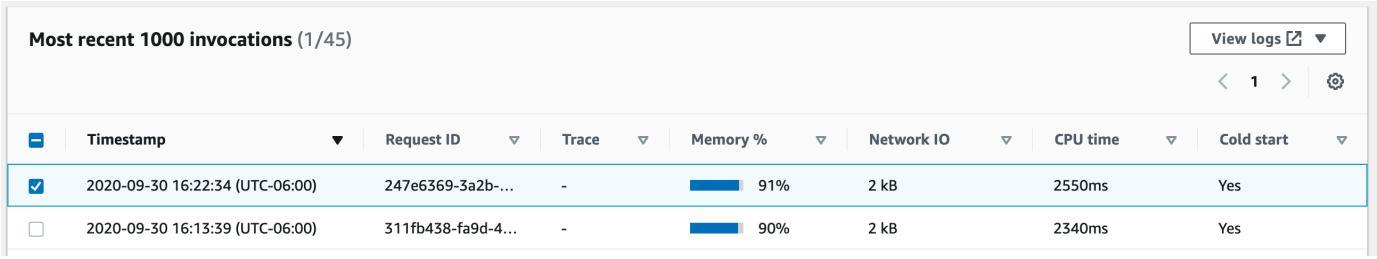
1. Abra a página Multi-funcion (Várias funções) no console do CloudWatch.
2. No painel Duration (Duração), selecione a função para filtrar as métricas de duração.



3. Abra a página Função única.

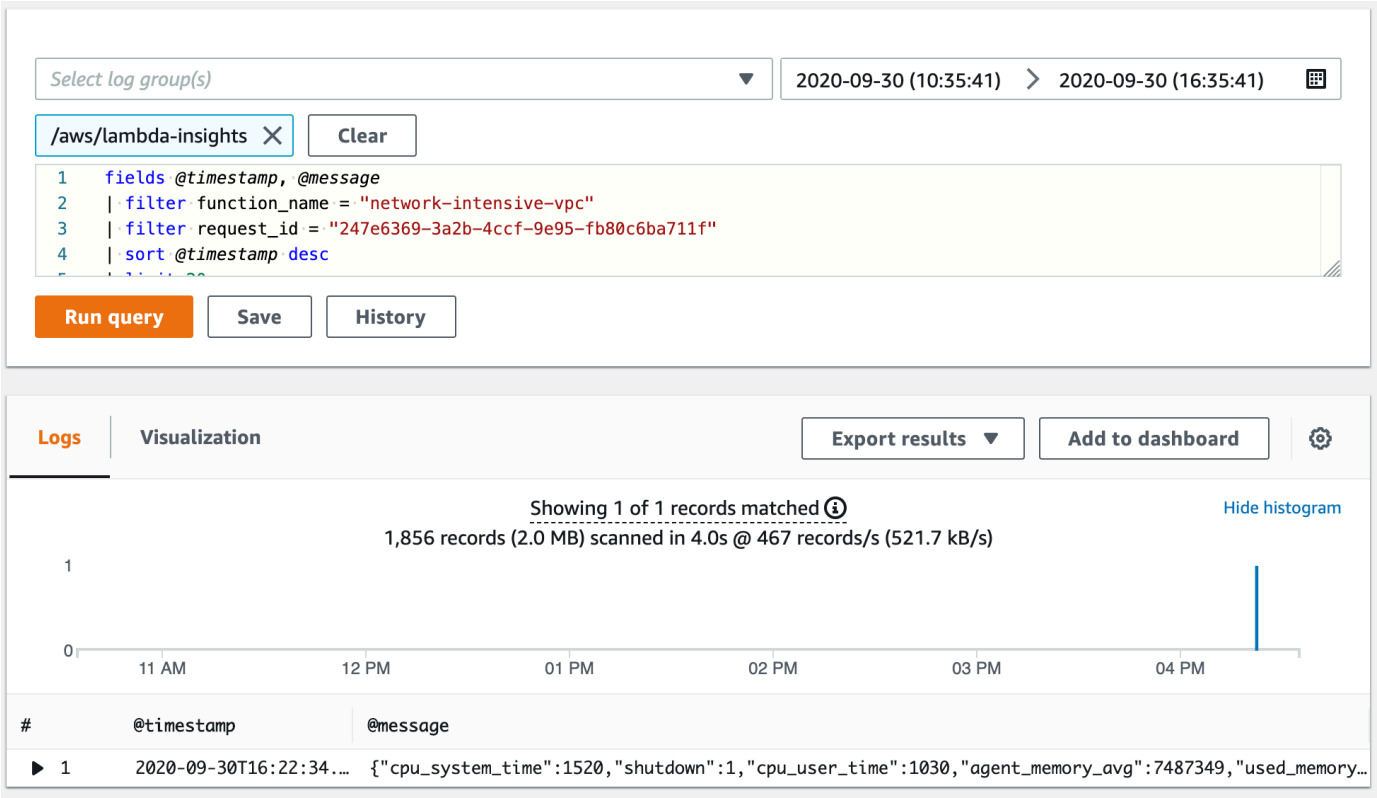
4. Selecione a lista suspensa Filter metrics by function name (Filtrar métricas por nome da função) e selecione a função.
5. Para visualizar os Most recent 1000 application logs (1000 logs de aplicações mais recentes), selecione a guia Application logs (Logs de aplicações).
6. Revise o Timestamp (Time stamp) e a Message (Mensagem) para identificar a solicitação de invocação que você deseja solucionar.

7. Para mostrar as Most recent 1000 invocations (1000 invocações mais recentes), selecione a guia Invocations (Invocações).
8. Selecione o Timestamp (Time stamp) ou a Message (Mensagem) para a solicitação de invocação que você deseja solucionar.



9. Selecione a lista suspensa View logs (Visualizar logs) e, depois, selecione View performance logs (Visualizar logs de performance).

   Uma consulta gerada automaticamente para a função é aberta no painel do Logs Insights.
10. Selecione Run query (Executar consulta) para gerar uma mensagem Logs para a solicitação de invocação.



Próximas etapas

- Saiba como criar um painel do CloudWatch Logs noCriar um painelnoGuia do usuário do Amazon CloudWatch.
- Saiba como adicionar consultas a um painel do CloudWatch Logs em Adicionar consulta ao painel ou exportar resultados de consulta no Manual do usuário do Amazon CloudWatch.

Usando o CodeGuru Profiler com sua função Lambda

Você pode usar o Amazon CodeGuru Profiler para obter informações sobre o desempenho em tempo de execução de suas funções do Lambda. Esta página descreve como ativar o CodeGuru Profiler a partir do console Lambda.

Seções

Tempos de execução compatíveis

Você pode ativar o CodeGuru Profiler no console Lambda se o tempo de execução da sua função for Python3.8, Python3.9, Java 8 com Amazon Linux 2, Java 11 ou Java 17. Para versões adicionais de tempo de execução, você pode ativar o CodeGuru Profiler manualmente.

- Para tempos de execução Java, consulteCriando o perfil de seus aplicativos Java que são executados emAWS Lambda.
- Para tempos de execução do Python, consultePerfilando seus aplicativos Python que são executados emAWS Lambda.

> **Note**
>
> CodeGuru Atualmente, o Profiler suporta apenas funções que usam a arquitetura x86_64.

Ativando o CodeGuru Profiler a partir do console Lambda

Esta seção descreve como ativar o CodeGuru Profiler a partir do console Lambda.

Para ativar o CodeGuru Profiler a partir do console Lambda

1. Abra a página Funções do console do Lambda.
2. Escolha a função.

3. Escolha a guia Configuração.
4. No painel Monitoring and operations tools (Ferramentas de monitoramento e operações), selecione Edit (Editar).
5. Em Amazon CodeGuru Profiler, ative a criação de perfil de código.
6. Escolha Salvar.

Após a ativação, cria CodeGuru automaticamente um grupo de criadores de perfil com o nome`aws-lambda-<your-function-name>`. Você pode alterar o nome no CodeGuru console.

O que acontece quando você ativa o CodeGuru Profiler no console Lambda?

Quando você ativa o CodeGuru Profiler a partir do console, o Lambda faz automaticamente o seguinte em seu nome:

- O Lambda adiciona uma camada CodeGuru Profiler à sua função. Para obter mais detalhes, consulte Usar AWS Lambda camadas no Guia do usuário do Amazon CodeGuru Profiler.
- O Lambda também adiciona variáveis de ambiente à sua função. O valor exato varia de acordo com o runtime.

  Variáveis de ambiente

Tempos de execução	Chave	Valor
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`

- O Lambda adiciona o`AmazonCodeGuruProfilerAgentAccess`à função de execução da sua função.

> **Note**
>
> Quando você desativa o CodeGuru Profiler do console, o Lambda remove automaticamente a CodeGuru camada Profiler da sua função. No entanto, o Lambda não remove as variáveis de ambiente nem a política `AmazonCodeGuruProfilerAgentAccess` da função de execução.

Próximas etapas

- Saiba mais sobre os dados coletados pelo seu grupo de criadores de perfil em Como trabalhar com visualizações no Guia do usuário do Amazon CodeGuru Profiler.

Exemplo de fluxos de trabalho usando outros serviços do AWS

O AWS Lambda pode ser integrado a outros serviços da AWS para ajudá-lo a monitorar, rastrear, depurar e solucionar problemas de suas funções do Lambda. Esta página mostra fluxos de trabalho que você pode usar com o AWS X-Ray e o AWS Trusted Advisor para rastrear e solucionar problemas das funções do Lambda.

Seções

Pré-requisitos

A seção a seguir descreve as etapas para usar o AWS X-Ray e o Trusted Advisor para solucionar problemas em suas funções do Lambda.

Usar o AWS X-Ray

O AWS X-Ray precisa ser ativado no console do Lambda para a conclusão dos fluxos de trabalho do AWS X-Ray nessa página. Se sua função de execução não tiver as permissões necessárias, o console do Lambda tentará adicioná-las à função.

Para ativar o AWS X-Ray no console do Lambda

1. Abra a página Funções do console do Lambda.
2. Escolha a função.
3. Escolha a guia Configuração.
4. No painel Monitoring tools (Ferramentas de monitoramento), selecione Edit (Editar).
5. Em AWS X-Ray, ative o Rastreamento ativo.
6. Escolha Salvar.

Usar o AWS Trusted Advisor

O AWS Trusted Advisor inspeciona o seu ambiente da AWS e faz recomendações sobre como economizar dinheiro, melhorar a performance e a disponibilidade do sistema ou ajuda a corrigir falhas de segurança. Você pode usar as verificações do Trusted Advisor para avaliar as funções e aplicações do Lambda em sua conta da AWS. As verificações fornecem etapas recomendadas a tomar e recursos para obter mais informações.

- Para obter mais informações sobre os planos de suporte da AWS para verificações do Trusted Advisor, consulte Planos de suporte.
- Para obter mais informações sobre as verificações para o Lambda, consulte a Lista de verificação de práticas recomendadas do AWS Trusted Advisor.
- Para obter mais informações sobre como usar o console do Trusted Advisor, consulte Comece a usar o AWS Trusted Advisor.
- Para obter instruções sobre como permitir e negar acesso ao console do Trusted Advisor, consulte os Exemplos de políticas do IAM.

Definição de preço

- Com o AWS X-Ray, você só paga pelo que usa, de acordo com o número de rastreamentos gravados, recuperados e verificados. Para obter mais informações, consulte Preços do AWS X-Ray.
- As verificações de otimização de custos do Trusted Advisor estão incluídas nas assinaturas de suporte Business e Enterprise da AWS. Para obter mais informações, consulte Preços do AWS Trusted Advisor.

Exemplo de fluxo de trabalho do AWS X-Ray para visualização de um mapa de rastreamento

Se você tiver ativadoAWS X-Ray, poderá visualizar um mapa de rastreamento no CloudWatch console. Um mapa de rastreamento exibe os recursos e os endpoints de serviço como nós e destaca o tráfego, a latência e os erros de cada nó e das suas conexões.

É possível escolher um nó para ver insights detalhados sobre as métricas, os logs e os rastreamentos correlacionados associados a essa parte do serviço. Isso permite investigar os problemas e seus efeitos no aplicativo.

Para visualizar o mapa de rastreamento e os traços usando o CloudWatch console

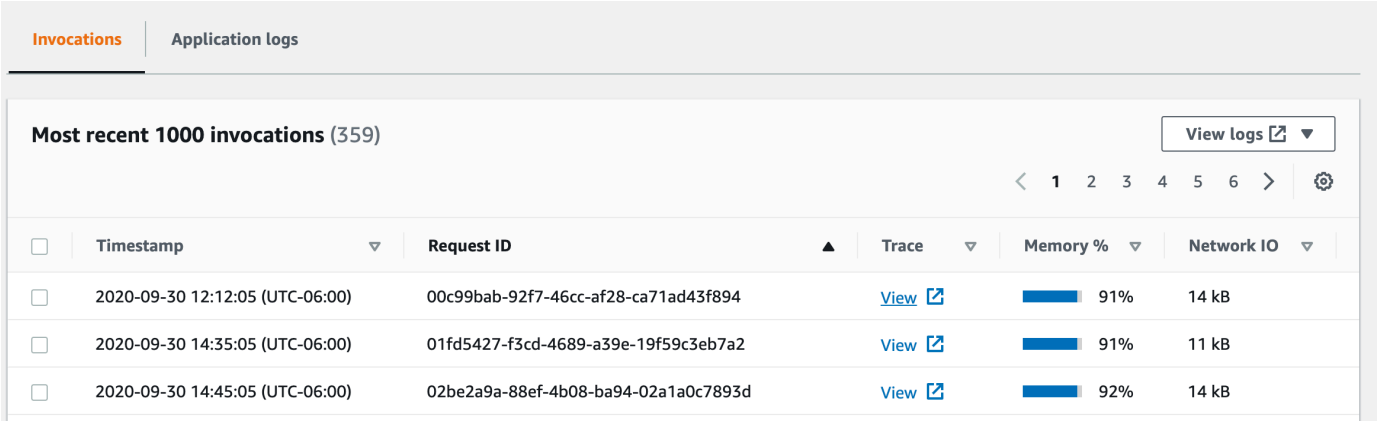
1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha Monitoramento.
4. Selecione Rastreamentos no X-Ray.
5. Escolha Mapa de rastreamento em Rastreamentos no X-Ray no painel de navegação esquerdo.
6. Escolha entre os intervalos de tempo predefinidos ou escolha um intervalo de tempo personalizado.
7. Para solucionar problemas de solicitações, selecione um filtro.

Exemplo de fluxo de trabalho do AWS X-Ray para a exibição de detalhes de rastreamento

Se você tiver habilitadoAWS X-Ray, poderá usar a visualização de função única no painel do CloudWatch Lambda Insights para mostrar os dados de rastreamento distribuídos de um erro de invocação de função. Por exemplo, se a mensagem de logs da aplicação mostrar um erro, você poderá abrir o mapa de rastreamento para ver os dados de rastreamento distribuídos e os outros serviços que estão processando a transação.

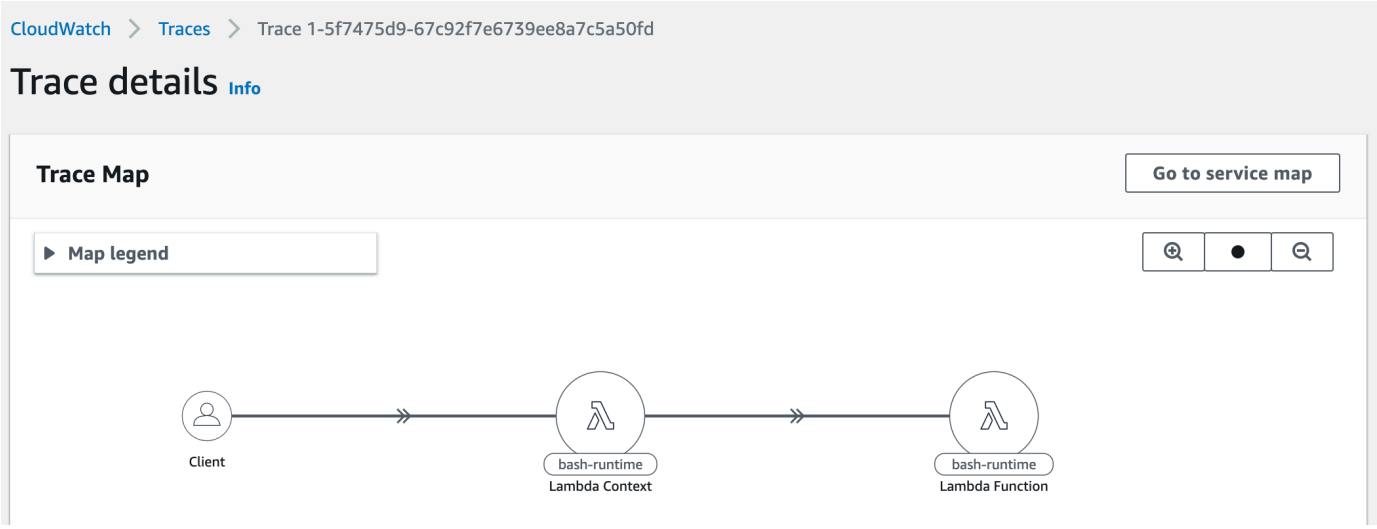
Para exibir detalhes de rastreamento de uma função

1. Abra a visualização de função única no CloudWatch console.
2. Selecione a aba Logs de aplicativos.
3. Use o Timestamp ou a Mensagem para identificar a solicitação de invocação que deseja solucionar.
4. Para mostrar as Most recent 1000 invocations (1000 invocações mais recentes), selecione a guia Invocations (Invocações).



5. Selecione a coluna ID da solicitação para classificar as entradas em ordem alfabética crescente.
6. Na coluna Rastreamento, selecione Exibir.

   A página Detalhes do rastreamento é aberta na visualização do mapa de rastreamento.



Exemplo de fluxo de trabalho do AWS Trusted Advisor para exibir recomendações

O Trusted Advisor verifica as funções do Lambda em todas as regiões da AWS para identificar as funções com o maior potencial de economia de custos, além de fornecer recomendações acionáveis para otimização. Ele analisa os seus dados de uso do Lambda, como runtime da função, duração faturada, memória utilizada, memória configurada, configuração de tempo limite e erros.

Por exemplo, a verificação de Funções Lambda com Alta Taxa de Erro recomenda que você use AWS X-Ray ou CloudWatch detecte erros com suas funções Lambda.

Para verificar se há funções com altas taxas de erro

1. Abra o console do Trusted Advisor.
2. Escolha a categoria Cost Optimization (Otimização de custos).
3. Role para baixo até Funções do AWS Lambda com altas taxas de erro. Expanda a seção para ver os resultados e as ações recomendadas.

Próximas etapas

- Saiba mais sobre como integrar rastreamentos, métricas, logs e alarmes em Using the X-Ray trace map.
- Saiba mais sobre como obter uma lista de verificações do Trusted Advisor em Como usar o Trusted Advisor como um serviço Web.

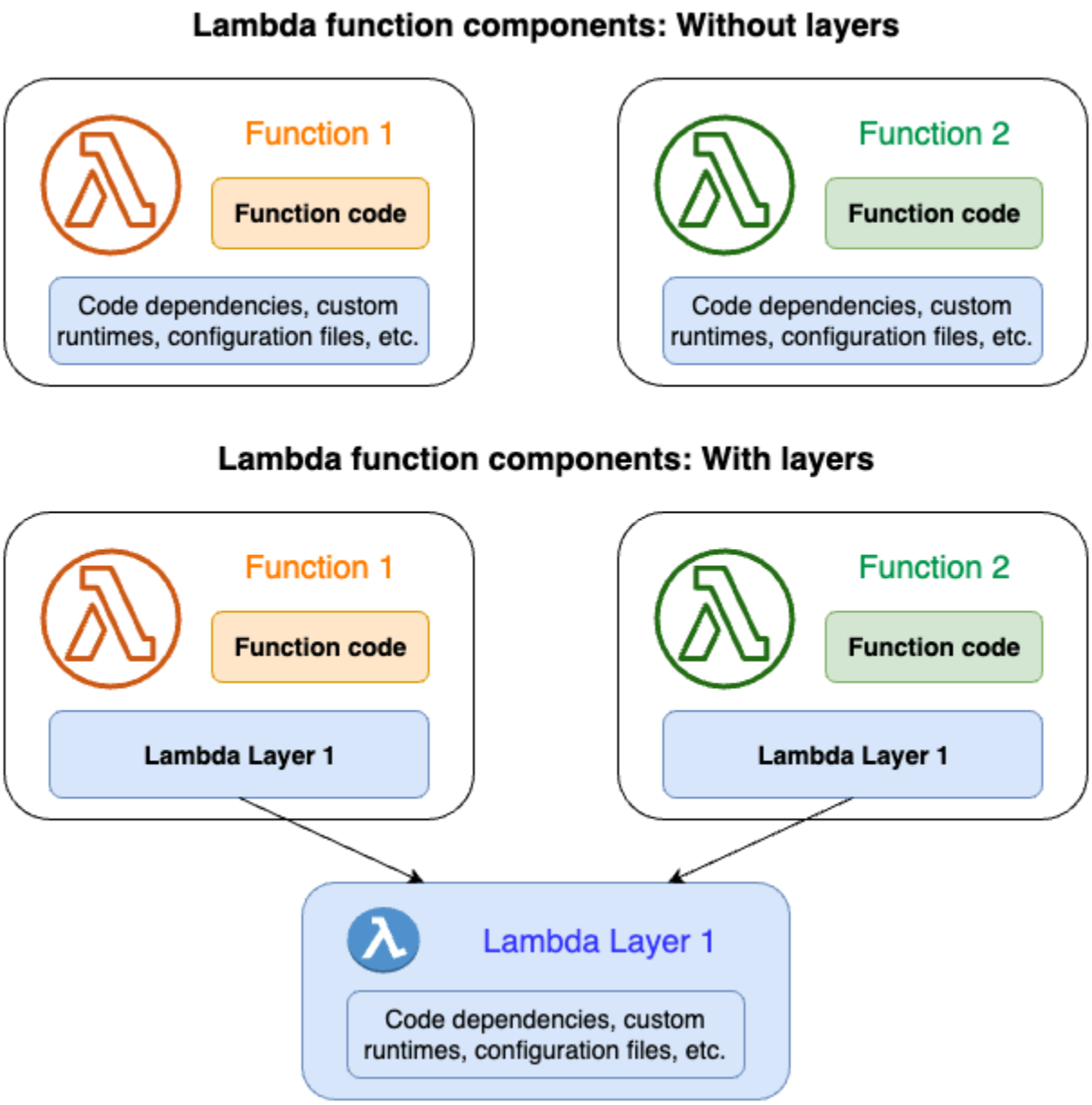
Gerenciar dependências do Lambda com camadas

Uma camada do Lambda é um arquivo .zip que pode conter código ou dados adicionais. As camadas geralmente contêm dependências de biblioteca, um runtime personalizado ou arquivos de configuração.

Há vários motivos pelos quais você pode considerar o uso de camadas:

- Para reduzir o tamanho de seus pacotes de implantação. Em vez de incluir todas as dependências da função junto com o código da função no pacote de implantação, coloque-as em uma camada. Isso mantém os pacotes de implantação pequenos e organizados.
- Para separar a lógica da função central das dependências. Com camadas, você pode atualizar suas dependências de função independentemente do código da função, e vice-versa. Isso promove a separação de preocupações e ajuda você a se concentrar na lógica da sua função.
- Para compartilhar dependências em várias funções. Depois de criar uma camada, você pode aplicá-la a qualquer número de funções em sua conta. Sem camadas, você precisa incluir as mesmas dependências em cada pacote de implantação individual.
- Para usar o editor de código do console do Lambda. O editor de código é uma ferramenta útil para testar rapidamente pequenas atualizações de código de funções. No entanto, você não pode usar o editor se o tamanho do pacote de implantação for muito grande. O uso de camadas reduz o tamanho do pacote e pode desbloquear o uso do editor de código.

O diagrama a seguir ilustra as diferenças arquitetônicas de alto nível entre duas funções que compartilham dependências. Uma usa camadas do Lambda e a outra, não.



Quando você adiciona uma camada a uma função, o Lambda extrai o conteúdo da camada para o diretório `/opt` no ambiente de execução da sua função. Todos os runtimes do Lambda com suporte nativo incluem caminhos para diretórios específicos dentro do diretório `/opt`. Isso dá à sua função acesso ao conteúdo da camada. Para obter mais informações sobre esses caminhos específicos e como empacotar adequadamente suas camadas, consulte the section called “Empacotar camadas”.

Você pode incluir até cinco camadas por função. Também é possível usar camadas somente com funções do Lambda implantadas como um arquivo .zip. Para funções definidas como uma imagem de contêiner, empacote seu runtime preferido e todas as dependências de código ao criar a imagem

de contêiner. Para obter mais informações, consulte Working with Lambda layers and extensions in container images no AWS Compute Blog.

Tópicos

Como usar camadas

Para criar uma camada, empacote suas dependências em um arquivo .zip, da mesma forma como você cria um pacote de implantação normal. Mais especificamente, o processo geral de criação e o uso de camadas envolve estas três etapas:

- Primeiro, empacote o conteúdo da camada. Isso significa criar um arquivo .zip. Para ter mais informações, consulte the section called “Empacotar camadas”.
- Em seguida, crie a camada no Lambda. Para ter mais informações, consulte the section called “Criar e excluir camadas”.
- Adicione a camada às funções. Para ter mais informações, consulte the section called “Adicionar camadas”.

Camadas e versões da camada

Uma versão da camada é um snapshot imutável de uma versão específica de uma camada. Quando você cria uma nova camada, o Lambda cria uma nova versão da camada com um número de versão de 1. Sempre que você publica uma atualização na camada, o Lambda incrementa o número da versão e cria uma nova versão da camada.

Cada versão da camada é identificada por um nome do recurso da Amazon (ARN) único. Ao adicionar uma camada à função, você deve especificar a versão exata da camada que deseja usar.

Empacotar seu conteúdo de camada

Uma camada do Lambda é um arquivo .zip que pode conter código ou dados adicionais. As camadas geralmente contêm dependências de biblioteca, um runtime personalizado ou arquivos de configuração.

Esta seção explica como empacotar adequadamente o conteúdo da camada. Para obter mais informações conceituais sobre camadas e por que você pode considerar usá-las, consulte Camadas do Lambda.

A primeira etapa para criar uma camada é agrupar todo o conteúdo da camada em um arquivo .zip. Devido às funções do Lambda serem executadas no Amazon Linux, seu conteúdo da camada deve ser capaz de compilar e criar em um ambiente Linux.

Para garantir o funcionamento correto do seu conteúdo de camada em um ambiente Linux, recomendamos criar seu conteúdo de camada usando uma ferramenta como o Docker ou o AWS Cloud9. O AWS Cloud9 é um ambiente de desenvolvimento integrado (IDE) baseado em nuvem que fornece acesso integrado a um servidor Linux para execução e teste de código. Para obter mais informações, consulte Using Lambda layers to simplify your development process no AWS Blog de computação.

Tópicos

- Caminhos da camada para cada runtime do Lambda

Caminhos da camada para cada runtime do Lambda

Quando você adiciona uma camada a uma função, o Lambda carrega o conteúdo da camada no diretório `/opt` desse ambiente de execução. Para cada runtime do Lambda, a variável `PATH` já inclui caminhos de pasta específica no diretório `/opt`. Para garantir que a variável `PATH` colete o conteúdo da camada, o arquivo .zip da camada deve ter suas dependências nos seguintes caminhos de pasta:

Caminhos de camada para cada runtime do Lambda

Runtime	Path
Node.js	`nodejs/node_modules`
	`nodejs/node14/node_modules` (`NODE_PATH`)

Runtime	Path
	`nodejs/node16/node_modules` (`NODE_PATH`)
	`nodejs/node18/node_modules` (`NODE_PATH`)
Python	`python`
	`python/lib/` *`python3.x`* `/site-packages` (diretórios do site)
Java	`java/lib` (`CLASSPATH`)
Ruby	`ruby/gems/3.2.0` (`GEM_PATH`)
	`ruby/lib` (`RUBYLIB`)
Todos os runtimes	`bin` (`PATH`)
	`lib` (`LD_LIBRARY_PATH`)

Os exemplos a seguir mostram como você pode estruturar as pastas no seu arquivo .zip da camada.

Node.js

Example estrutura de arquivos do AWS X-Ray SDK for Node.js

```
xray-sdk.zip
# nodejs/node_modules/aws-xray-sdk
```

Python

Example estrutura de arquivos da biblioteca Requests

```
layer_content.zip
# python
    # lib
        # python3.11
            # site-packages
                # requests
                # <other_dependencies> (i.e. dependencies of the requests package)
```

```
                # ...
```

Ruby

Example estrutura de arquivos da 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 estrutura de arquivos do arquivo JAR do Jackson

```
layer_content.zip
# java
    # lib
        # jackson-core-2.17.0.jar
        # <other potential dependencies>
        # ...
```

All

Example estrutura de arquivos da biblioteca jq

```
jq.zip
# bin/jq
```

Para obter instruções específicas da linguagem sobre como empacotar, criar e adicionar uma camada, consulte as páginas a seguir:

- Python: the section called “Camadas”
- Java: the section called “Camadas”

Criar e excluir camadas no Lambda

Uma camada do Lambda é um arquivo .zip que pode conter código ou dados adicionais. As camadas geralmente contêm dependências de biblioteca, um runtime personalizado ou arquivos de configuração.

Esta seção explica como criar e excluir camadas no Lambda. Para obter mais informações conceituais sobre camadas e por que você pode considerar usá-las, consulte Camadas do Lambda.

Depois de ter empacotado seu conteúdo de camada, a próxima etapa é criar a camada no Lambda. Esta seção demonstra como criar e excluir camadas usando somente o console do Lambda ou a API do Lambda. Para criar uma camada usando o AWS CloudFormation, consulte the section called “Camadas com o AWS CloudFormation”. Para criar uma camada usando o AWS Serverless Application Model (AWS SAM), consulte the section called “Camadas com o AWS SAM”.

Tópicos

Criar uma camada

Para criar uma camada, você pode fazer o upload do arquivo .zip em sua máquina local ou no Amazon Simple Storage Service (Amazon S3). O Lambda extrai o conteúdo da camada para o diretório `/opt` ao configurar o ambiente de execução para a função.

As camadas podem ter uma ou mais versões da camada. Quando você cria uma camada, o Lambda define a versão da camada para a versão 1. Você pode alterar as permissões em uma versão da camada existente a qualquer momento. No entanto, para atualizar o código ou fazer outras alterações na configuração, é preciso criar uma nova versão da camada.

Para criar uma camada (console)

1. Abra a página Layers (Camadas) do console do Lambda.
2. Escolha Criar camada.
3. Em Configuração de camada, insira um nome para sua camada em Nome.
4. (Opcional) Em Descrição, insira uma descrição para a sua camada.
5. Para fazer upload do código da camada, siga um destes procedimentos:

- Para carregar um arquivo.zip do seu computador, escolha Upoad a .zip file (Fazer upload de um arquivo .zip). Selecione Upload (Fazer upload) para escolher seu arquivo .zip local.
- Para fazer upload de um arquivo do Amazon S3, escolhaPara fazer upload de um arquivo do Amazon S3. Então, em Amazon S3 link URL (URL do link do Amazon S3), insira um link para o arquivo.

6. (Opcional) Em Arquiteturas compatíveis, escolha um valor ou ambos os valores. Para ter mais informações, consulte the section called “Conjuntos de instruções (ARM/x86)”.
7. (Opcional) Para Runtimes compatíveis, escolha os runtimes com os quais sua camada é compatível.
8. (Opcional) Em License (Licença), insira todas as informações necessárias sobre licença.
9. Escolha Criar.

Como alternativa, você também pode usar a PublishLayerVersionAPI para criar uma camada. Por exemplo, você pode usar o comando `publish-layer-version` da AWS Command Line Interface (CLI) com nome, descrição e arquivo .zip especificados. As informações da licença, os runtimes compatíveis e os parâmetros de arquitetura compatíveis são opcionais.

```
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"
```

Você deve ver saída semelhante a:

```
{
    "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"
    ]
}
```

Toda vez que você chama `publish-layer-version`, você cria uma nova versão dessa camada.

Excluir uma versão da camada

Para excluir uma versão de camada, use a DeleteLayerVersionAPI. Por exemplo, você pode usar o comando `delete-layer-version` da CLI com o nome da camada e a versão da camada especificados.

```
aws lambda delete-layer-version --layer-name my-layer --version-number 1
```

Ao excluir uma versão de camada, não é mais possível configurar uma função do Lambda para usá-la. No entanto, todas as funções que já usem a versão continua a ter acesso a ela. Além disso, o Lambda nunca reutiliza números de versão para o nome de uma camada.

Adicionar camadas às funções

Uma camada do Lambda é um arquivo .zip que pode conter código ou dados adicionais. As camadas geralmente contêm dependências de biblioteca, um runtime personalizado ou arquivos de configuração.

Esta seção explica como adicionar uma camada a uma função do Lambda. Para obter mais informações conceituais sobre camadas e por que você pode considerar usá-las, consulte Camadas do Lambda.

Antes de configurar uma função do Lambda para usar uma camada, é necessário:

- Empacotar o conteúdo da sua camada
- Criar uma camada no Lambda
- Certifique-se de ter permissão para chamar a GetLayerVersionAPI na versão da camada. Para funções em sua Conta da AWS, você deve adicionar essa permissão em sua política de usuário. Para usar uma camada em outra conta, o proprietário dessa conta deve conceder permissão à sua conta em uma política baseada em recursos. Para ver exemplos, consulte the section called "Conceder acesso de camada a outras contas".

É possível adicionar até cinco camadas a uma função do Lambda. O tamanho total descompactado da função e de todas as camadas não pode exceder a cota de tamanho do pacote de implantação descompactado de 250 MB. Para ter mais informações, consulte Cotas Lambda.

Suas funções podem continuar usando qualquer versão de camada que você já tenha adicionado, mesmo após a exclusão dessa versão ou após a revogação de sua permissão para acessar a camada. No entanto, você não pode criar uma nova função que use uma versão de camada excluída.

> **Note**
>
> Certifique-se de que as camadas adicionadas a uma função sejam compatíveis com a arquitetura do runtime e do conjunto de instruções da função.

Para adicionar tags a uma função (console)

1. Abra a página Funções do console do Lambda.

2. Escolha a função a ser configurada.
3. Em Camadas, selecione Adicionar uma camada
4. Em Escolher uma camada, escolha uma origem da camada:

    a. Para as origens da camada Camadas da AWS ou Camadas personalizadas, escolha uma camada no menu suspenso. Em Versão, escolha uma versão da camada no menu suspenso.

    b. Para a origem da camada Especificar um ARN, insira um ARN na caixa de texto e escolha Verificar. Em seguida, escolha Adicionar.

A ordem na qual você adiciona as camadas é a ordem na qual o Lambda mescla posteriormente o conteúdo da camada no ambiente de execução. É possível alterar a ordem de mesclagem das camadas usando o console.

Para atualizar a ordem de mesclagem das camadas para sua função (console)

1. Abra a página Funções do console do Lambda.
2. Escolha a função a ser configurada.
3. Em Layers (Camadas), escolha Edit (Editar)
4. Escolha uma das camadas.
5. Escolha Merge earlier (Mesclar antes) ou Merge later (Mesclar depois) para ajustar a ordem das camadas.
6. Escolha Salvar.

As camadas contêm versionamento. O conteúdo de cada versão da camada é imutável. O proprietário de uma camada pode liberar novas versões dela para fornecer conteúdo atualizado. É possível usar o console para atualizar a versão da camada anexada às suas funções.

Para atualizar versões da camada para sua função (console)

1. Abra a página Camadas do console do Lambda.
2. Escolha a camada para a qual você deseja atualizar a versão.
3. Escolha a guia Funções usando esta versão.
4. Escolha as funções que você deseja modificar e, em seguida, escolha Editar.
5. Em Versão da camada, escolha a versão da camada a ser alterada.

6. Escolha Atualizar funções.

Você não pode atualizar as versões da camada de funções nas contas da AWS.

Tópicos

Acessar o conteúdo da camada da sua função

Quando a função do Lambda inclui camadas, o Lambda extrai o conteúdo da camada para o diretório `/opt` no ambiente de execução da função. O Lambda extrai as camadas na ordem (baixa para alta) listada pela função. O Lambda mescla pastas com o mesmo nome. Se o mesmo arquivo aparecer em várias camadas, a função utiliza a versão na última camada extraída.

Cada runtime do Lambda adiciona pastas do diretório `/opt` específicas à variável `PATH`. Seu código de função pode acessar o conteúdo da camada sem precisar especificar o caminho. Para obter mais informações sobre as configurações de caminhos no ambiente de execução do Lambda, consulte the section called “Variáveis de ambiente com runtime definido”.

Consulte the section called “Caminhos da camada para cada runtime do Lambda” para saber onde incluir suas bibliotecas ao criar uma camada.

Se você estiver usando um runtime Node.js ou Python, poderá usar o editor de código integrado no console do Lambda. Você deve ser capaz de importar qualquer biblioteca que tenha adicionado como uma camada para a função atual.

Encontrar informações da camada

Para encontrar camadas em sua conta que sejam compatíveis com o tempo de execução da sua função, use a ListLayersAPI. Por exemplo, é possível usar o seguinte comando `list-layers` da AWS Command Line Interface (CLI):

```
aws lambda list-layers --compatible-runtime python3.9
```

Você deve ver saída semelhante a:

```
{
```

```
    "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",
                ]
            }
        }
    ]
}
```

Para listar todas as camadas da sua conta, omitia a opção `--compatible-runtime`. Os detalhes da resposta mostram a versão mais recente de cada camada.

Você também pode obter a versão mais recente de uma camada usando a ListLayerVersionsAPI. Por exemplo, é possível usar o seguinte comando `list-layer-versions` da CLI:

```
aws lambda list-layer-versions --layer-name my-layer
```

Você deve ver saída semelhante a:

```
{
    "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"
            ]
        }
    ]
}
```

Usar o AWS CloudFormation com camadas

Você pode usar o AWS CloudFormation para criar uma camada e associá-la à sua função do Lambda. O modelo de exemplo a seguir cria uma camada chamada `my-lambda-layer` e a anexa à função do Lambda usando a propriedade Camadas.

```
---
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
```

Usar o AWS SAM com camadas

Você pode usar o AWS Serverless Application Model (AWS SAM) para automatizar a criação de camadas em sua aplicação. O tipo de recurso `AWS::Serverless::LayerVersion` cria uma versão da camada à qual você pode fazer referência na configuração da função do 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
```

Ampliar funções do Lambda usando extensões do Lambda

É possível usar extensões do Lambda para aumentar as funções do Lambda. Por exemplo, use extensões do Lambda para integrar funções com suas ferramentas preferidas de monitoramento, observação, segurança e governança. Você pode escolher entre várias ferramentas que os parceiros do AWS Lambda fornecem, ou pode criar suas próprias extensões do Lambda.

O Lambda é compatível com extensões internas e externas. Uma extensão externa é executada como um processo independente no ambiente de execução e continua a ser executada após a invocação da função ser totalmente processada. Como as extensões são executadas como processos separados, elas podem ser escritas em uma linguagem diferente da função. Todos os Runtimes do Lambda são compatíveis com extensões.

Uma extensão interna é executada como parte do processo de runtime. A função acessa extensões internas usando scripts wrapper ou mecanismos no processo, como `JAVA_TOOL_OPTIONS`. Para ter mais informações, consulte Modificar o ambiente de runtime.

É possível adicionar extensões a uma função usando o console do Lambda, a AWS Command Line Interface (AWS CLI) ou serviços e ferramentas da infraestrutura como código (IaC), como o AWS CloudFormation, o AWS Serverless Application Model (AWS SAM) e o Terraform.

Você é cobrado pelo tempo de execução consumido pela extensão (em incrementos de 1 ms). Não há custo para instalar suas próprias extensões. Para obter mais informações sobre a definição de preço para extensões, consulte Definição de preço do AWS Lambda. Para obter informações de definição de preço sobre extensões de parceiros, consulte os sites desses parceiros. Consulte the section called “Parceiros de extensões” para obter uma lista de extensões oficiais de parceiros.

Para ver um tutorial sobre extensões e como usá-las com suas funções do Lambda, consulte o Workshop de extensões do AWS Lambda.

Tópicos

- Usar a API de extensões do Lambda para criar extensões
- API de Telemetria do Lambda

Ambiente de execução

O Lambda invoca a função em um ambiente de execução, que fornece um ambiente do runtime seguro e isolado. O ambiente de execução gerencia os recursos necessários para executar sua função e fornece suporte ao ciclo de vida para o runtime e extensões da função.

O ciclo de vida do ambiente de execução inclui as seguintes fases:

- **`Init`**: nessa fase, o Lambda cria ou descongela um ambiente de execução com os recursos configurados, faz download do código da função e de todas as camadas, inicializa todas as extensões e o runtime, em seguida executa o código de inicialização da função (o código fora do handler principal). O`Init`ocorre durante a primeira invocação, ou antes de invocações de função se você tiver habilitadoSimultaneidade provisionada.

  O`Init`é dividida em três subfases:`Extension init`,`Runtime init`, e`Function init`. Essas subfases garantem que todas as extensões e o runtime concluam suas tarefas de configuração antes que o código da função seja executado.

  Quando o Lambda SnapStart está ativado, a fase `Init` ocorre ao você publicar uma versão da função. O Lambda salva um snapshot do estado da memória e do disco do ambiente de execução inicializado, mantém o snapshot criptografado e o armazena em cache para acesso de baixa latência. Se você tiver um hook de runtime `beforeCheckpoint`, então, o código será executado ao final da fase `Init`.
- **`Restore`** (somente para SnapStart): quando você invoca uma função do SnapStart pela primeira vez e à medida que aumenta a escala verticalmente da função, o Lambda retoma novos ambientes de execução do snapshot retido, em vez de inicializar a função do zero. Se você tiver um hook de runtime `afterRestore()`, o código será executado ao final da fase `Restore`. Você terá cobranças pela duração dos hooks de runtime `afterRestore()`. O runtime (JVM) deve ser carregado e os hooks de runtime `afterRestore()` devem ser concluídos dentro do limite de tempo limite (dez segundos). Caso contrário, você obterá uma SnapStartTimeoutException. Quando a fase `Restore` é concluída, o Lambda invoca o manipulador de função (a Fase de invocação).
- **`Invoke`**: Nesta fase, o Lambda chama o manipulador de função. Depois que a função é executada até a conclusão, o Lambda se prepara para manipular outra invocação de função.

- **`Shutdown`**: Esta fase é acionada se a função do Lambda não receber quaisquer invocações por um período de tempo. Na fase `Shutdown`, o Lambda encerra o runtime, alerta as extensões para permitir que elas parem de forma limpa e, em seguida, remove o ambiente. Lambda envia um`Shutdown`para cada extensão, que informa a extensão que o ambiente está prestes a ser encerrado.

Durante o`Init`, o Lambda extrai camadas contendo extensões para o`/opt`no ambiente de execução. O Lambda procura extensões no`/opt/extensions/`, interpreta cada arquivo como um bootstrap executável para iniciar a extensão e inicia todas as extensões em paralelo.

Impacto na performance e nos recursos

O tamanho das extensões da função conta para o limite de tamanho do pacote de implantação. Para um arquivo .zip, o tamanho total descompactado da função e de todas as extensões não pode exceder o limite de tamanho do pacote de implantação descompactado de 250 MB.

As extensões podem afetar a performance da função porque compartilham recursos de função, como CPU, memória e armazenamento. Por exemplo, se uma extensão executa operações de computação intensiva, você pode ver o aumento da duração da execução da função.

Cada extensão deve concluir sua inicialização antes de o Lambda invocar a função. Portanto, uma extensão que consome tempo de inicialização significativo pode aumentar a latência da invocação da função.

Para medir o tempo adicional que a extensão leva após a execução da função, você pode usar a[métrica da função](#) `PostRuntimeExtensionsDuration`. Para medir o aumento da memória usada, você pode usar a métrica `MaxMemoryUsed`. Para entender o impacto de uma extensão específica, você pode executar diferentes versões das funções lado a lado.

Permissões

As extensões têm acesso aos mesmos recursos que as funções. Como as extensões são executadas no mesmo ambiente que a função, as permissões são compartilhadas entre a função e a extensão.

Para um arquivo .zip, é possível criar um modelo do AWS CloudFormation para simplificar a tarefa de anexar a mesma configuração de extensão, incluindo permissões do AWS Identity and Access Management (IAM), a várias funções.

Configuração de extensões do Lambda

Configurar extensões (arquivamento de arquivo.zip)

Você pode adicionar uma extensão à sua função como uma camada do Lambda. O uso de camadas permite compartilhar extensões em toda a organização ou com toda a comunidade de desenvolvedores do Lambda. É possível adicionar uma ou mais extensões a uma camada. É possível registrar até 10 extensões para uma função.

Adicione a extensão à função usando o mesmo método que faria para qualquer camada. Para ter mais informações, consulte Camadas do Lambda.

Adicionar uma extensão à função (console)

1. Abra a página Funções do console do Lambda.
2. Escolha uma função.
3. Escolha a guia Código se ainda não estiver selecionada.
4. Em Camadas, escolha Editar.
5. Em Choose a layer (Selecionar uma camada), selecione Specify an ARN (Especificar um ARN).
6. Em Specify an ARN (Especificar um ARN), insira o nome de recurso da Amazon (ARN) de uma camada de extensão.
7. Escolha Adicionar.

Uso de extensões em imagens de contêiner

Você pode adicionar extensões à imagem do contêiner. A configuração de imagem de contêiner ENTRYPOINT especifica o processo principal para a função. Configure a configuração ENTRYPOINT no Dockerfile ou como uma substituição na configuração da função.

Você pode executar vários processos dentro de um contêiner. Lambda gerencia o ciclo de vida do processo principal e quaisquer processos adicionais. O Lambda usa oAPI Extensions dopara gerenciar o ciclo de vida da extensão.

Exemplo: Adicionando uma extensão externa

Uma extensão externa é executada em um processo separado da função Lambda. O Lambda inicia um processo para cada extensão no`/opt/extensions/`Diretório. O Lambda usa a API de

extensões para gerenciar o ciclo de vida da extensão. Depois que a função for executada até a conclusão, o Lambda envia um evento `Shutdown` para cada extensão externa.

Example de adicionar uma extensão externa a uma imagem 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
```

Próximas etapas

Para saber mais sobre extensões, recomendamos os seguintes recursos:

- Para obter um exemplo básico de trabalho, consulte Building Extensions for AWS Lambda no AWS Compute Blog.
- Para obter informações sobre extensões fornecidas pelos parceiros da AWS Lambda, consulte Introducing AWS Lambda Extensions no AWS Compute Blog.
- Para ver exemplos de extensões e scripts de wrapper disponíveis, consulte AWS LambdaExtensões no GitHub repositório AWS Samples.

Parceiros de extensões do AWS Lambda

O AWS Lambda fez uma parceria com várias entidades de terceiros para fornecer extensões que se integram às suas funções Lambda. A lista a seguir detalha extensões de terceiros que estão prontas para uso a qualquer momento.

- AppDynamics: fornece instrumentação automática das funções do Lambda no Node.js ou Python, oferecendo visibilidade e alertando sobre a performance das funções.
- Check Point CloudGuard: uma solução de runtime baseada em extensão que oferece segurança durante todo o ciclo de vida para aplicações com tecnologia sem servidor.
- Datadog: fornece visibilidade abrangente em tempo real das aplicações com tecnologia sem servidor por meio do uso de métricas, rastreamentos e logs.
- Dynatrace: fornece visibilidade de rastreamentos e métricas e utiliza IA para detecção automatizada de erros e análise de causa raiz em toda a pilha de aplicações.
- Elastic: fornece monitoramento de desempenho de aplicativos (APM) para identificar e resolver problemas de causa raiz usando rastreamentos, métricas e logs.
- Epsagon: escuta eventos de invocação, armazena rastreamentos e envia-os em paralelo às execuções de funções do Lambda.
- Fastly: protege as funções do Lambda contra atividades suspeitas, como ataques no estilo de injeção, invasão de contas por meio de inserção de credenciais, bots maliciosos e abuso de API.
- HashiCorp Vault: gerencia segredos e disponibiliza-os para os desenvolvedores usarem em código de funções, sem tornar as funções habilitadas para reconhecimento de cofre.
- Honeycomb: ferramenta de observabilidade para depurar a pilha de aplicações.
- Lumigo: perfila invocações de funções do Lambda e coleta métricas para solucionar problemas em ambientes de tecnologia sem servidor e de microsserviços.
- New Relic: é executado junto com as funções do Lambda, coletando, aperfeiçoando e transportando automaticamente telemetria para a plataforma de observabilidade unificada New Relic.
- Sedai: uma plataforma autônoma de gerenciamento de nuvem, baseada em IA/ML, que oferece otimização contínua para equipes de operações de nuvem a fim de maximizar a economia de custos, o desempenho e a disponibilidade da nuvem em grande escala.
- Sentry: faça o diagnóstico, corrija e otimize a performance das funções do Lambda.
- Site24x7: obtenha observabilidade em tempo real dos ambientes do Lambda

- Splunk: coleta métricas de alta resolução e baixa latência para monitoramento eficiente e eficaz das funções do Lambda.
- Sumo Logic: fornece visibilidade da integridade e performance das aplicações com tecnologia sem servidor.
- Thundra: fornece relatórios de telemetria assíncronos, como rastreamentos, métricas e logs.
- Salt Security: simplifica a governança de posturas da API e a segurança da API para funções do Lambda por meio de configuração e compatibilidade automatizadas para runtimes diversos.

Extensões gerenciadas pela AWS

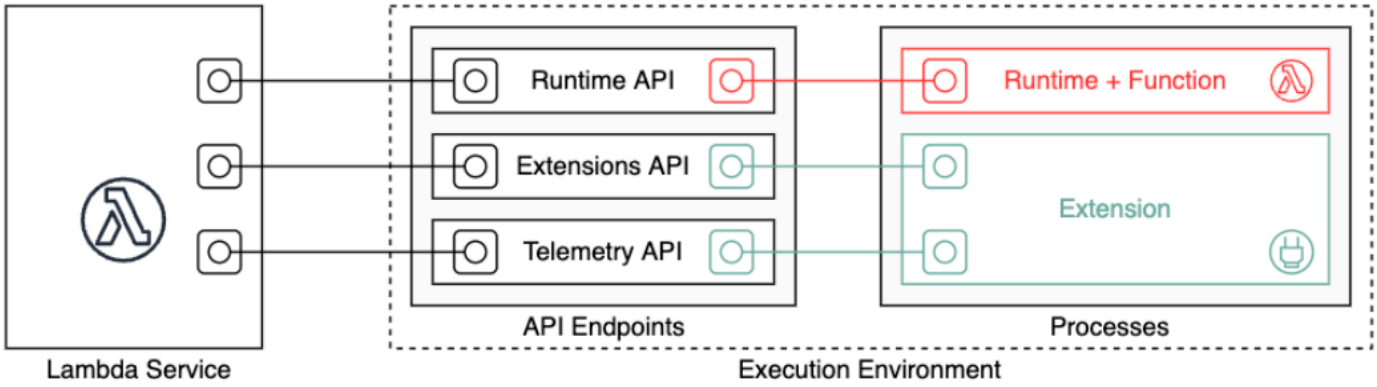
A AWS fornece suas próprias extensões gerenciadas, incluindo:

- AWS AppConfig: use sinalizadores de recursos e dados dinâmicos para atualizar suas funções Lambda. Utilize também essa extensão para atualizar outras configurações dinâmicas, como controle de utilização e ajuste de operações.
- Amazon CodeGuru Profiler: melhora a performance da aplicação e reduz os custos identificando a linha de código mais cara da aplicação e fornecendo recomendações para melhorar o código.
- CloudWatch Lambda Insights: monitore, solucione problemas e otimize a performance de suas funções Lambda por painéis automatizados.
- AWS Distro para OpenTelemetry (ADOT): possibilita que as funções enviem dados de rastreamento para os serviços de monitoramento da AWS, como o AWS X-Ray, e para os destinos que oferecem suporte ao OpenTelemetry, como o Honeycomb e o Lightstep.
- AWS Parameters and Secrets: permite que os clientes recuperem, com segurança, parâmetros da AWS Systems Manager Parameter Store e segredos do AWS Secrets Manager.

Para ver outros exemplos de extensões e projetos de demonstração, consulte as extensões do AWS Lambda.

Usar a API de extensões do Lambda para criar extensões

Autores de funções do Lambda usam extensões para integrar o Lambda às suas ferramentas preferidas para monitoramento, observabilidade, segurança e governança. Os autores de funções podem usar extensões da AWS, de parceiros da AWS e de projetos de código aberto. Para obter mais informações, consulte Introducing AWS Lambda Extensions no AWS Compute Blog. Esta seção descreve como usar a API de extensões do Lambda, o ciclo de vida do ambiente de execução do Lambda e a referência de API de extensões do Lambda.



Como um autor de extensões, você pode usar as API de extensões do Lambda para integrar profundamente no ambiente de execução do Lambda. Sua extensão pode inscrever-se para eventos de ciclo de vida do ambiente de execução e função. Em resposta a esses eventos, você pode iniciar novos processos, executar lógica e controlar e participar de todas as fases do ciclo de vida do Lambda: inicialização, invocação e desligamento. Além disso, você pode usar os API Runtime Logs para receber um stream de logs.

Uma extensão externa é executada como um processo independente no ambiente de execução e continua a ser executada após a invocação da função ser totalmente processada. Como as extensões são executadas como processos, elas podem ser escritas em uma linguagem diferente da função. Recomendamos que você implemente as extensões usando uma linguagem compilada. Nesse caso, a extensão é um binário autônomo compatível com os tempos de execução compatíveis. Todos os Runtimes do Lambda são compatíveis com extensões. Se você usar uma linguagem não compilada, inclua um runtime compatível na extensão.

O Lambda também oferece suporte a extensões internas. Uma extensão interna é executada como um thread separado no processo de runtime. O runtime inicia e interrompe a extensão interna. Uma maneira alternativa de se integrar com o ambiente do Lambda é usar variáveis de ambiente com

linguagem específica e scripts wrapper. Você pode usar isso para configurar o ambiente do runtime e modificar o comportamento de startup do processo do runtime.

Você pode adicionar extensões a uma função de duas maneiras. Para uma função implantada como um arquivo.zip, você implanta a extensão como uma camada. Para uma função definida como uma imagem de contêiner, você adiciona as extensões à imagem do contêiner.

> **Note**
>
> Por exemplo, extensões e scripts de wrapper, consulte AWS Lambda Extensões no repositório AWS Samples GitHub.

Tópicos

- Ciclo de vida do ambiente de execução do Lambda
- Referência de API de extensões

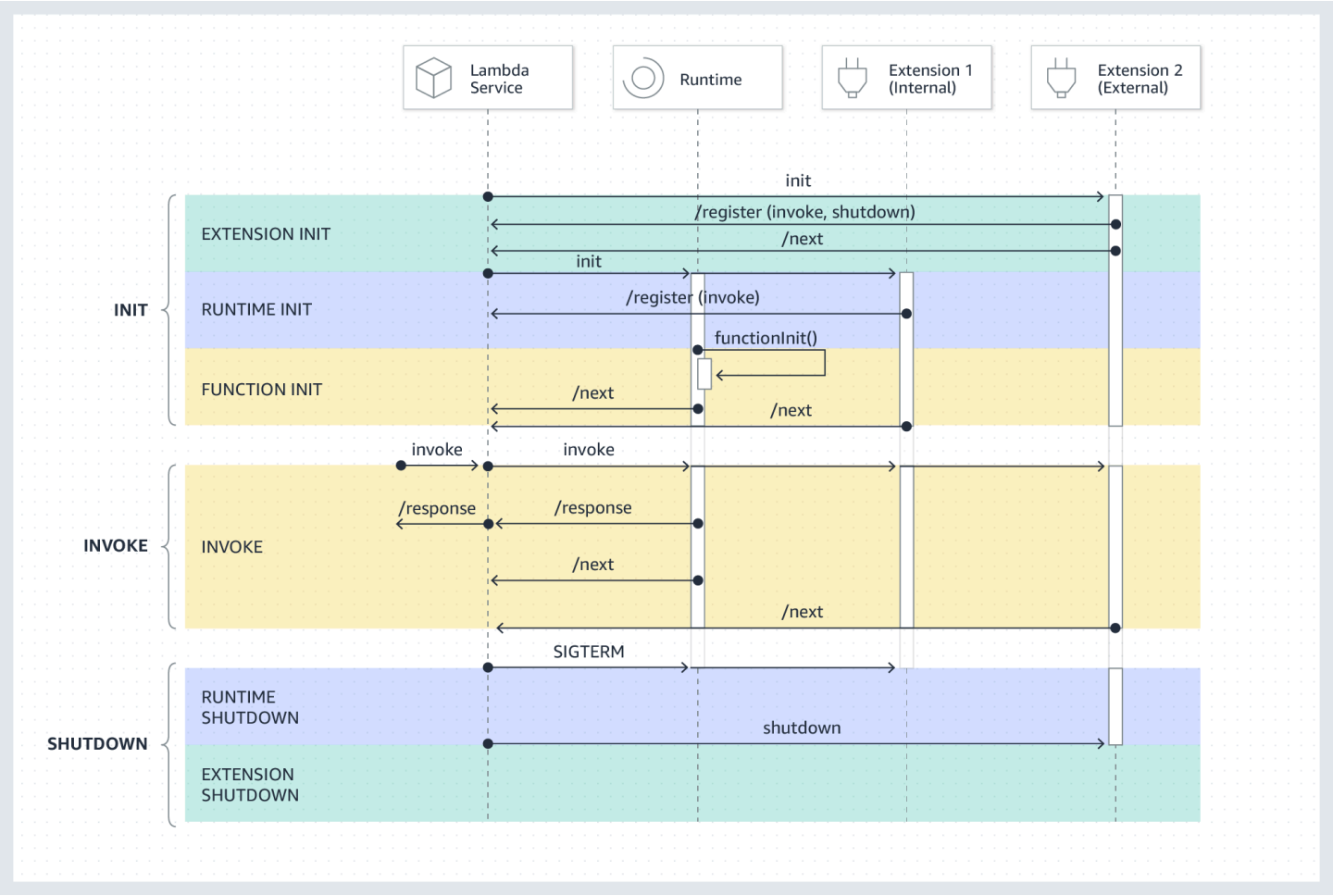
Ciclo de vida do ambiente de execução do Lambda

O ciclo de vida do ambiente de execução inclui as seguintes fases:

- **`Init`**: nessa fase, o Lambda cria ou descongela um ambiente de execução com os recursos configurados, faz download do código da função e de todas as camadas, inicializa todas as extensões e o runtime, em seguida executa o código de inicialização da função (o código fora do handler principal). O`Init`ocorre durante a primeira invocação, ou antes de invocações de função se você tiver habilitadoSimultaneidade provisionada.

  O`Init`é dividida em três subfases:`Extension init`,`Runtime init`, e`Function init`. Essas subfases garantem que todas as extensões e o runtime concluam suas tarefas de configuração antes que o código da função seja executado.
- **`Invoke`**: Nesta fase, o Lambda chama o manipulador de função. Depois que a função é executada até a conclusão, o Lambda se prepara para manipular outra invocação de função.
- **`Shutdown`**: Esta fase é acionada se a função do Lambda não receber quaisquer invocações por um período de tempo. Na fase `Shutdown`, o Lambda encerra o runtime, alerta as extensões para permitir que elas parem de forma limpa e, em seguida, remove o ambiente. O Lambda envia um`Shutdown`para cada extensão, que informa a extensão que o ambiente está prestes a ser encerrado.

Cada fase começa com um evento do serviço do Lambda para o runtime e para todas as extensões registradas. O runtime e cada conclusão do sinal de extensão enviando uma Solicitação de API `Next`. O Lambda congela o ambiente de execução quando cada processo é concluído e não há eventos pendentes.



Tópicos

- Fase de inicialização
- Fase de invocação
- Fase de desligamento
- Permissões e configuração
- Tratamento de falhas
- Solução de problemas de extensões

Fase de inicialização

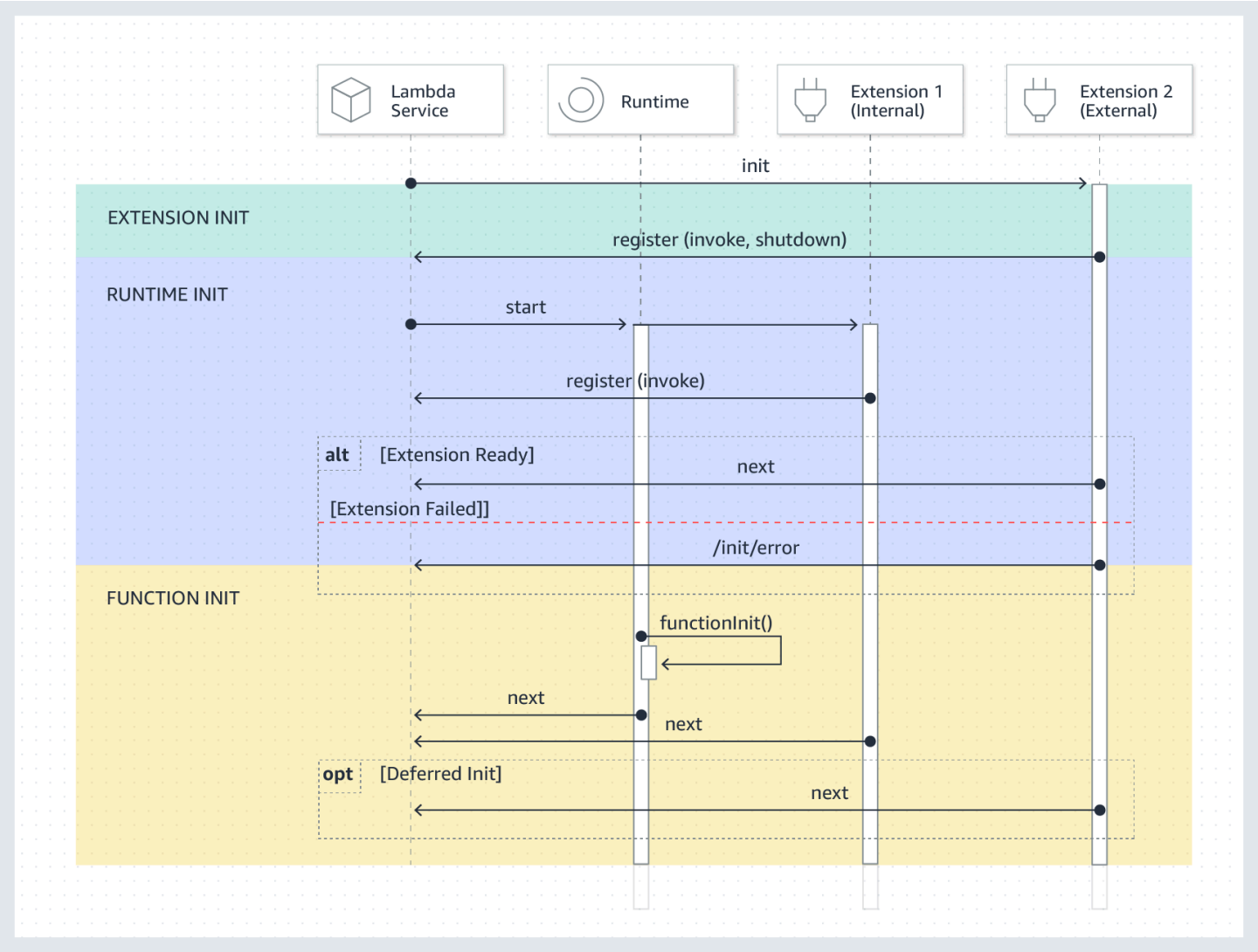
Durante o`Extension init`, cada extensão precisa se registrar com o Lambda para receber eventos. O Lambda usa o nome completo do arquivo da extensão para validar que a extensão concluiu a sequência de bootstrap. Portanto, cada chamada de API `Register` deve incluir o cabeçalho `Lambda-Extension-Name` com o nome de arquivo completo da extensão.

É possível registrar até 10 extensões para uma função. Esse limite é imposto pela chamada de API `Register`.

Depois que cada extensão é registrada, o Lambda inicia a fase `Runtime init`. O processo de runtime chama `functionInit` para iniciar a fase `Function init`.

A fase `Init` é concluída após o runtime e cada extensão registrada indica a conclusão enviando uma solicitação de API `Next`.

> **Note**
>
> As extensões podem concluir sua inicialização em qualquer momento da fase `Init`.



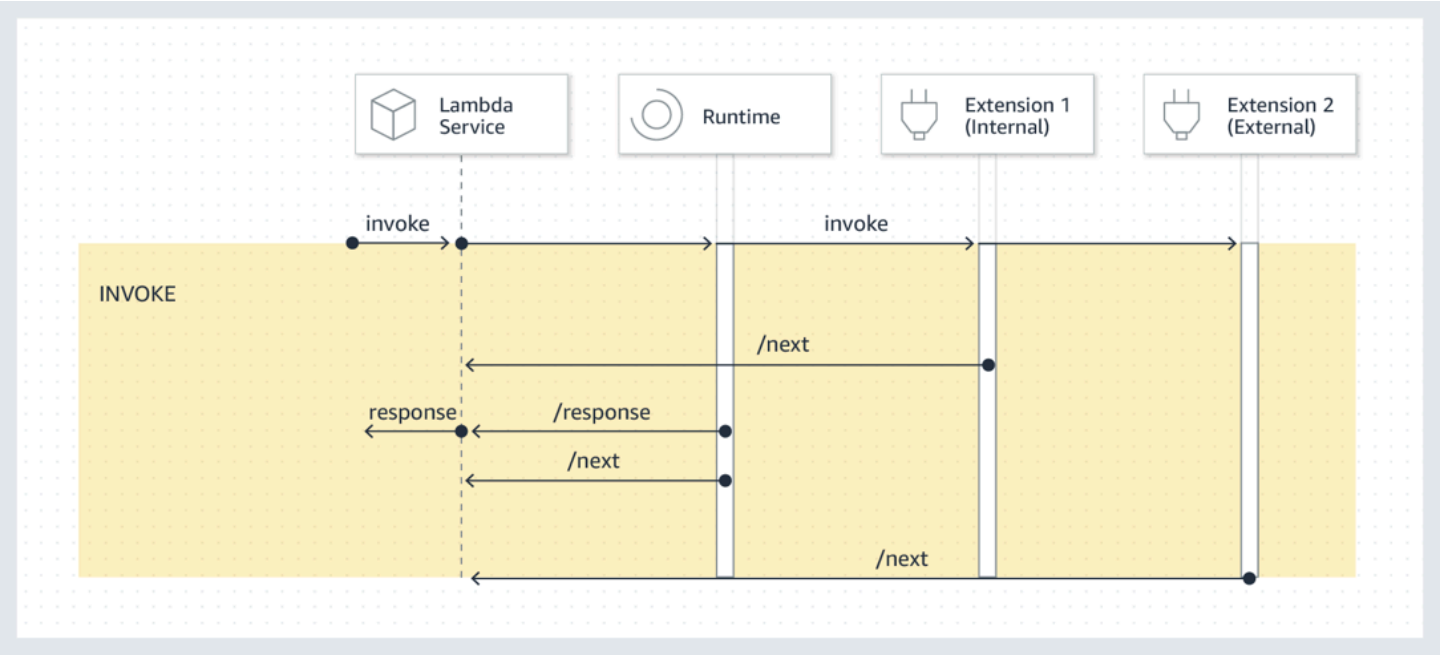
Fase de invocação

Quando uma função do Lambda é invocada em resposta a uma solicitação de API `Next`, o envia um evento `Invoke` para o runtime e para cada extensão registrada para o evento `Invoke`.

Durante a invocação, as extensões externas são executadas em paralelo com a função. Elas também continuam em execução após a conclusão da função. Isso permite que você capture informações para diagnóstico ou para enviar logs, métricas e rastreamentos para um local de sua escolha.

Depois de receber a resposta da função do runtime, o Lambda retorna a resposta ao cliente, mesmo que as extensões ainda estejam em execução.

A fase `Invoke` termina após o runtime e todas as extensões sinalizam que elas foram concluídas enviando uma solicitação de API `Next`.



Carga útil do evento: o evento enviado para o runtime (e a função do Lambda) leva toda a solicitação, os cabeçalhos (como `RequestId`) e carga útil. O evento enviado para cada extensão contém metadados que descrevem o conteúdo do evento. Este evento de ciclo de vida inclui o tipo do evento, o tempo em que a função timas-out (`deadlineMs`) `requestId`, o nome de recurso da Amazon (ARN) da função chamada e cabeçalhos de rastreamento.

As extensões que desejam acessar o corpo do evento de função podem usar um SDK no runtime que se comunica com a extensão. Os desenvolvedores de funções usam o SDK no runtime para enviar a carga útil para a extensão quando a função é invocada.

Veja um exemplo de carga útil:

```
{
    "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"
```

```
    }
}
```

Limite de duração: a definição do tempo limite da função limita a duração de toda a fase `Invoke`. Por exemplo, se você definir o tempo limite da função como 360 segundos, a função e todas as extensões precisam ser concluídas em até 360 segundos. Observe que não há fase de pós-invocação independente. A duração é o tempo total até a conclusão do seu runtime e das invocações de todas as extensões. Esse valor não é calculado até que a função e todas as extensões tenham terminado a execução.

Impacto na performance e sobrecarga de extensão: as extensões podem afetar a performance da função. Como autor da extensão, você tem controle sobre o impacto de performance da extensão. Por exemplo, se a extensão executa operações com uso intenso de computação, a duração da função aumenta, pois a extensão e o código da função compartilham os mesmos recursos de CPU. Além disso, se a extensão executar operações extensas após a conclusão da invocação da função, a duração da função aumentará porque a fase `Invoke` continuará até que todas as extensões sinalizem que foram concluídas.

> **Note**
>
> O Lambda aloca a potência da CPU em proporção à configuração de memória da função. Você pode ver maior duração de execução e inicialização em configurações de memória mais baixas porque os processos de função e extensão estão competindo pelos mesmos recursos da CPU. Para reduzir a duração da execução e inicialização, tente aumentar a configuração de memória.

Para ajudar a identificar o impacto na performance apresentado pelas extensões na fase `Invoke`, o Lambda gera a métrica `PostRuntimeExtensionsDuration`. Essa métrica mede o tempo cumulativo gasto entre a solicitação de API `Next` do runtime e a última solicitação de API `Next` da extensão. Para medir o aumento da memória usada, use a métrica `MaxMemoryUsed`. Para obter mais informações sobre métricas de funções, consulte Trabalhar com métricas de funções Lambda.

Os desenvolvedores de funções podem executar diferentes versões de suas funções lado a lado para entender o impacto de uma extensão específica. Recomendamos que os autores de extensão publiquem o consumo esperado de recursos para facilitar aos desenvolvedores de funções a escolha de uma extensão adequada.

Fase de desligamento

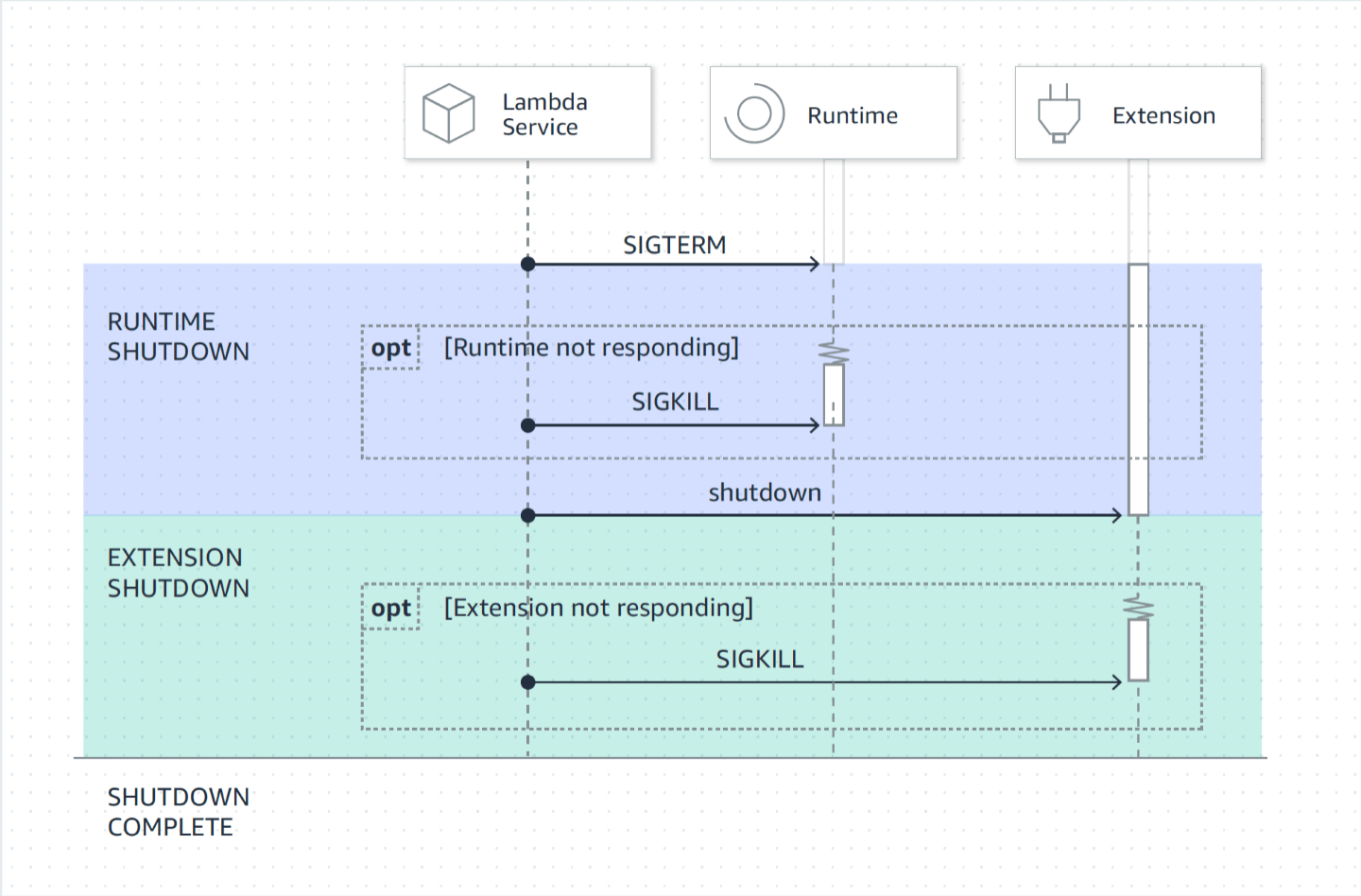
Quando o Lambda estiver prestes a encerrar o runtime, ele enviará um `Shutdown` a cada extensão externa registrada. As extensões podem usar esse tempo para tarefas de limpeza finais. O evento `Shutdown` é enviado em resposta a uma solicitação de API `Next`.

Limite de duração: a duração máxima da fase `Shutdown` depende da configuração das extensões registradas:

- 0 ms: uma função sem extensões registradas
- 500 ms: uma função com uma extensão interna registrada
- 2.000 ms: uma função com uma ou mais extensões externas registradas

Para uma função com extensões externas, o Lambda reserva até 300 ms (500 ms para um runtime com uma extensão interna) para que o processo de runtime execute um desligamento normal. Lambda aloca o restante do limite de 2.000 ms para extensões externas para desligar.

Se o runtime ou uma extensão não responder ao evento `Shutdown` dentro do limite, o Lambda encerrará o processo usando um sinal `SIGKILL`.



Carga útil do evento: o evento `Shutdown` contém o motivo do desligamento e o tempo restante em milissegundos.

O `shutdownReason` inclui os seguintes valores:

- SPINDOWN: desligamento normal
- TIMEOUT: limite da duração do tempo expirado
- FAILURE: condição de erro, como um evento `out-of-memory`

```
{
  "eventType": "SHUTDOWN",
  "shutdownReason": "reason for shutdown",
  "deadlineMs": "the time and date that the function times out in Unix time
 milliseconds"
}
```

Permissões e configuração

As extensões são executadas no mesmo ambiente de execução que a função do Lambda. As extensões também compartilham recursos com a função, como CPU, memória e armazenamento em disco `/tmp`. Além disso, as extensões usam a mesma função do AWS Identity and Access Management (IAM) e o mesmo contexto de segurança que a função.

Permissões de acesso ao sistema de arquivos e à rede: as extensões são executadas no mesmo sistema de arquivos e namespace de nome de rede que o runtime da função. Isso significa que as extensões precisam ser compatíveis com o sistema operacional associado. Se uma extensão exigir regras adicionais de saída do tráfego de rede, você deverá aplicar essas regras à configuração da função.

> **Note**
>
> Como o diretório de código de função é somente leitura, as extensões não podem modificar o código da função.

Variáveis de ambiente: as extensões podem acessar as variáveis de ambiente da função, exceto as seguintes variáveis específicas do processo de runtime:

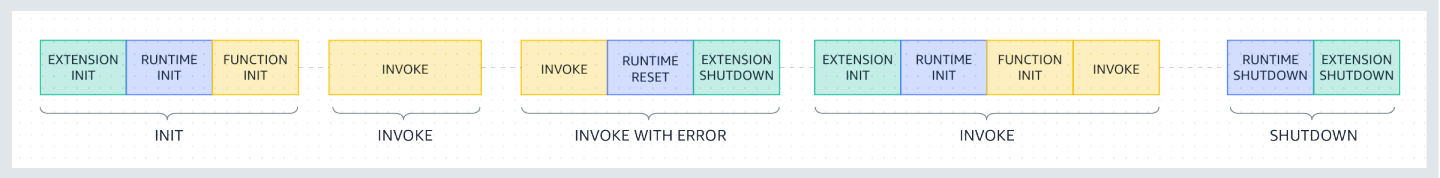
- `AWS_EXECUTION_ENV`
- `AWS_LAMBDA_LOG_GROUP_NAME`
- `AWS_LAMBDA_LOG_STREAM_NAME`
- `AWS_XRAY_CONTEXT_MISSING`
- `AWS_XRAY_DAEMON_ADDRESS`
- `LAMBDA_RUNTIME_DIR`
- `LAMBDA_TASK_ROOT`
- `_AWS_XRAY_DAEMON_ADDRESS`
- `_AWS_XRAY_DAEMON_PORT`
- `_HANDLER`

Tratamento de falhas

Falhas de inicialização: se uma extensão falhar, o Lambda reiniciará o ambiente de execução para impor um comportamento consistente e incentivar falhas rapidamente para extensões. Além disso, para alguns clientes, as extensões devem atender às necessidades críticas, como registro em log, segurança, governança e coleta de telemetria.

Invocar falhas (como falta de memória, tempo limite da função): como as extensões compartilham recursos com o runtime, elas são afetadas pelo esgotamento da memória. Quando o runtime falha, todas as extensões e o próprio runtime participam da fase `Shutdown`. Além disso, o runtime é reiniciado automaticamente como parte do chamado atual ou por meio de um mecanismo de reinicialização adiado.

Se houver uma falha (como um erro de tempo limite ou de runtime de uma função) durante `Invoke`, o serviço Lambda executará uma redefinição. A redefinição se comporta como um evento `Shutdown`. Primeiro, o Lambda encerra o runtime, depois, envia um evento `Shutdown` para cada extensão externa registrada. O evento inclui o motivo do desligamento. Se esse ambiente for usado para uma nova invocação, a extensão e o runtime serão reinicializados como parte da próxima invocação.



Para obter uma explicação mais detalhada do diagrama anterior, consulte Falhas durante a fase de invocação.

Logs de extensões: o Lambda envia a saída de log de extensões para o CloudWatch Logs. O Lambda também gera um evento de log adicional para cada extensão duranteInit. O evento de log registra o nome e a preferência de registro (evento, configuração) em caso de sucesso ou o motivo da falha em caso de falha.

Solução de problemas de extensões

- Se uma solicitação `Register` falhar, o cabeçalho `Lambda-Extension-Name` na chamada de API `Register` deverá conter o nome completo do arquivo da extensão.
- Se a solicitação `Register` falhar para uma extensão interna, a solicitação não deverá estar registrada para o evento `Shutdown`.

Referência de API de extensões

A especificação OpenAPI para a versão da API de extensões 2020-01-01 está disponível aqui: extensions-api.zip

É possível recuperar o valor do endpoint da API da variável de ambiente `AWS_LAMBDA_RUNTIME_API`. Para enviar uma solicitação `Register`, use o prefixo `2020-01-01/` antes de cada caminho da API. Por exemplo:

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
```

Métodos de API

- Inscreva-se
- Próximo
- Erro de inicialização
- Erro de saída

Inscreva-se

Durante`Extension init`, todas as extensões precisam se registrar no Lambda para receber eventos. O Lambda usa o nome completo do arquivo da extensão para validar que a extensão concluiu a sequência de bootstrap. Portanto, cada chamada de API `Register` deve incluir o cabeçalho `Lambda-Extension-Name` com o nome de arquivo completo da extensão.

As extensões internas são iniciadas e interrompidas pelo processo de runtime, portanto, elas não têm permissão para se registrar para o evento `Shutdown`.

Caminho: `/extension/register`

Método – POST

Cabeçalhos de solicitação

- `Lambda-Extension-Name`: o nome completo do arquivo da extensão. Obrigatório: sim. Tipo: string
- `Lambda-Extension-Accept-Feature`: use isso para especificar os recursos opcionais de extensões durante o registro. Obrigatório: não Tipo: strings separada por vírgula. Recursos disponíveis para serem especificados usando essa configuração:

- `accountId`: se especificado, a resposta de registro da extensão conterá o ID da conta associada à função do Lambda para a qual você está registrando a extensão.

Parâmetros do corpo da solicitação

- `events`: matriz dos eventos para registrar. Obrigatório: não Tipo: matriz de strings. Strings válidas: `INVOKE`, `SHUTDOWN`.

Cabeçalhos de resposta

- `Lambda-Extension-Identifier`: identificador de agente exclusivo gerado (string UUID) que é obrigatório para todas as solicitações subsequentes.

Códigos de resposta

- 200: o corpo de resposta contém o nome da função, a versão da função e o nome do manipulador.
- 400: solicitação inválida
- 403: proibido
- 500: erro de contêiner. Estado não recuperável. A extensão deve sair imediatamente.

Example Exemplo de corpo da solicitação

```
{
    'events': [ 'INVOKE', 'SHUTDOWN']
}
```

Example Exemplo de corpo da resposta

```
{
    "functionName": "helloWorld",
    "functionVersion": "$LATEST",
    "handler": "lambda_function.lambda_handler"
}
```

Example Exemplo de corpo de resposta com recurso opcional accountId

```
{
```

```
    "functionName": "helloWorld",
    "functionVersion": "$LATEST",
    "handler": "lambda_function.lambda_handler",
    "accountId": "123456789012"
}
```

Próximo

As extensões enviam uma solicitação de API `Next` para receber o próximo evento, que pode ser um evento `Invoke` ou um evento `Shutdown`. O corpo da resposta contém a carga útil, que é um documento JSON com dados do evento.

A extensão envia uma solicitação de API `Next` para sinalizar que está pronta para receber novos eventos. Essa é uma chamada de bloqueio.

Não defina um tempo limite na chamada GET, pois a extensão pode ser suspensa por um período até que haja um evento a ser retornado.

Caminho: `/extension/event/next`

Método – GET

Cabeçalhos de solicitação

- `Lambda-Extension-Identifier`: identificador exclusivo para extensão (string UUID). Obrigatório: sim. Tipo: string UUID.

Cabeçalhos de resposta

- `Lambda-Extension-Event-Identifier`: identificador exclusivo para o evento (string UUID).

Códigos de resposta

- 200: a resposta contém informações sobre o próximo evento (`EventInvoke` ou `EventShutdown`).
- 403: proibido
- 500: erro de contêiner. Estado não recuperável. A extensão deve sair imediatamente.

Erro de inicialização

A extensão usa esse método para relatar um erro de inicialização para o Lambda. Chame-a quando a extensão falhar ao inicializar após ter sido registrada. Depois que o Lambda recebe o erro, não há mais chamadas de API bem-sucedidas. A extensão deve sair depois de receber a resposta do Lambda.

Caminho: `/extension/init/error`

Método – POST

Cabeçalhos de solicitação

- `Lambda-Extension-Identifier`: identificador exclusivo para extensão. Obrigatório: sim. Tipo: string UUID.
- `Lambda-Extension-Function-Error-Type`: tipo de erro encontrado pela extensão. Obrigatório: sim. Este cabeçalho consiste em um valor de string. Lambda aceita qualquer string, mas recomendamos o formato <category.reason>. Por exemplo:
  - Extension.NoSuchHandler
  - Extension.APIKeyNotFound
  - Extension.ConfigInvalid
  - Extension.UnknownReason

Parâmetros do corpo da solicitação

- `ErrorRequest`: informações adicionais sobre o erro. Obrigatório: não

Este campo é um objeto JSON com a seguinte estrutura:

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

Observe que o Lambda aceita qualquer valor para `errorType`.

O exemplo a seguir mostra uma mensagem de erro de função do Lambda na qual a função não pôde analisar os dados do evento fornecidos na chamada.

Example Erro de função

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Códigos de resposta

- 202: aceito
- 400: solicitação inválida
- 403: proibido
- 500: erro de contêiner. Estado não recuperável. A extensão deve sair imediatamente.

Erro de saída

A extensão usa esse método para relatar um erro ao Lambda antes de sair. Chame-a quando encontrar uma falha inesperada. Depois que o Lambda recebe o erro, não há mais chamadas de API bem-sucedidas. A extensão deve sair depois de receber a resposta do Lambda.

Caminho: `/extension/exit/error`

Método – POST

Cabeçalhos de solicitação

- `Lambda-Extension-Identifier`: identificador exclusivo para extensão. Obrigatório: sim. Tipo: string UUID.
- `Lambda-Extension-Function-Error-Type`: tipo de erro encontrado pela extensão. Obrigatório: sim. Este cabeçalho consiste em um valor de string. Lambda aceita qualquer string, mas recomendamos o formato <category.reason>. Por exemplo:
  - Extension.NoSuchHandler
  - Extension.APIKeyNotFound
  - Extension.ConfigInvalid
  - Extension.UnknownReason

Parâmetros do corpo da solicitação

- `ErrorRequest`: informações adicionais sobre o erro. Obrigatório: não

Este campo é um objeto JSON com a seguinte estrutura:

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

Observe que o Lambda aceita qualquer valor para `errorType`.

O exemplo a seguir mostra uma mensagem de erro de função do Lambda na qual a função não pôde analisar os dados do evento fornecidos na chamada.

Example Erro de função

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Códigos de resposta

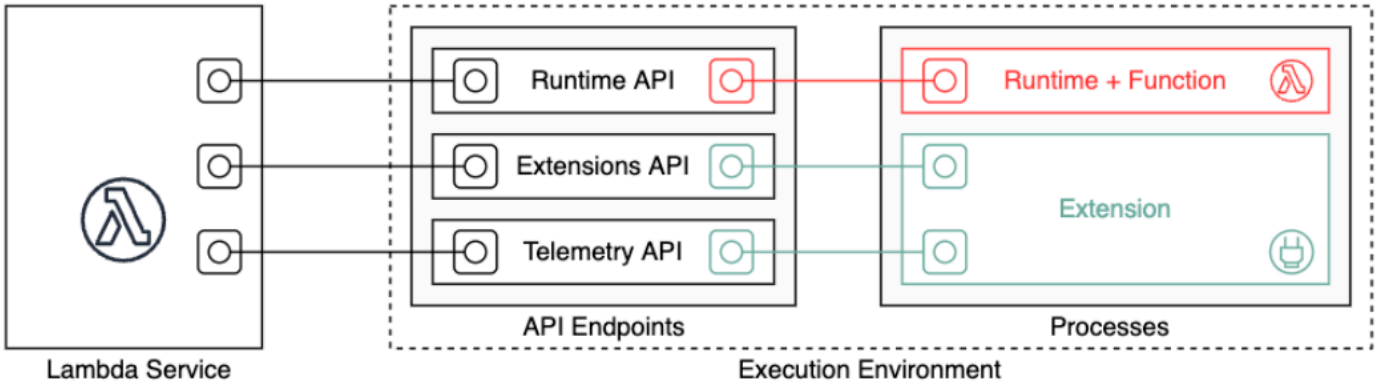
- 202: aceito
- 400: solicitação inválida
- 403: proibido
- 500: erro de contêiner. Estado não recuperável. A extensão deve sair imediatamente.

API de Telemetria do Lambda

A API de Telemetria permite que as extensões recebam dados de telemetria diretamente do Lambda. Durante a inicialização e a invocação da função, o Lambda captura automaticamente a telemetria, incluindo logs, métricas de plataforma e rastreamentos de plataforma. A API de telemetria permite que as extensões acessem esses dados de telemetria diretamente no Lambda e quase em tempo real.

No ambiente de execução do Lambda, você pode inscrever as extensões do Lambda em fluxos de telemetria. Após a inscrição, o Lambda transmite automaticamente todos os dados de telemetria para as extensões. Em seguida, é possível processar, filtrar e entregar esses dados para o destino de preferência, como um bucket do Amazon Simple Storage Service (Amazon S3) ou um provedor externo de ferramentas de observabilidade.

O diagrama a seguir mostra como a API de extensões e a API de telemetria conectam extensões ao Lambda no ambiente de execução. Além disso, a API de runtime conecta o seu runtime e a função ao Lambda.



> **Important**
>
> A API de telemetria do Lambda substitui a API de logs do Lambda. Embora a API de logs permaneça totalmente funcional, recomendamos usar apenas a API de telemetria daqui para frente. É possível inscrever sua extensão em um fluxo de telemetria usando a API de telemetria ou a API de logs. Após se inscrever usando uma dessas APIs, qualquer tentativa de se inscrever usando a outra API retornará um erro.

As extensões podem usar a API de telemetria para se inscrever em três fluxos de telemetria diferentes:

- Telemetria de plataforma: logs, métricas e rastreamentos que descrevem eventos e erros relacionados ao ciclo de vida de runtime do ambiente de execução, ao ciclo de vida de extensão e às invocações de função.
- Logs de função: logs personalizados gerados pelo código da função do Lambda.
- Logs de extensão: logs personalizados gerados pelo código de extensão do Lambda.

> **Note**
>
> O Lambda envia registros CloudWatch, métricas e rastreamentos para o X-Ray (se você tiver ativado o rastreamento), mesmo que uma extensão assine fluxos de telemetria.

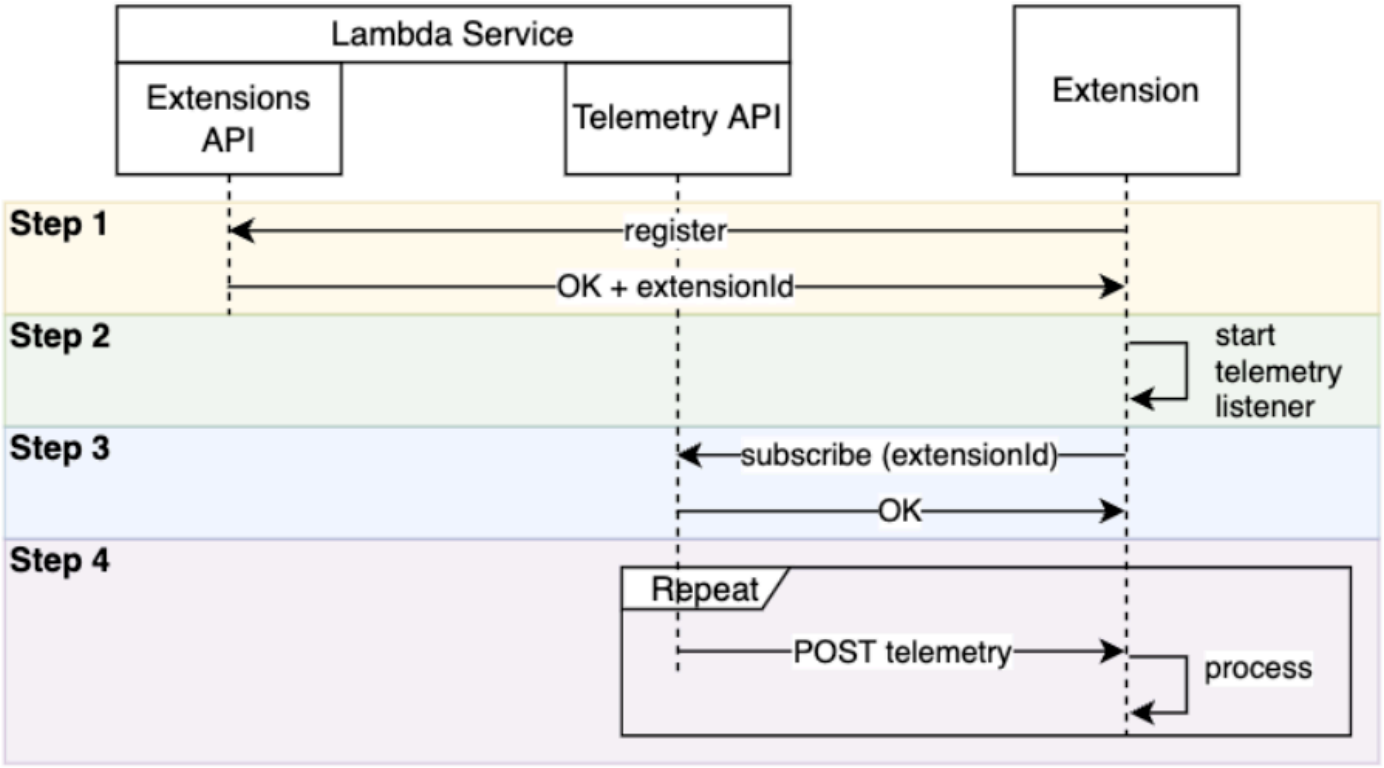
Seções

Criação de extensões usando a API de telemetria

As extensões do Lambda são executadas como processos independentes no ambiente de execução. As extensões podem continuar sendo executadas após a conclusão da invocação da função. Como as extensões são processos separados, é possível escrevê-las em uma linguagem diferente do

código da função. Recomendamos a gravação de extensões usando uma linguagem compilada, como Golang ou Rust. Dessa forma, a extensão corresponde a um binário independente que pode ser compatível com qualquer runtime com suporte.

O diagrama a seguir ilustra um processo de quatro etapas para a criação de uma extensão que recebe e processa dados de telemetria usando a API de telemetria.



Veja a seguir cada etapa com mais detalhes:

1. Registre sua extensão usando a the section called “API de extensões”. Isso fornece um `Lambda-Extension-Identifier`, que você precisará para as etapas a seguir. Para obter mais informações sobre como registrar sua extensão, consulte the section called “Como registrar sua extensão”.
2. Crie um receptor de telemetria. Pode ser um servidor HTTP ou TCP comum. O Lambda usa o URI do receptor de telemetria para enviar dados de telemetria para a extensão. Para ter mais informações, consulte the section called “Como criar um receptor de telemetria”.
3. Use a API de inscrição na API de telemetria para inscrever a extensão nos fluxos de telemetria desejados. Você precisará do URI do receptor de telemetria para esta etapa. Para ter mais informações, consulte the section called “Como enviar uma solicitação de assinatura para a API de telemetria”.

4. Obtenha os dados de telemetria do Lambda por meio do receptor de telemetria. É possível fazer qualquer processamento personalizado desses dados, como enviar os dados para o Amazon S3 ou para um serviço de observabilidade externo.

> **Note**
>
> O ambiente de execução de uma função do Lambda pode ser iniciado e interrompido diversas vezes como parte de seu ciclo de vida. Em geral, o código de extensão é executado durante as invocações de função e também por até dois segundos durante a fase de desligamento. Recomendamos processar a telemetria em lotes à medida que ela chegar ao receptor. Em seguida, use os eventos de ciclo de vida `Invoke` e `Shutdown` para enviar cada lote aos destinos desejados.

Como registrar sua extensão

Antes de se inscrever em dados de telemetria, você deve registrar a extensão do Lambda. O registro ocorre durante a fase de inicialização da extensão. O exemplo a seguir mostra uma solicitação HTTP para registrar uma extensão.

```
POST http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
 Lambda-Extension-Name: lambda_extension_name
{
    'events': [ 'INVOKE', 'SHUTDOWN']
}
```

Se o pedido for bem-sucedido, o assinante receberá uma resposta de sucesso HTTP 200. O cabeçalho de resposta contém o `Lambda-Extension-Identifier`. O corpo de resposta contém outras propriedades da função.

```
HTTP/1.1 200 OK
Lambda-Extension-Identifier: a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
{
    "functionName": "lambda_function",
    "functionVersion": "$LATEST",
    "handler": "lambda_handler",
    "accountId": "123456789012"
}
```

Para obter mais informações, consulte the section called “Referência de API de extensões”.

Como criar um receptor de telemetria

Sua extensão do Lambda deve ter um receptor que processe as solicitações recebidas da API de telemetria. O código a seguir mostra um exemplo de implementação de receptor de telemetria em 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
}
```

Como especificar um protocolo de destino

Ao se inscrever para receber telemetria usando a API de telemetria, é possível especificar um protocolo de destino além do URI de destino:

```
{
    "destination": {
        "protocol": "HTTP",
        "URI": "http://sandbox.localdomain:8080"
    }
}
```

O Lambda aceita dois protocolos para o recebimento de telemetria:

- HTTP (recomendado): o Lambda fornece telemetria para um endpoint HTTP local (`http://sandbox.localdomain:${PORT}/${PATH}`) como uma matriz de registros no formato JSON. O parâmetro `$PATH` é opcional. O Lambda oferece suporte somente para HTTP, não para HTTPS. O Lambda fornece telemetria por meio de solicitações POST.
- TCP: o Lambda fornece telemetria para uma porta TCP no formato Newline delimited JSON (NDJSON).

> **Note**
>
> Recomendamos fortemente o uso de HTTP em vez de TCP. Com o TCP, a plataforma do Lambda não reconhece que a telemetria é entregue à camada da aplicação. Portanto, se sua extensão falhar, você poderá perder a telemetria. O HTTP não tem essa limitação.

Antes de se inscrever para receber telemetria, estabeleça o receptor HTTP ou a porta TCP local. Durante a configuração, observe o seguinte:

- O Lambda envia telemetria somente para destinos que estão dentro do ambiente de execução.
- O Lambda tenta novamente enviar telemetria (com recuo) na ausência de um receptor ou se a solicitação POST encontrar algum erro. Se o receptor de telemetria apresentar falhas, ele continuará a receber telemetria depois que o Lambda reiniciar o ambiente de execução.
- Lambda reserva porto 9001. Não há outras restrições ou recomendações de número de porta.

Como configurar o uso de memória e o armazenamento em buffer

O uso de memória em um ambiente de execução cresce linearmente com o número de assinantes. As assinaturas consomem recursos de memória porque cada assinatura abre um novo buffer de memória para armazenar dados de telemetria. O uso da memória buffer contribui para o consumo geral de memória no ambiente de execução.

Ao se inscrever para receber telemetria usando a API de telemetria, você tem a opção de armazenar em buffer os dados de telemetria e entregá-los aos assinantes em lotes. Para otimizar o uso da memória, é possível especificar uma configuração de armazenamento em buffer:

```
{
    "buffering": {
        "maxBytes": 256*1024,
        "maxItems": 1000,
        "timeoutMs": 100
    }
}
```

Definições de configuração de armazenamento em buffer

Parâmetro	Descrição	Padrões e limites
`maxBytes`	O volume máximo de telemetria (em bytes) para armazenar em buffer na memória.	Padrão: 262.144 Mínimo: 262.144 Máximo: 1.048.576

Parâmetro	Descrição	Padrões e limites
`maxItems`	O número máximo de eventos para armazenar em buffer na memória.	Padrão: 10.000 Mínimo: 1.000 Máximo: 10.000.
`timeoutMs`	O tempo máximo (em milissegundos) para armazenar em buffer um lote.	Padrão: 1.000 Mínimo: 25 Máximo: 30.000

Ao configurar o buffer, lembre-se dos seguintes pontos:

- Se algum dos fluxos de entrada estiver fechado, o Lambda liberará os logs. Por exemplo, isso pode acontecer se o runtime apresentar falhas.
- Cada assinante pode personalizar a configuração de buffer durante a solicitação de assinatura.
- Ao determinar o tamanho do buffer para ler os dados, preveja o recebimento de cargas úteis tão grandes quanto `2 * maxBytes + metadataBytes`, em que `maxBytes` é um componente da configuração de buffer. Para avaliar a quantidade de `metadataBytes` a ser considerada, analise os metadados a seguir. O Lambda anexa metadados semelhantes a este para cada registro:

```
{
   "time": "2022-08-20T12:31:32.123Z",
   "type": "function",
   "record": "Hello World"
}
```

- Se o assinante não puder processar a telemetria de entrada com rapidez suficiente ou se o código da sua função gerar um volume de log muito alto, o Lambda poderá descartar registros para manter a utilização da memória limitada. Quando isso ocorre, o Lambda envia um evento `platform.logsDropped`.

Como enviar uma solicitação de assinatura para a API de telemetria

As extensões do Lambda podem se inscrever para receber dados de telemetria enviando uma solicitação de assinatura para a API de telemetria. A solicitação de assinatura deve conter informações sobre os tipos de eventos que você deseja que a extensão assine. Além disso, a solicitação pode conter informações do destino de entrega e uma configuração de armazenamento em buffer.

Antes de enviar uma solicitação de assinatura, você deve ter um ID de extensão (`Lambda-Extension-Identifier`). Ao registrar sua extensão na API de extensões, você obtém um ID de extensão a partir da resposta da API.

A assinatura ocorre durante a fase de inicialização da extensão. O exemplo a seguir mostra uma solicitação HTTP para assinar todos os três fluxos de telemetria: telemetria de plataforma, logs de função e logs de extensão.

```
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 a solicitação tiver êxito, o assinante receberá uma resposta de sucesso HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Mensagens da API de telemetria de entrada

Depois de se inscrever usando a API de telemetria, uma extensão começa automaticamente a receber telemetria do Lambda por meio de solicitações POST. Cada corpo de solicitação POST contém uma matriz de objetos `Event`. Cada `Event` tem o seguinte esquema:

```
{
   time: String,
   type: String,
   record: Object
}
```

- A propriedade `time` define quando a plataforma do Lambda gerou o evento. Isso é diferente de quando o evento realmente ocorreu. O valor da string `time` é um carimbo de data/hora no formato ISO 8601.
- A propriedade `type` define o tipo de evento. A tabela a seguir descreve todos os valores possíveis.
- A propriedade `record` define um objeto JSON que contém os dados de telemetria. O esquema desse objeto JSON depende do arquivo `type`.

A tabela a seguir resume todos os tipos de objetos `Event` e apresenta links para a referência de esquema `Event` da API de telemetria para cada tipo de evento.

Tipos de mensagem da API de telemetria

Categoria	Tipo de evento	Descrição	Esquema de registro de evento
Evento de plataforma	`platform.initStart`	A inicialização da função foi iniciada.	Esquema the section called “`platform.initStart`”
Evento de plataforma	`platform.initRuntimeDone`	A inicialização da função foi concluída.	Esquema the section called “`platform.initRuntimeDone`”

Categoria	Tipo de evento	Descrição	Esquema de registro de evento
Evento de plataforma	`platform.initReport`	Um relatório sobre a inicialização da função.	Esquema the section called “`platform.initReport`”
Evento de plataforma	`platform.start`	A invocação da função foi iniciada.	Esquema the section called “`platform.start`”
Evento de plataforma	`platform.runtimeDone`	O runtime concluiu o processamento de um evento com sucesso ou com falha.	Esquema the section called “`platform.runtimeDone`”
Evento de plataforma	`platform.report`	Um relatório sobre a invocação de função.	Esquema the section called “`platform.report`”
Evento de plataforma	`platform.restoreStart`	A restauração do runtime foi iniciada.	Esquema the section called “`platform.restoreStart`”
Evento de plataforma	`platform.restoreRuntimeDone`	A restauração do runtime foi concluída.	Esquema the section called “`platform.restoreRuntimeDone`”
Evento de plataforma	`platform.restoreReport`	Relatório de restauração do runtime.	Esquema the section called “`platform.restoreReport`”
Evento de plataforma	`platform.telemetrySubscription`	A extensão foi inscrita na API de telemetria.	Esquema the section called “`platform.telemetrySubscription`”

Categoria	Tipo de evento	Descrição	Esquema de registro de evento
Evento de plataforma	`platform.logsDropped`	O Lambda descartou entradas de log.	Esquema the section called “`platform.logsDropped`”
Registros de função	`function`	Uma linha de log do código da função.	Esquema the section called “`function`”
Logs de extensões	`extension`	Uma linha de log do código de extensão.	Esquema the section called “`extension`”

Referência da API de Telemetria do Lambda

Use o endpoint da API de Telemetria do Lambda para assinar extensões para fluxos de telemetria. É possível recuperar o endpoint da API de telemetria da variável de ambiente `AWS_LAMBDA_RUNTIME_API`. Para enviar uma solicitação de API, vincule a versão da API (`2022-07-01/`) e `telemetry/`. Por exemplo: .

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Para obter a definição da Especificação OpenAPI (OAS) da versão das respostas de assinatura `2022-12-13`, consulte os itens a seguir:

- HTTP — telemetry-api-http-schema.zip
- TCP — .zip telemetry-api-tcp-schema

Operações de API

- Assinar

Assinar

Para assinar um fluxo de telemetria, uma extensão do Lambda pode enviar uma solicitação para a API de assinatura.

- Caminho: `/telemetry`
- Method (Método): `PUT`
- Cabeçalhos
  - `Content-Type`: `application/json`
- Parâmetros do corpo da solicitação
  - schemaVersion
    - Obrigatório: sim
    - Tipo: String
    - Valores válidos: `"2022-12-13"` ou `"2022-07-01"`
  - destination (destino): as configurações que definem o destino do evento de telemetria e o protocolo para a entrega do evento.
    - Exigido: Sim

- Tipo: Objeto

```
{
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080"
}
```

  - protocol (protocolo): o protocolo usado pelo Lambda para enviar dados de telemetria.
    - Obrigatório: sim
    - Tipo: String
    - Valores válidos: `"HTTP"|"TCP"`
  - URI: o URI para o qual os dados de telemetria serão enviados.
    - Obrigatório: sim
    - Tipo: String
  - Para ter mais informações, consulte the section called “Como especificar um protocolo de destino”.
- types (tipos): os tipos de telemetria que você deseja que a extensão assine.
  - Exigido: Sim
  - Tipo: matriz de strings
  - Valores válidos: `"platform"|"function"|"extension"`
- buffering (armazenamento em buffer): as configurações definidas para o armazenamento em buffer de eventos.
  - Obrigatório: Não
  - Tipo: Objeto

```
{
   "buffering": {
        "maxItems": 1000,
        "maxBytes": 256*1024,
        "timeoutMs": 100
   }
}
```

  - maxItems: o número máximo de eventos a serem colocados em buffer na memória.
    - Obrigatório: Não
    - Tipo: inteiro

    - Padrão: 1.000
    - Mínimo: 1.000
    - Máximo: 10.000.
- maxBytes: o volume máximo de telemetria (em bytes) para armazenar em buffer na memória.
    - Obrigatório: Não
    - Tipo: inteiro
    - Padrão: 262.144
    - Mínimo: 262.144
    - Máximo: 1.048.576
- timeoutMs: o tempo máximo (em milissegundos) para colocar um lote em buffer.
    - Obrigatório: Não
    - Tipo: inteiro
    - Padrão: 1.000
    - Mínimo: 25
    - Máximo: 30.000
- Para ter mais informações, consulte the section called “Como configurar o uso de memória e o armazenamento em buffer”.

Exemplo de solicitação para a API de assinatura

```
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 a solicitação de assinatura obtiver êxito, a extensão receberá uma resposta de sucesso HTTP 200:

```
HTTP/1.1 200 OK
"OK"
```

Se a solicitação de assinatura falhar, a extensão receberá uma resposta de erro. Por exemplo: .

```
HTTP/1.1 400 OK
{
    "errorType": "ValidationError",
    "errorMessage": "URI port is not provided; types should not be empty"
}
```

Veja a seguir alguns códigos de resposta adicionais que a extensão pode receber:

- 200: solicitação concluída com êxito
- 202: solicitação aceita. Resposta à solicitação de assinatura no ambiente de teste local.
- 400: solicitação inválida.
- 500: erro do serviço

Referência de esquema para **`Event`** da API de Telemetria do Lambda

Use o endpoint da API de Telemetria do Lambda para assinar extensões para fluxos de telemetria. É possível recuperar o endpoint da API de telemetria da variável de ambiente `AWS_LAMBDA_RUNTIME_API`. Para enviar uma solicitação de API, vincule a versão da API (`2022-07-01/`) e `telemetry/`. Por exemplo:

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Para obter a definição da Especificação OpenAPI (OAS) da versão das respostas de assinatura `2022-12-13`, consulte os itens a seguir:

- HTTP: telemetry-api-http-schema.zip
- TCP: telemetry-api-tcp-schema.zip

A tabela a seguir é um resumo de todos os tipos de objetos `Event` aos quais a API de telemetria oferece suporte.

Tipos de mensagem da API de telemetria

Categoria	Tipo de evento	Descrição	Esquema de registro de evento
Evento de plataforma	`platform.initStart`	A inicialização da função foi iniciada.	Esquema the section called “`platform.initStart`”
Evento de plataforma	`platform.initRuntimeDone`	A inicialização da função foi concluída.	Esquema the section called “`platform.initRuntimeDone`”
Evento de plataforma	`platform.initReport`	Um relatório sobre a inicialização da função.	Esquema the section called “`platform.initReport`”

Categoria	Tipo de evento	Descrição	Esquema de registro de evento
Evento de plataforma	`platform.start`	A invocação da função foi iniciada.	Esquema the section called “`platform.start`”
Evento de plataforma	`platform.runtimeDone`	O runtime concluiu o processamento de um evento com sucesso ou com falha.	Esquema the section called “`platform.runtimeDone`”
Evento de plataforma	`platform.report`	Um relatório sobre a invocação de função.	Esquema the section called “`platform.report`”
Evento de plataforma	`platform.restoreStart`	A restauração do runtime foi iniciada.	Esquema the section called “`platform.restoreStart`”
Evento de plataforma	`platform.restoreRuntimeDone`	A restauração do runtime foi concluída.	Esquema the section called “`platform.restoreRuntimeDone`”
Evento de plataforma	`platform.restoreReport`	Relatório de restauração do runtime.	Esquema the section called “`platform.restoreReport`”
Evento de plataforma	`platform.telemetrySubscription`	A extensão foi inscrita na API de telemetria.	Esquema the section called “`platform.telemetrySubscription`”
Evento de plataforma	`platform.logsDropped`	O Lambda descartou entradas de log.	Esquema the section called “`platform.logsDropped`”

Categoria	Tipo de evento	Descrição	Esquema de registro de evento
Registros de função	`function`	Uma linha de log do código da função.	Esquema the section called “`function`”
Logs de extensões	`extension`	Uma linha de log do código de extensão.	Esquema the section called “`extension` ”

Sumário

- Span
- Status
- TraceContext
- TracingType

Tipos de objeto **`Event`** da API de telemetria

Esta seção detalha os tipos de objetos `Event` aos quais a API de Telemetria do Lambda oferece suporte. Nas descrições dos eventos, um ponto de interrogação (`?`) indica que o atributo pode não estar presente no objeto.

`platform.initStart`

Um evento `platform.initStart` indica que a fase de inicialização da função foi iniciada. Um objeto `Event` `platform.initStart` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = platform.initStart
- record: PlatformInitStart
```

O objeto `PlatformInitStart` tem os seguintes atributos:

- functionName: `String`
- functionVersion: `String`
- initializationType: objeto `the section called “InitType”`
- instanceId?: `String`
- instanceMaxMemory?: `Integer`
- phase (fase): objeto `the section called “InitPhase”`
- runtimeVersion?: `String`
- runtimeVersionArn?: `String`

Veja a seguir um exemplo de `Event` do 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`

Um evento `platform.initRuntimeDone` indica que a fase de inicialização da função foi concluída. Um objeto `Event` `platform.initRuntimeDone` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = platform.initRuntimeDone
- record: PlatformInitRuntimeDone
```

O objeto `PlatformInitRuntimeDone` tem os seguintes atributos:

- initializationType: objeto the section called “InitType”
- phase (fase): objeto the section called “InitPhase”
- status: objeto the section called “Status”
- spans? : lista de objetos the section called “Span”

Veja a seguir um exemplo de `Event` do 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`

Um evento `platform.initReport` contém um relatório geral da fase de inicialização da função. Um objeto `Event` `platform.initReport` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = platform.initReport
- record: PlatformInitReport
```

O objeto `PlatformInitReport` tem os seguintes atributos:

- errorType?: string
- initializationType: objeto the section called “InitType”
- phase (fase): objeto the section called “InitPhase”
- metrics (métricas): objeto the section called “InitReportMetrics”
- spans? : lista de objetos the section called “Span”
- status: objeto the section called “Status”

Veja a seguir um exemplo de `Event` do 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`

Um evento `platform.start` indica que a fase de invocação da função foi iniciada. Um objeto `Event` `platform.start` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = platform.start
- record: PlatformStart
```

O objeto `PlatformStart` tem os seguintes atributos:

- requestId: `String`
- version?: `String`
- tracing?: the section called “TraceContext”

Veja a seguir um exemplo de `Event` do 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`

Um evento `platform.runtimeDone` indica que a fase de invocação da função foi concluída. Um objeto `Event` `platform.runtimeDone` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = platform.runtimeDone
- record: PlatformRuntimeDone
```

O objeto `PlatformRuntimeDone` tem os seguintes atributos:

- errorType?: `String`
- metrics?: objeto the section called “RuntimeDoneMetrics”
- requestId: `String`
- status: objeto the section called “Status”
- spans? : lista de objetos the section called “Span”
- tracing?: objeto the section called “TraceContext”

Veja a seguir um exemplo de `Event` do 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`

Um evento `platform.report` contém um relatório geral da fase de inicialização da função. Um objeto `Event` `platform.report` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = platform.report
- record: PlatformReport
```

O objeto `PlatformReport` tem os seguintes atributos:

- metrics (métricas): objeto the section called “ReportMetrics”
- requestId: `String`
- spans? : lista de objetos the section called “Span”
- status: objeto the section called “Status”
- tracing?: objeto the section called “TraceContext”

Veja a seguir um exemplo de `Event` do 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`

Um evento `platform.restoreStart` indica que um evento de restauração do ambiente de função foi iniciado. Em um evento de restauração do ambiente, o Lambda cria o ambiente de um snapshot armazenado em cache, em vez de inicializá-lo do zero. Para ter mais informações, consulte Lambda SnapStart. Um objeto `Event` `platform.restoreStart` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = platform.restoreStart
- record: PlatformRestoreStart
```

O objeto `PlatformRestoreStart` tem os seguintes atributos:

- functionName: `String`
- functionVersion: `String`
- instanceId?: `String`
- instanceMaxMemory?: `String`
- runtimeVersion?: `String`
- runtimeVersionArn?: `String`

Veja a seguir um exemplo de `Event` do 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`

Um evento `platform.restoreRuntimeDone` indica que um evento de restauração do ambiente de função foi concluído. Em um evento de restauração do ambiente, o Lambda cria o ambiente de um snapshot armazenado em cache, em vez de inicializá-lo do zero. Para ter mais informações, consulte Lambda SnapStart. Um objeto `Event` `platform.restoreRuntimeDone` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = platform.restoreRuntimeDone
- record: PlatformRestoreRuntimeDone
```

O objeto `PlatformRestoreRuntimeDone` tem os seguintes atributos:

- errorType?: `String`
- spans? : lista de objetos the section called “Span”
- status: objeto the section called “Status”

Veja a seguir um exemplo de `Event` do 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`

Um evento `platform.restoreReport` contém um relatório geral de um evento de restauração de função. Um objeto `Event` `platform.restoreReport` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = platform.restoreReport
- record: PlatformRestoreReport
```

O objeto `PlatformRestoreReport` tem os seguintes atributos:

- errorType?: string
- metrics?: objeto the section called “RestoreReportMetrics”
- spans? : lista de objetos the section called “Span”
- status: objeto the section called “Status”

Veja a seguir um exemplo de `Event` do 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`

Um evento `extension` contém logs do código de extensão. Um objeto `Event` `extension` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = extension
- record: {}
```

O objeto `PlatformExtension` tem os seguintes atributos:

- events (eventos): lista de `String`
- name (nome): `String`
- state (estado): `String`

Veja a seguir um exemplo de `Event` do 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`

Um evento `platform.telemetrySubscription` contém informações sobre uma assinatura de extensão. Um objeto `Event` `platform.telemetrySubscription` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = platform.telemetrySubscription
- record: PlatformTelemetrySubscription
```

O objeto `PlatformTelemetrySubscription` tem os seguintes atributos:

- name (nome): `String`
- state (estado): `String`
- types (tipos): lista de `String`

Veja a seguir um exemplo de `Event` do 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`

Um evento `platform.logsDropped` contém informações sobre eventos descartados. O Lambda emite o evento `platform.logsDropped` quando uma função gera logs em uma taxa muito alta para o Lambda processá-los. Quando o Lambda não consegue enviar logs para o CloudWatch ou para a extensão inscrita na API Telemetry na velocidade em que a função os produz, ele descarta logs para impedir que a execução da função fique mais lenta. Um objeto `Event` `platform.logsDropped` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = platform.logsDropped
- record: PlatformLogsDropped
```

O objeto `PlatformLogsDropped` tem os seguintes atributos:

- droppedBytes: `Integer`
- droppedRecords: `Integer`
- reason (razão): `String`

Veja a seguir um exemplo de `Event` do 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`

Um evento `function` contém logs do código da função. Um objeto `Event` `function` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = function
- record: {}
```

O formato do campo `record` vai variar dependendo se os logs da função estão em formato de texto simples ou JSON. Para saber mais sobre as opções de configuração de formato de logs, consulte the section called “Configurar logs em formato de texto simples e JSON”.

Veja abaixo um exemplo de `Event` de tipo `function` em que o log está formatado em texto simples:

```
{
    "time": "2022-10-12T00:03:50.000Z",
    "type": "function",
    "record": "[INFO] Hello world, I am a function!"
}
```

Veja abaixo um exemplo de `Event` de tipo `function` em que o log está formatado em 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 a versão do esquema que você está usando for mais antiga que a versão `2022-12-13`, o campo `"record"` será sempre renderizado como uma string, mesmo se o formato do log da função estiver configurado como JSON.

`extension`

Um evento `extension` contém logs do código de extensão. Um objeto `Event` `extension` tem a forma a seguir:

```
Event: Object
- time: String
- type: String = extension
- record: {}
```

O formato do campo `record` vai variar dependendo se os logs da função estão em formato de texto simples ou JSON. Para saber mais sobre as opções de configuração de formato de logs, consulte the section called “Configurar logs em formato de texto simples e JSON”.

Veja abaixo um exemplo de `Event` de tipo `extension` em que o log está formatado em texto simples:

```
{
    "time": "2022-10-12T00:03:50.000Z",
    "type": "extension",
    "record": "[INFO] Hello world, I am an extension!"
}
```

Veja abaixo um exemplo de `Event` de tipo `extension` em que o log está formatado em 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 a versão do esquema que você está usando for mais antiga que a versão `2022-12-13`, o campo `"record"` será sempre renderizado como uma string, mesmo se o formato do log da função estiver configurado como JSON.

Tipos de objetos compartilhados

Esta seção detalha os tipos de objetos compartilhados aos quais a API de Telemetria do Lambda oferece suporte.

`InitPhase`

Uma enumeração da string que descreve a fase em que ocorre a etapa de inicialização. Na maioria dos casos, o Lambda executa o código de inicialização da função durante a fase `init`. No entanto, em alguns casos de erro, o Lambda pode executar novamente o código de inicialização da função durante a fase `invoke`. (Isso é chamado de suppressed init [inicialização suprimida]).

- Tipo: `String`
- Valores válidos: `init|invoke|snap-start`

`InitReportMetrics`

Um objeto que contém as métricas sobre uma fase de inicialização.

- Tipo: `Object`

Um objeto `InitReportMetrics` tem a forma a seguir:

```
InitReportMetrics: Object
- durationMs: Double
```

Veja a seguir um exemplo de um objeto `InitReportMetrics`:

```
{
    "durationMs": 247.88
}
```

`InitType`

Uma enumeração de string que descreve como o Lambda inicializou o ambiente.

- Tipo: `String`
- Valores válidos: `on-demand`|`provisioned-concurrency`

`ReportMetrics`

Um objeto que contém métricas sobre uma fase concluída.

- Tipo: `Object`

Um objeto `ReportMetrics` tem a forma a seguir:

```
ReportMetrics: Object
- billedDurationMs: Integer
- durationMs: Double
- initDurationMs?: Double
- maxMemoryUsedMB: Integer
- memorySizeMB: Integer
- restoreDurationMs?: Double
```

Veja a seguir um exemplo de um objeto `ReportMetrics`:

```
{
    "billedDurationMs": 694,
    "durationMs": 693.92,
    "initDurationMs": 397.68,
    "maxMemoryUsedMB": 84,
```

```
    "memorySizeMB": 128
}
```

`RestoreReportMetrics`

Um objeto que contém métricas sobre uma fase de restauração concluída.

- Tipo: `Object`

Um objeto `RestoreReportMetrics` tem a forma a seguir:

```
RestoreReportMetrics: Object
- durationMs: Double
```

Veja a seguir um exemplo de um objeto `RestoreReportMetrics`:

```
{
    "durationMs": 15.19
}
```

`RuntimeDoneMetrics`

Um objeto que contém métricas sobre uma fase de invocação.

- Tipo: `Object`

Um objeto `RuntimeDoneMetrics` tem a forma a seguir:

```
RuntimeDoneMetrics: Object
- durationMs: Double
- producedBytes?: Integer
```

Veja a seguir um exemplo de um objeto `RuntimeDoneMetrics`:

```
{
    "durationMs": 200.0,
    "producedBytes": 15
}
```

`Span`

Um objeto que contém detalhes sobre um span. Um span representa uma unidade de trabalho ou de operação em um rastreamento. Para obter mais informações sobre os spans, consulte Span na página Tracing API (API de rastreamento) do site OpenTelemetry Docs.

O Lambda é compatível com os seguintes spans para o evento `platform.RuntimeDone`:

- O span `responseLatency` descreve quanto tempo sua função do Lambda demorou para começar a enviar a resposta.
- O span `responseDuration` descreve quanto tempo sua função do Lambda demorou para concluir o envio da resposta completa.
- O span `runtimeOverhead` descreve quanto tempo levou para o runtime do Lambda sinalizar que está pronto para processar a próxima invocação da função. Esse é o tempo que o runtime levou para chamar a próxima API de invocação para obter o próximo evento após retornar a resposta da função.

Veja a seguir um exemplo de um objeto do span `responseLatency`:

```
{
        "name": "responseLatency",
        "start": "2022-08-02T12:01:23.521Z",
        "durationMs": 23.02
      }
```

`Status`

Um objeto que descreve o status de uma fase de inicialização ou invocação. Se o status for `failure` ou`error`, o objeto `Status` também conterá um campo `errorType` com a descrição do erro.

- Tipo: `Object`
- Valores de status válidos: `success`|`failure`|`error`|`timeout`

`TraceContext`

Um objeto que descreve as propriedades de um rastreamento.

- Tipo: `Object`

Um objeto `TraceContext` tem a forma a seguir:

```
TraceContext: Object
- spanId?: String
- type: TracingType enum
- value: String
```

Veja a seguir um exemplo de um objeto `TraceContext`:

```
{
    "spanId": "073a49012f3c312e",
    "type": "X-Amzn-Trace-Id",
    "value":
 "Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
}
```

`TracingType`

Uma enumeração de string que descreve o tipo de rastreamento em um objeto the section called “TraceContext”.

- Tipo: `String`
- Valores válidos: `X-Amzn-Trace-Id`

Conversão de objetos da API de telemetria Lambda em **`Event`** spans OpenTelemetry

O esquema da API de AWS Lambda telemetria é semanticamente compatível com OpenTelemetry (OTel). Isso significa que você pode converter seus `Event` objetos da API de AWS Lambda telemetria em extensões OpenTelemetry (OTel). Ao converter, você não deve mapear um único objeto `Event` para um único span do OTel. Em vez disso, você deve apresentar todos os três eventos relacionados a uma fase do ciclo de vida em um único span do OTel. Por exemplo, os eventos `start`, `runtimeDone` e `runtimeReport` representam uma única invocação de função. Apresente todos esses três eventos como um único span do OTel.

É possível converter seus eventos usando “Span Events” (Eventos de span) ou “Child Spans” (aninhado) (Spans secundários). As tabelas nesta página descrevem os mapeamentos entre as propriedades do esquema da API de telemetria e as propriedades do span do OTel para ambas as abordagens. Para obter mais informações sobre oTel Spans, consulte Span na página da API de rastreamento do site do OpenTelemetry Docs.

Seções

Mapeamento para spans do OTel com “Span Events” (Eventos de span)

Nas tabelas a seguir, e representa o evento proveniente da fonte de telemetria.

Como mapear os eventos **`*Start`**

OpenTelemetry	Esquema da API de Telemetria do Lambda
`Span.Name`	Sua extensão gera esse valor com base no campo `type`.
`Span.StartTime`	Use `e.time`.
`Span.EndTime`	N/A, porque o evento ainda não foi concluído.
`Span.Kind`	Definido como `Server`.
`Span.Status`	Definido como `Unset`.

OpenTelemetry	Esquema da API de Telemetria do Lambda
`Span.TraceId`	Analise o cabeçalho do AWS X-Ray encontrado em `e.tracing.value` e, em seguida, use o valor `TraceId`.
`Span.ParentId`	Analise o cabeçalho do X-Ray encontrado em `e.tracing.value` e, em seguida, use o valor `Parent`.
`Span.SpanId`	Use `e.tracing.spanId` , se disponível. Caso contrário, gere um novo `SpanId`.
`Span.SpanContext.TraceState`	N/A para um contexto de rastreamento do X-Ray.
`Span.SpanContext.TraceFlags`	Analise o cabeçalho do X-Ray encontrado em `e.tracing.value` e, em seguida, use o valor `Sampled`.
`Span.Attributes`	Sua extensão pode adicionar quaisquer valores personalizados aqui.

Como mapear os eventos **`*RuntimeDone`**

OpenTelemetry	Esquema da API de Telemetria do Lambda
`Span.Name`	Sua extensão gera o valor com base no campo `type`.
`Span.StartTime`	Use `e.time` a partir do evento `*Start` correspondente. Como alternativa, use `e.time - e.metrics.durationMs` .
`Span.EndTime`	N/A, porque o evento ainda não foi concluído.
`Span.Kind`	Definido como `Server`.

OpenTelemetry	Esquema da API de Telemetria do Lambda
`Span.Status`	Se `e.status` não for igual a `success`, defina como `Error`. Caso contrário, defina como `Ok`.
`Span.Events[]`	Use `e.spans[]` .
`Span.Events[i].Name`	Use `e.spans[i].name` .
`Span.Events[i].Time`	Use `e.spans[i].start` .
`Span.TraceId`	Analise o cabeçalho do AWS X-Ray encontrado em `e.tracing.value` e, em seguida, use o valor `TraceId`.
`Span.ParentId`	Analise o cabeçalho do X-Ray encontrado em `e.tracing.value` e, em seguida, use o valor `Parent`.
`Span.SpanId`	Use o mesmo `SpanId` do evento `*Start`. Se não estiver disponível, use `e.tracing.spanId` ou gere um novo `SpanId`.
`Span.SpanContext.TraceState`	N/A para um contexto de rastreamento do X-Ray.
`Span.SpanContext.TraceFlags`	Analise o cabeçalho do X-Ray encontrado em `e.tracing.value` e, em seguida, use o valor `Sampled`.
`Span.Attributes`	Sua extensão pode adicionar quaisquer valores personalizados aqui.

Como mapear os eventos **`*Report`**

OpenTelemetry	Esquema da API de Telemetria do Lambda
`Span.Name`	Sua extensão gera o valor com base no campo `type`.
`Span.StartTime`	Use `e.time` a partir do evento `*Start` correspondente. Como alternativa, use `e.time - e.metrics.durationMs` .
`Span.EndTime`	Use `e.time`.
`Span.Kind`	Definido como `Server`.
`Span.Status`	Use o mesmo valor do evento `*RuntimeDone` .
`Span.TraceId`	Analise o cabeçalho do AWS X-Ray encontrado em `e.tracing.value` e, em seguida, use o valor `TraceId`.
`Span.ParentId`	Analise o cabeçalho do X-Ray encontrado em `e.tracing.value` e, em seguida, use o valor `Parent`.
`Span.SpanId`	Use o mesmo `SpanId` do evento `*Start`. Se não estiver disponível, use `e.tracing.spanId` ou gere um novo `SpanId`.
`Span.SpanContext.TraceState`	N/A para um contexto de rastreamento do X-Ray.
`Span.SpanContext.TraceFlags`	Analise o cabeçalho do X-Ray encontrado em `e.tracing.value` e, em seguida, use o valor `Sampled`.

OpenTelemetry	Esquema da API de Telemetria do Lambda
`Span.Attributes`	Sua extensão pode adicionar quaisquer valores personalizados aqui.

Mapeamento para spans do OTel com “Child Spans” (Spans secundários)

A tabela a seguir descreve como converter eventos da API de Telemetria do Lambda em spans do OTel com “Child Spans” (aninhado) (Spans secundários) para spans `*RuntimeDone`. Para mapeamentos `*Start` e `*Report`, consulte as tabelas em the section called “Mapeamento para spans do OTel com “Span Events” (Eventos de span)”, pois elas são semelhantes para “Child Spans” (Spans secundários). Nesta tabela, e representa o evento proveniente da fonte de telemetria.

Como mapear os eventos **`*RuntimeDone`**

OpenTelemetry	Esquema da API de Telemetria do Lambda
`Span.Name`	Sua extensão gera o valor com base no campo `type`.
`Span.StartTime`	Use `e.time` a partir do evento `*Start` correspondente. Como alternativa, use `e.time - e.metrics.durationMs` .
`Span.EndTime`	N/A, porque o evento ainda não foi concluído.
`Span.Kind`	Definido como `Server`.
`Span.Status`	Se `e.status` não for igual a `success`, defina como `Error`. Caso contrário, defina como `Ok`.
`Span.TraceId`	Analise o cabeçalho do AWS X-Ray encontrado em `e.tracing.value` e, em seguida, use o valor `TraceId`.

OpenTelemetry	Esquema da API de Telemetria do Lambda
`Span.ParentId`	Analise o cabeçalho do X-Ray encontrado em `e.tracing.value` e, em seguida, use o valor `Parent`.
`Span.SpanId`	Use o mesmo `SpanId` do evento `*Start`. Se não estiver disponível, use `e.tracing.spanId` ou gere um novo `SpanId`.
`Span.SpanContext.TraceState`	N/A para um contexto de rastreamento do X-Ray.
`Span.SpanContext.TraceFlags`	Analise o cabeçalho do X-Ray encontrado em `e.tracing.value` e, em seguida, use o valor `Sampled`.
`Span.Attributes`	Sua extensão pode adicionar quaisquer valores personalizados aqui.
`ChildSpan[i].Name`	Use `e.spans[i].name` .
`ChildSpan[i].StartTime`	Use `e.spans[i].start` .
`ChildSpan[i].EndTime`	Use `e.spans[i].start + e.spans[i].durations` .
`ChildSpan[i].Kind`	Semelhante ao principal `Span.Kind` .
`ChildSpan[i].Status`	Semelhante ao principal `Span.Status` .
`ChildSpan[i].TraceId`	Semelhante ao principal `Span.TraceId` .
`ChildSpan[i].ParentId`	Use o principal `Span.SpanId` .
`ChildSpan[i].SpanId`	Gere um novo `SpanId`.
`ChildSpan[i].SpanContext.TraceState`	N/A para um contexto de rastreamento do X-Ray.

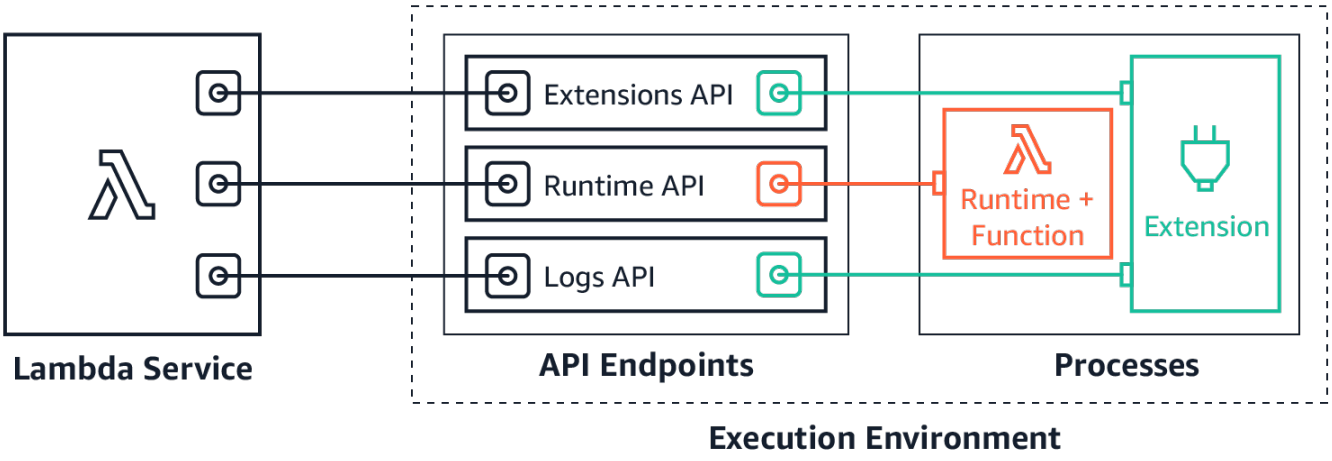
OpenTelemetry	Esquema da API de Telemetria do Lambda
`ChildSpan[i].SpanContext.TraceFlags`	Semelhante ao principal `Span.SpanContext.TraceFlags` .

API Lambda Logs

> **Important**
>
> A API de telemetria do Lambda substitui a API de logs do Lambda. Embora a API de logs permaneça totalmente funcional, recomendamos usar apenas a API de telemetria daqui para frente. É possível inscrever sua extensão em um fluxo de telemetria usando a API de telemetria ou a API de logs. Após se inscrever usando uma dessas APIs, qualquer tentativa de se inscrever usando a outra API retornará um erro.

O Lambda captura automaticamente os registros de tempo de execução e os transmite para a Amazon. CloudWatch Essa transmissão de log contém os logs que seu código de função e extensões geram, e também os gerados pelo Lambda como parte da chamada de função.

Extensões do Lambdapode usar a API RLambda time Logs para assinar fluxos de log diretamente do Lambdaambiente de execução. O Lambda transmite os logs para a extensão e a extensão pode processar, filtrar e enviar os logs para qualquer destino preferido.



A API Logs permite que as extensões se inscrevam em três fluxos de logs diferentes:

- Logs de função que a função do Lambda gera e grava em `stdout` ou `stderr`.
- Registros de extensão que o código de extensão gera.
- Logs da plataforma do Lambda que registram eventos e erros relacionados a invocações e extensões.

> **Note**
>
> O Lambda envia todos os registros para CloudWatch, mesmo quando uma extensão se inscreve em um ou mais dos fluxos de registros.

Tópicos

Assinando para receber registros

Uma extensão do Lambda pode se inscrever para receber registros enviando uma solicitação de assinatura para a API Logs.

Para se inscrever para receber registros, você precisa do identificador de extensão (`Lambda-Extension-Identifier`). Primeiro registre a extensão para receber o identificador de extensão. Em seguida, inscreva-se na API Logs durante a inicialização. Após a conclusão da fase de inicialização, o Lambda não processa solicitações de assinatura.

> **Note**
>
> A assinatura da API de registros é idempotente. As solicitações de assinatura duplicadas não resultam em assinaturas duplicadas.

Uso de memória

O uso da memória aumenta linearmente à medida que o número de assinantes aumenta. As assinaturas consomem recursos de memória porque cada assinatura abre um novo buffer de

memória para armazenar os logs. Para ajudar a otimizar o uso da memória, você pode ajustar a configuração de buffer. O uso da memória buffer conta para o consumo geral de memória no ambiente de execução.

protocolos de destino

Você pode escolher um dos seguintes protocolos para receber os logs:

1. HTTP (recomendado): o Lambda entrega os logs a um endpoint HTTP local (`http://sandbox.localdomain:${PORT}/${PATH}`) como uma matriz de registros no formato JSON. O parâmetro `$PATH` é opcional. Observe que somente HTTP é suportado, não HTTPS. Você pode optar por receber registros através PUT ou POST.
2. TCP: o Lambda entrega logs a uma porta TCP no formato JSON delimitado por nova linha (NDJSON).

Recomendamos usar HTTP em vez de TCP. Com o TCP, a plataforma do Lambda não reconhece que os logs são entregues à camada da aplicação. Portanto, você poderá perder os logs se a extensão falhar. O HTTP não compartilha essa limitação.

Também recomendamos configurar o ouvinte HTTP local ou a porta TCP antes de se inscrever para receber logs. Durante a configuração, observe o seguinte:

- O Lambda envia logs somente para destinos que estão dentro do ambiente de execução.
- O Lambda tenta novamente enviar os logs (com recuo) se não houver listener ou se a solicitação POST ou PUT resultar em erro. Se o assinante do log falhar, ele continuará a receber logs depois que o Lambda reiniciar o ambiente de execução.
- Lambda reserva porto 9001. Não há outras restrições ou recomendações de número de porta.

Configuração de buffer

O Lambda pode armazenar logs de buffer e entregá-los ao assinante. Você pode configurar esse comportamento na solicitação de assinatura especificando os campos opcionais a seguir. Observe que o Lambda usa o valor padrão para qualquer campo que você não especificar.

- timeoutMs: o tempo máximo (em milissegundos) para colocar um lote em buffer. Padrão: 1.000. Mínimo: 25. Máximo: 30.000.
- maxBytes: o tamanho máximo (em bytes) dos logs para colocar em buffer na memória. Padrão: 262.144. Mínimo: 262.144. Máximo: 1.048.576.

- maxItems: o número máximo de eventos a serem colocados em buffer na memória. Padrão: 10.000. Mínimo: 1.000. Máximo: 10.000.

Durante a configuração de buffer, observe os seguintes pontos:

- O Lambda libera os logs se alguma das transmissões de entrada estiver fechada, por exemplo, se o tempo de execução falhar.
- Cada assinante pode especificar uma configuração de buffer diferente durante a solicitação de assinatura.
- Considere o tamanho do buffer que você precisa para ler os dados. Espere receber cargas úteis tão grandes quanto `2*maxBytes+metadata`, onde `maxBytes` é configurado na solicitação de assinatura. Por exemplo, o Lambda adiciona os seguintes bytes de metadados a cada registro:

```
{
"time": "2020-08-20T12:31:32.123Z",
"type": "function",
"record": "Hello World"
}
```

- Se o assinante não puder processar logs de entrada com rapidez suficiente, o Lambda pode descartar logs para manter a utilização da memória limitada. Para indicar o número de registros descartados, o Lambda envia um log `platform.logsDropped`.

Exemplo de assinatura

O exemplo a seguir mostra uma solicitação para se inscrever na plataforma e logs de funções.

```
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 o pedido for bem-sucedido, o assinante receberá uma resposta de sucesso HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Código de exemplo para Logs API

Para obter exemplos de código mostrando como enviar logs para um destino personalizado, consulte Using AWS Lambda extensions to send logs to custom destinations no AWS Compute Blog.

Para exemplos de código em Python e Go que mostram como desenvolver uma extensão básica do Lambda e assinar a API Logs, consulte AWS LambdaExtensões no repositório de amostras. AWS GitHub Para obter mais informações sobre a criação de uma extensão do Lambda, consulte the section called “API de extensões”.

Referência da API Logs

É possível recuperar o endpoint da API Logs da variável de ambiente `AWS_LAMBDA_RUNTIME_API`. Para enviar uma solicitação de API, use o prefixo `2020-08-15/` antes do caminho da API. Por exemplo: .

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs
```

A especificação OpenAPI para a Logs API versão 2020-08-15 está disponível aqui: .zip logs-api-request

Assinar

Para se inscrever em um ou mais transmissões de log disponíveis no ambiente de execução do Lambda, as extensões enviam uma solicitação de assinatura da API.

Caminho: `/logs`

Método – PUT

Body parameters (Parâmetros do corpo)

`destination`: consulte the section called “protocolos de destino”. Obrigatório: sim. Tipo: strings.

`buffering`: consulte the section called “Configuração de buffer”. Obrigatório: não Tipo: strings.

`types`: uma matriz dos tipos de logs a serem recebidos. Obrigatório: sim. Tipo: matriz de strings. Valores válidos: “plataforma”, “função”, “extensão”.

`schemaVersion`: obrigatório: não Valor padrão: “2020-08-15”. Defina como “2021-03-18” para que a extensão receba `platform.runtimeDone` mensagens.

Parâmetros de resposta

As especificações OpenAPI para as respostas de assinatura, versão 2020-08-15, estão disponíveis para HTTP e TCP:

- HTTP: logs-api-http-response.zip
- TCP: .zip logs-api-tcp-response

Códigos de resposta

- 200: solicitação concluída com êxito
- 202: solicitação aceita. Resposta a uma solicitação de assinatura durante testes locais.
- 4XX: solicitação inválida
- 500: erro do serviço

Se o pedido for bem-sucedido, o assinante receberá uma resposta de sucesso HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Se a solicitação falhar, o assinante receberá uma resposta de erro. Por exemplo:

```
HTTP/1.1 400 OK
{
    "errorType": "Logs.ValidationError",
    "errorMessage": URI port is not provided; types should not be empty"
```

```
}
```

Mensagens de log

A API Logs permite que as extensões se inscrevam em três fluxos de logs diferentes:

- Função: logs que a função do Lambda gera e grava em `stdout` ou `stderr`.
- Extensão: logs gerados pelo código da extensão.
- Plataforma: logs gerados pela plataforma de tempo de execução, que registram eventos e erros relacionados a invocações e extensões.

Tópicos

Registros de função

A função e as extensões internas do Lambda geram logs de funções e os gravam em `stdout` ou `stderr`.

O exemplo a seguir mostra o formato de uma mensagem de log de função. { "time": "2020-08-20T12:31:32.123Z", "type": "function", "record": "ERROR encountered. Rastreamento da pilha:\n\my-function (line 10)\n" }

Logs de extensões

As extensões podem gerar logs de extensões. O formato do log é o mesmo que para um log de funções.

Logs de plataformas

O Lambda gera mensagens de log para eventos de plataforma como `platform.end`, `platform.start` e `platform.fault`.

Opcionalmente, você pode se inscrever na versão 2021-03-18 do esquema da API de logs, que inclui a mensagem de log `platform.runtimeDone`.

Exemplo de mensagens de log da plataforma

O exemplo a seguir mostra o início da plataforma e os logs finais da plataforma. Esses logs indicam a hora de início da invocação e o horário de término para a invocação que o requestID especifica.

```
{
    "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"}
}
```

A plataforma. initRuntimeDonea mensagem de registro mostra o status da `Runtime init` subfase, que faz parte da fase do ciclo de vida inicial. Quando o `Runtime init` ocorre com êxito, o tempo de execução envia uma solicitação de API de tempo de execução `/next` (para os tipos `provisioned-concurrency` e de inicialização `on-demand`) ou `restore/next` (para o tipo de inicialização `snap-start`). O exemplo a seguir mostra uma plataforma bem-sucedida. initRuntimeDonemensagem de log para o tipo de `snap-start` inicialização.

```
{
  "time":"2022-07-17T18:41:57.083Z",
  "type":"platform.initRuntimeDone",
  "record":{
      "initializationType":"snap-start",
      "status":"success"
  }
}
```

A mensagem de log Platform.initReport mostra quanto tempo durou a fase `Init` e por quantos milissegundos você foi cobrado durante essa fase. Quando o tipo de inicialização é `provisioned-concurrency`, o Lambda envia essa mensagem durante a invocação. Quando o tipo de inicialização é `snap-start`, o Lambda envia essa mensagem após restaurar o snapshot. O exemplo a seguir mostra uma mensagem de log platform.initReport para o tipo de inicialização `snap-start`.

```
{
```

```
  "time":"2022-07-17T18:41:57.083Z",
  "type":"platform.initReport",
  "record":{
      "initializationType":"snap-start",
      "metrics":{
          "durationMs":731.79,
          "billedDurationMs":732
          }
  }
}
```

O log de relatório da plataforma inclui métricas sobre a invocação que o requestID especifica. O campo `initDurationMs` é incluído no log somente se a invocação incluir uma partida a frio. Se o rastreamento do AWS X-Ray estiver ativo, o log incluirá metadados do X-Ray. O exemplo a seguir mostra um log de relatório de plataforma para uma invocação que incluiu uma partida a frio.

```
{
    "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
        }
    }
}
```

O log da plataforma captura erros de ambiente de execução ou tempo de execução. O exemplo a seguir mostra uma mensagem de log de falha da plataforma.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.fault",
    "record": "RequestId: d783b35e-a91d-4251-af17-035953428a2c Process exited before
 completing request"
}
```

O Lambda gera um log de extensão de plataforma quando uma extensão se registra com a API Extensions. O exemplo a seguir mostra uma mensagem de extensão da plataforma.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.extension",
    "record": {"name": "Foo.bar",
        "state": "Ready",
        "events": ["INVOKE", "SHUTDOWN"]
     }
}
```

O Lambda gera um log de assinatura de logs de plataforma quando uma extensão se inscreve na API Logs. O exemplo a seguir mostra uma mensagem de assinatura de logs.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.logsSubscription",
    "record": {"name": "Foo.bar",
        "state": "Subscribed",
        "types": ["function", "platform"],
    }
}
```

O Lambda gera um log descartado de logs de plataforma quando uma extensão não é capaz de processar o número de logs que está recebendo. O exemplo a seguir mostra uma mensagem de log `platform.logsDropped`.

```
{
    "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
    }
}
```

A mensagem de log platform.restoRestoRestart mostra o horário em que a fase `Restore` começou (somente para o tipo de inicialização `snap-start`). Exemplo:

```
{
  "time":"2022-07-17T18:43:44.782Z",
```

```
  "type":"platform.restoreStart",
  "record":{}
}
```

A mensagem de log platform.restoreReport mostra quanto tempo durou a fase `Restore` e por quantos milissegundos você foi cobrado durante essa fase (somente para o tipo de inicialização`snap-start`). Exemplo:

```
{
  "time":"2022-07-17T18:43:45.936Z",
  "type":"platform.restoreReport",
  "record":{
      "metrics":{
          "durationMs":70.87,
          "billedDurationMs":13
      }
  }
}
```

Mensagens **`runtimeDone`** da plataforma

Se você definir a versão do esquema como “2021-03-18” na solicitação de assinatura, o Lambda enviará uma mensagem `platform.runtimeDone` após a conclusão da invocação da função com êxito ou com um erro. A extensão pode usar esta mensagem para interromper toda a coleção de telemetria para esta invocação de função.

A especificação OpenAPI para o tipo de evento do log no esquema versão 2021-03-18 está disponível aqui: schema-2021-03-18.zip

O Lambda gera a mensagem de log `Next` quando o tempo de execução envia uma solicitação de API do tempo de execução `platform.runtimeDone` ou `Error`. O `platform.runtimeDone` log informa os consumidores da API Logs que a invocação de função foi concluída. As extensões podem usar essas informações para decidir quando enviar toda a telemetria coletada durante essa invocação.

Exemplos

O Lambda envia a mensagem `platform.runtimeDone` depois que o tempo de execução envia a solicitação NEXT, quando a invocação da função for concluída. Os exemplos a seguir mostram mensagens para cada um dos valores de status: sucesso, falha e tempo limite.

Example Exemplo de mensagem de sucesso

```
{
    "time": "2021-02-04T20:00:05.123Z",
    "type": "platform.runtimeDone",
    "record": {
       "requestId":"6f7f0961f83442118a7af6fe80b88",
       "status": "success"
    }
}
```

Example Exemplo de mensagem de falha

```
{
   "time": "2021-02-04T20:00:05.123Z",
   "type": "platform.runtimeDone",
   "record": {
      "requestId":"6f7f0961f83442118a7af6fe80b88",
      "status": "failure"
   }
}
```

Example Exemplo de mensagem de tempo limite

```
{
   "time": "2021-02-04T20:00:05.123Z",
   "type": "platform.runtimeDone",
   "record": {
      "requestId":"6f7f0961f83442118a7af6fe80b88",
      "status": "timeout"
  }
}
```

Example Exemplo de plataforma. restoreRuntimeDone mensagem (somente tipo de **`snap-start`** inicialização)

A plataforma. restoreRuntimeDonea mensagem de registro mostra se a `Restore` fase foi bem-sucedida ou não. mO Lambda envia essa mensagem quando o runtime envia uma solicitação de API de runtime `restore/next`. Existem três status possíveis: com êxito, com falha e tempo limite. O exemplo a seguir mostra uma plataforma bem-sucedida. restoreRuntimeDonemensagem de registro.

```
{
  "time":"2022-07-17T18:43:45.936Z",
  "type":"platform.restoreRuntimeDone",
  "record":{
      "status":"success"
  }
}
```

Solução de problemas no Lambda

Os tópicos a seguir fornecem orientações para a solução de erros e problemas que você pode encontrar ao usar a API, o console ou as ferramentas do Lambda. Se encontrar um problema que não esteja listado aqui, você poderá usar o botão Feedback desta página para relatá-lo.

Para obter mais orientações sobre solução de problemas e respostas a perguntas comuns de suporte, acesse a Central de Conhecimento da AWS.

Para obter mais informações sobre depuração e solução de problemas de aplicações do Lambda, consulte Debugging no Serverless Land.

Tópicos

Solucionar problemas de implantação no Lambda

Quando você atualiza a função, o Lambda implanta a alteração executando novas instâncias da função com o código ou as configurações atualizados. Erros de implantação impedem que a nova versão seja usada e podem surgir de problemas com o pacote de implantação, o código, as permissões ou as ferramentas.

Ao implantar atualizações na função diretamente com a API do Lambda ou com um cliente, como a AWS CLI, é possível visualizar os erros do Lambda diretamente na saída. Se você usar serviços como o AWS CloudFormation, o AWS CodeDeploy ou o AWS CodePipeline, procure a resposta do Lambda nos logs ou no fluxo de eventos desses serviços.

Os tópicos a seguir fornecem orientações para a solução de erros e problemas que você pode encontrar ao usar a API, o console ou as ferramentas do Lambda. Se encontrar um problema que não esteja listado aqui, você poderá usar o botão Feedback desta página para relatá-lo.

Para obter mais orientações sobre solução de problemas e respostas a perguntas comuns de suporte, acesse a Central de Conhecimento da AWS.

Para obter mais informações sobre depuração e solução de problemas de aplicações do Lambda, consulte Debugging no Serverless Land.

Tópicos

Geral: A permissão foi negada/Não é possível carregar esse arquivo

Erro: EACCES: permissão negada, abra '/var/task/index.js'

Erro: não é possível carregar esse arquivo -- função

Erro: [Errno 13] Permissão negada: '/var/task/function.py'

O runtime do Lambda precisa de permissão para ler os arquivos no pacote de implantação. Na notação octal de permissões do Linux, o Lambda precisa de 644 permissões para arquivos não executáveis (rw-r--r--) e 755 permissões (rwxr-xr-x) para diretórios e arquivos executáveis.

No Linux e no MacOS, use o comando `chmod` para alterar as permissões de arquivo em arquivos e diretórios do seu pacote de implantação. Por exemplo, para dar a um arquivo executável as permissões corretas, execute o comando a seguir.

```
chmod 755 <filepath>
```

Para alterar as permissões de arquivo no Windows, consulte Set, View, Change, or Remove Permissions on an Object na documentação do Microsoft Windows.

Geral: Ocorre um erro ao acionar o updateFunctionCode

Erro: ocorreu um erro (RequestEntityTooLargeException) ao chamar a operação UpdateFunctionCode

Quando você faz upload de um pacote de implantação ou de um arquivamento de camada diretamente no Lambda, o tamanho do arquivo ZIP é limitado a 50 MB. Para fazer upload de um arquivo maior, armazene-o no Amazon S3 e use os parâmetros S3Bucket e S3Key.

> **Note**
>
> Quando você faz upload de um arquivo diretamente com a AWS CLI, o AWS SDK ou de outra forma, o arquivo ZIP binário é convertido em base64, o que aumenta o tamanho dele em cerca de 30%. Para permitir isso e o tamanho de outros parâmetros na solicitação, o limite do tamanho real da solicitação que o Lambda aplica é maior. Por isso, o limite de 50 MB é aproximado.

Amazon S3: Código de erro PermanentRedirect.

Erro: Ocorreu um erro ao GetObject. Código de erro S3: PermanentRedirect. Mensagem de erro do S3: O bucket está nesta Região: us-east-2. Use essa região para repetir a solicitação

Ao carregar o pacote de implantação de uma função de um bucket do Amazon S3, o bucket deve estar na mesma Região que a função. Esse problema pode ocorrer ao especificar um objeto do Amazon S3 em uma chamada para UpdateFunctionCode ou usar o pacote e implantar comandos na AWS CLI ou na CLI do AWS SAM. Crie um bucket de artefato de implantação para cada Região em que você desenvolve aplicativos.

Geral: Não é possível localizar, não é possível carregar, não é possível importar, classe não encontrada, o arquivo ou diretório não existe

Erro: não é possível localizar o módulo "function"

Erro: não é possível carregar esse arquivo -- função

Erro: não é possível importar o módulo "function"

Erro: classe não encontrada: function.Handler

Erro: fork/exec /var/task/function: nenhum arquivo ou diretório

Erro: não é possível carregar o tipo "Function.Handler" do assembly "Function".

O nome do arquivo ou classe na configuração do handler da função não corresponde ao seu código. Para obter mais informações, consulte a seção a seguir.

Geral: Handler de método indefinido

Erro: index.handler está indefinido ou não foi exportado

Erro: Handler "handler" ausente no módulo "function"

Erro: método indefinido "handler" para #<LambdaHandler:0x000055b76ccebf98>

Erro: nenhum método público chamado handleRequest com a assinatura de método apropriada encontrada na classe function.Handler

Erro: não foi possível encontrar o método "handleRequest" no tipo "Function.Handler" do assembly "Function"

O nome do método do handler na configuração do handler da função não corresponde ao seu código. Cada runtime define uma convenção de nomenclatura para os handlers, como *`filename.methodname`*. O handler é o método no código da função que é executado pelo runtime quando a função é invocada.

Para alguns idiomas, o Lambda fornece uma biblioteca com uma interface que espera que um método de handler tenha um nome específico. Para obter detalhes sobre a nomenclatura de handlers para cada idioma, consulte os tópicos a seguir.

- Criar funções do Lambda com Node.js
- Criar funções do Lambda com Python
- Construir funções do Lambda com Ruby
- Construir funções do Lambda com Java
- Criar funções do Lambda com Go
- Construir funções do Lambda com C#
- Construir funções do Lambda com o PowerShell

Lambda: falha na conversão de camadas

Erro: falha na conversão de camadas do Lambda. Para obter orientação sobre como resolver esse problema, consulte a página Solucionar problemas de implantação no Lambda no Guia do usuário do Lambda.

Quando você configura uma função do Lambda com uma camada, o Lambda mescla a camada com seu código de função. Se esse processo não for concluído, o Lambda retornará esse erro. Se você se deparar com esse erro, faça o seguinte:

- Exclua todos os arquivos não utilizados da sua camada
- Exclua todos os links simbólicos em sua camada
- Renomeie todos os arquivos que tenham o mesmo nome de um diretório em qualquer uma das camadas da sua função

Lambda: InvalidParameterValueException ou RequestEntityToolargeException

Erro: InvalidParameterValueException: o Lambda não conseguiu configurar as variáveis de ambiente porque as variáveis de ambiente fornecidas excederam o limite de 4 KB. Cadeia de caracteres medida: {"A1":"uSFeY5cyPiPn7AtnX5BsM...

Erro: RequestEntityTooLargeException: a solicitação deve ser menor do que 5.120 bytes para a operação UpdateFunctionConfiguration

O tamanho máximo do objeto de variáveis que é armazenado na configuração da função não deve exceder 4096 bytes. Isso inclui nomes de chaves, valores, aspas, vírgulas e colchetes. O tamanho total do corpo da solicitação HTTP também é limitado.

```
{
    "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"
```

```
        }
    },
    ...
}
```

Neste exemplo, o objeto tem 39 caracteres e ocupa 39 bytes quando é armazenado (sem espaços em branco) como a string `{"BUCKET":"DOC-EXAMPLE-BUCKET","KEY":"file.txt"}`. Caracteres ASCII padrão em valores de variáveis de ambiente usam um byte cada. Caracteres ASCII e Unicode estendidos podem usar entre 2 bytes e 4 bytes por caractere.

Lambda: InvalidParameterValueException

Erro: InvalidParameterValueException: o Lambda não conseguiu configurar as variáveis de ambiente porque as variáveis de ambiente fornecidas contêm chaves reservadas que atualmente não têm suporte para modificação.

O Lambda reserva algumas chaves de variáveis de ambiente para uso interno. Por exemplo, `AWS_REGION` é usada pelo runtime para determinar a região atual e não pode ser substituída. Outras variáveis, como `PATH`, são usadas pelo runtime, mas podem ser estendidas na configuração de função. Para obter uma lista completa, consulte Variáveis de ambiente com runtime definido.

Lambda: simultaneidade e cotas de memória

Erro: ConcurrentExecutions especificado para a função diminui UnreservedConcurrentExecution da conta para um valor abaixo do mínimo

Error: O valor de “MemorySize” não atende à restrição: o valor de Member (Membro) deve ser menor ou igual a 3008

Esses erros ocorrem ao exceder a simultaneidade ou as cotas de memória da conta. As novas contas da AWS reduziram as cotas de simultaneidade e de memória. Para resolver erros relacionados à simultaneidade, é possível solicitar um aumento de cota. Não é possível solicitar aumentos de cota de memória.

- Simultaneidade: é possível receber um erro se tentar criar uma função usando simultaneidade reservada ou provisionada ou se sua solicitação de simultaneidade por função (PutFunctionConcurrency) exceder a cota de simultaneidade da conta.
- Memória: os erros ocorrerão se a quantidade de memória alocada para a função exceder a cota de memória da conta.

Solucionar problemas de invocação no Lambda

Quando você invoca uma função do Lambda, o Lambda valida a solicitação e verifica a capacidade de escalabilidade antes de enviar o evento para a função ou, na invocação assíncrona, para a fila de eventos. Erros de invocação podem ser causados por problemas com os parâmetros da solicitação, a estrutura do evento, as configurações da função, as permissões do usuário, as permissões de recursos ou os limites.

Se você invocar a função diretamente, verá todos os erros de invocação na resposta do Lambda. Se invocar a função de forma assíncrona, com um mapeamento de origem de evento ou por meio de outro serviço, você poderá encontrar erros em logs, em uma fila de mensagens mortas ou em um destino de evento com falha. As opções de tratamento de erros e o comportamento de repetição variam dependendo de como você invoca a função e do tipo de erro.

Para obter uma lista de tipos de erro que a operação `Invoke` pode retornar, consulte Invoke.

IAM: lambda:InvokeFunction não autorizado

Erro: usuário: arn:aws:iam::123456789012:user/developer não tem autorização para executar: lambda:InvokeFunction no recurso: my-function

Seu usuário, ou o perfil que você assume, deve ter permissões para invocar uma função. Esse requisito também se aplica a funções do Lambda e a outros recursos de computação que invocam funções. Adicione a política gerenciada pela AWS AWSLambdaRole ao seu usuário ou adicione uma política personalizada que permita a ação `lambda:InvokeFunction` na função de destino.

> **Note**
>
> O nome da ação do IAM (`lambda:InvokeFunction`) refere-se à operação da API `Invoke` do Lambda.

Para obter mais informações, consulteGerenciando permissões no AWS Lambda

Lambda: não foi possível encontrar um bootstrap válido (Runtime.InvalidEntrypoint)

Erro: não foi possível encontrar bootstrap(s) válido(s): [/var/task/bootstrap /opt/bootstrap]

Esse erro normalmente ocorre quando a raiz do pacote de implantação não contém um arquivo executável denominado `bootstrap`. Por exemplo, se você estiver implantando uma função `provided.al2023` com um arquivo .zip, o arquivo `bootstrap` deverá estar na raiz do arquivo .zip e não em um diretório.

Lambda: A operação não pode ser executada ResourceConflictException

Erro: ResourceConflictException: não é possível executar a operação neste momento. A função está atualmente no seguinte estado: Pendente

Quando você conecta uma função a uma virtual private cloud (VPC) no momento da criação, a função entra em estado `Pending` enquanto o Lambda cria interfaces de rede elástica. Durante esse período, não será possível invocar ou modificar sua função. Se conectar sua função a uma VPC após a criação, você poderá invocá-la enquanto a atualização estiver pendente, mas não poderá modificar seu código ou configuração.

Para obter mais informações, consulteEstados da função do Lambda

Lambda: A função está paralisada em Pendente

Erro: uma função está presa no estado *`Pending`* há vários minutos.

Se uma função ficar paralisada no estado `Pending` por mais de seis minutos, chame uma das operações da API a seguir para desbloqueá-la:

- UpdateFunctionCode
- UpdateFunctionConfiguration
- PublishVersion

O Lambda cancela a operação pendente e coloca a função no estado `Failed`. Em seguida, você pode tentar outra atualização.

Lambda: Uma função está usando toda a simultaneidade

Problema: uma função está usando toda a simultaneidade disponível, fazendo com que outras funções fiquem limitadas.

Para dividir a simultaneidade disponível da sua conta da AWS em uma região da AWS em grupos, use a simultaneidade reservada. A simultaneidade reservada garante que uma função sempre possa escalar para a sua simultaneidade atribuída, e que ela não escale além da simultaneidade atribuída.

Geral: Não é possível invocar a função com outras contas ou serviços

Problema: você consegue invocar sua função diretamente, mas ela não é executada quando outro serviço ou conta a invoca.

Você concede a outros serviços e contas permissão para invocar uma função na política baseada em recursosda função. Se o invocador estiver em outra conta, esse usuário também precisará da permissão para invocar funções.

Geral: A invocação da função está em loop

Problema: a função é invocada continuamente em um loop.

Isso geralmente ocorre quando a sua função gerencia recursos no mesmo serviço da AWS que o aciona. Por exemplo, é possível criar uma função que armazena um objeto em um bucket do Amazon Simple Storage Service (Amazon S3) que é configurado com uma notificação que invoca a função novamente. Para interromper a execução da função, reduza a simultaneidade disponível para zero, o que controla a utilização de todas as invocações futuras. Em seguida, identifique o caminho de código ou erro de configuração que gerou a invocação recursiva. O Lambda detecta e interrompe automaticamente loops recursivos para alguns serviços e SDKs da AWS. Para ter mais informações, consulte the section called “Detecção de loop recursivo”.

Lambda: Roteamento de alias com simultaneidade provisionada

Problema: Chamadas de spillover de simultaneidade provisionadas durante o roteamento de alias.

O Lambda usa um modelo probabilístico simples para distribuir o tráfego entre as duas versões de função. Em níveis de tráfego baixos, você pode ver uma alta variação entre a porcentagem configurada e real de tráfego em cada versão. Se sua função usa simultaneidade provisionada, você pode evitarInvocações de transbordamentoconfigurando um número maior de instâncias de simultaneidade provisionadas durante o tempo em que o roteamento de alias está ativo.

Lambda: As inicializações a frio começam com simultaneidade provisionada

Problema: Você vê inicializações a frio depois de habilitar a simultaneidade provisionada.

Quando o número de execuções simultâneas em uma função é menor ou igual ao nível configurado de simultaneidade provisionada, não deve haver nenhuma inicialização a frio. Para ajudar a confirmar se a simultaneidade provisionada está operando normalmente, faça o seguinte:

- Verifique se a simultaneidade provisionada está habilitada na versão ou alias da função.

> **Note**
>
> A simultaneidade provisionada não é configurável na versão da função não publicada ($LATEST).

- Certifique-se de que seus gatilhos invoquem a versão ou alias correto da função. Por exemplo, se você estiver usando o Amazon API Gateway, verifique se ele invoca a versão ou o alias da função com simultaneidade provisionada, e não a $LATEST. Para confirmar se a simultaneidade provisionada está em uso, você pode verificar a métrica ProvisionedConcurrencyInvocations do Amazon CloudWatch. Um valor diferente de zero indica que a função está processando invocações em ambientes de execução inicializados.
- Determine se a simultaneidade da função excede o nível configurado de simultaneidade provisionada verificando a métrica ProvisionedConcurrencySpilloverInvocations do CloudWatch. Um valor diferente de zero indica que toda a simultaneidade provisionada está em uso e que ocorreu alguma invocação com inicialização a frio.
- Verifique a sua frequência de invocação (solicitações por segundo). Funções com simultaneidade provisionada apresentam uma taxa máxima de dez solicitações por segundo por simultaneidade provisionada. Por exemplo, uma função configurada com 100 casos de simultaneidade provisionada pode lidar com 1.000 solicitações por segundo. Se a taxa de invocação exceder 1.000 solicitações por segundo, podem ocorrer algumas inicializações a frio.

Lambda: As inicializações a frio começam com novas versões

Problema: Você vê inicializações a frio ao implantar novas versões da sua função.

Quando você atualiza um alias da função, o Lambda automaticamente muda a simultaneidade provisionada para a nova versão com base nos pesos configurados no alias.

Erro: KMSDisabledException: o Lambda não conseguiu descriptografar as variáveis de ambiente porque a chave do KMS usada está desabilitada. Verifique as configurações da chave do KMS da função.

Esse erro pode ocorrer se a sua chave do AWS Key Management Service (AWS KMS) estiver desabilitada ou se a concessão que permite que o Lambda use a chave for revogada. Se a concessão estiver ausente, configure a função para usar outra chave. Em seguida, reatribua a chave personalizada para recriar a concessão.

EFS: A função não pôde montar o sistema de arquivos do EFS

Erro: EFSMountFailureException: a função não pôde montar o sistema de arquivos do EFS com o ponto de acesso arn:aws:elasticfilesystem:us-east-2:123456789012:access-point/fsap-015cxmplb72b405fd.

A solicitação de montagem para o sistema de arquivos da função foi rejeitada. Verifique as permissões da função e confirme se o sistema de arquivos e o ponto de acesso existem e estão prontos para uso.

EFS: A função não pôde se conectar ao sistema de arquivos do EFS

Erro: EFSMountConnectivityException: a função não pôde se conectar ao sistema de arquivos do Amazon EFS com o ponto de acesso arn:aws:elasticfilesystem:us-east-2:123456789012:access-point/fsap-015cxmplb72b405fd. Verifique a configuração de rede e tente novamente.

A função não pôde estabelecer uma conexão com o sistema de arquivos da função com o protocolo NFS (TCP porta 2049). Verifique a configuração do grupo de segurança e de roteamento das sub-redes da VPC.

Se você receber esses erros após atualizar as configurações de VPC da função, tente desmontar e remontar o sistema de arquivos.

EFS: A função não pôde montar o sistema de arquivos do EFS devido ao tempo limite

Erro: EFSMountTimeoutException: A função não pôde montar o sistema de arquivos do EFS com o ponto de acesso {arn:aws:elasticfilesystem:us-east-2:123456789012:access-point/fsap-015cxmplb72b405fd} devido ao tempo limite da montagem.

A função pôde se conectar ao sistema de arquivos da função, mas a operação de montagem atingiu o tempo limite. Tente novamente após um breve intervalo e considere limitar a simultaneidade da função para reduzir a carga no sistema de arquivos.

Lambda: O Lambda detectou um processo de E/S que estava demorando muito

EFSIOException: esta instância de função foi interrompida porque o Lambda detectou um processo de E/S que estava demorando muito.

Uma invocação anterior atingiu o tempo limite e o Lambda não conseguiu finalizar o handler de funções. Esse problema pode ocorrer quando um sistema de arquivos anexado fica sem crédito de intermitência e a throughput de linha de base é insuficiente. Para aumentar o throughput, você pode aumentar o tamanho do sistema de arquivos ou usar o throughput provisionado. Para obter mais informações, consulteThroughput

Solucionar problemas de execução no Lambda

Quando o runtime do Lambda executa o código de função, o evento pode ser processado em uma instância da função que esteja processando eventos por algum tempo, ou pode exigir que uma nova instância seja inicializada. Podem ocorrer erros durante a inicialização da função, quando o código do manipulador processa o evento, ou quando a função retorna (ou falha ao retornar) uma resposta.

Erros de execução de função podem ser causados por problemas com o código, a configuração de função, recursos de downstream ou permissões. Se invocar sua função diretamente, você verá erros de função na resposta do Lambda. Se invocar a função de forma assíncrona, com um mapeamento de origem de evento ou por meio de outro serviço, você poderá encontrar erros em logs, em uma fila de mensagens mortas ou em um destino em caso de falha. As opções de tratamento de erros e o comportamento de repetição variam dependendo de como você invoca a função e do tipo de erro.

Quando o código de função ou o runtime do Lambda retornar um erro, o código de status na resposta do Lambda será 200 OK. A presença de um erro na resposta é indicada por um cabeçalho chamado`X-Amz-Function-Error`Os códigos de status das séries 400 e 500 são reservados paraErros de invocação.

Lambda: A execução leva muito tempo

Problema: a execução da função leva muito tempo.

Se o código demorar muito mais tempo para ser executado no Lambda do que na sua máquina local, ele poderá ser restrito pela memória ou pela potência de processamento disponível para a função. Configure a função com memória adicional para aumentar a memória e a CPU.

Lambda: Os logs ou rastreamentos não aparecem

Problema: Os logs não aparecem no CloudWatch Logs.

Problema: os rastreamentos não aparecem no AWS X-Ray.

Sua função precisa de permissão para chamar o CloudWatch Logs e o X-Ray. Atualize a função de execução para conceder permissão a ela. Adicione as políticas gerenciadas a seguir para habilitar logs e rastreamento.

- AWSLambdaBasicExecutionRole
- AWSXRayDaemonWriteAccess

Ao adicionar permissões à sua função, faça também uma atualização simples em seu código ou configuração. Isso força as instâncias em execução da função, com credenciais desatualizadas, a serem encerradas e substituídas.

> **Note**
>
> Pode levar de 5 a 10 minutos para que os logs apareçam após uma invocação de função.

Lambda: nem todos os logs da função aparecem

Problema: os logs de funções estão ausentes no CloudWatch Logs, mesmo que minhas permissões estejam corretas

Se sua Conta da AWS atingir os limites de cota do CloudWatch Logs, o CloudWatch controlará a utilização do registro em log de funções. Quando isso acontece, alguns dos logs gerados pelas funções podem não aparecer no CloudWatch Logs.

Se a função gerar logs em uma taxa muito alta para o Lambda processá-los, isso também pode fazer com que os logs não apareçam no CloudWatch Logs. Quando o Lambda não consegue enviar logs para o CloudWatch na velocidade em que a função os produz, ele descarta logs para impedir que a execução da função fique mais lenta.

Se sua função estiver configurada para usar logs no formato JSON, o Lambda tentará enviar um evento `logsDropped` para o CloudWatch Logs ao descartar os logs. No entanto, quando o CloudWatch controla a utilização do registro de log da sua função, esse evento pode não chegar ao CloudWatch Logs; por isso, você nem sempre verá um registro quando o Lambda descartar os logs.

Para verificar se sua Conta da AWS atingiu os limites de cota do CloudWatch Logs, faça o seguinte:

1. Abra o console do Service Quotas.
2. No painel de navegação, escolha Serviços da AWS.

3. Na lista Serviços da AWS, procure Amazon CloudWatch Logs.
4. Na lista Service Quotas, escolha as cotas `CreateLogGroup throttle limit in transactions per second`, `CreateLogStream throttle limit in transactions per second` e `PutLogEvents throttle limit in transactions per second` para visualizar sua utilização.

Você também pode definir alarmes do CloudWatch para receber um alerta quando a utilização da sua conta exceder o limite especificado para essas cotas. Consulte Criar um alarme do CloudWatch com base em um limite estático para saber mais.

Se os limites de cota padrão do CloudWatch Logs não forem suficientes para seu caso de uso, você poderá solicitar um aumento de cota.

Lambda: A função retorna antes da conclusão da execução

Problema: (Node.js) a função retorna antes que o código termine de ser executado

Muitas bibliotecas, incluindo o AWS SDK, operam de forma assíncrona. Ao fazer uma chamada de rede ou executar outra operação que exija aguardar uma resposta, as bibliotecas retornam um objeto chamado de promessa que rastreia o progresso da operação em segundo plano.

Para aguardar que a promessa seja resolvida em uma resposta, use a palavra-chave `await`. Isso impedirá que o código do handler seja executado até que a promessa seja resolvida em um objeto que contenha a resposta. Se não precisar usar os dados da resposta em seu código, você poderá retornar a promessa diretamente ao runtime.

Algumas bibliotecas não retornam promessas, mas podem ser encapsuladas em um código que retorne. Para ter mais informações, consulte Definir o manipulador de função do Lambda em Node.js.

AWS SDK: versões e atualizações

Problema: o AWS SDK incluído no runtime não é da versão mais recente

Problema: o AWS SDK incluído no runtime é atualizado automaticamente

Os tempos de execução para linguagens de desenvolvimento de scripts incluem o AWS SDK e são atualizados periodicamente para a versão mais recente. A versão atual para cada runtime está listada na página de runtimes. Para usar uma versão mais recente do AWS SDK ou para bloquear suas funções em uma versão específica, agrupe a biblioteca com seu código de função ou crie

uma camada do Lambda. Para obter detalhes sobre como criar um pacote de implantação com dependências, consulte os seguintes tópicos:

Node.js

- Implantar funções do Lambda em Node.js com arquivos .zip

Python

- Trabalhar com arquivos .zip para funções do Lambda em Python

Ruby

- Como trabalhar com arquivos .zip para funções do Lambda em Ruby

Java

- Implantar funções do Lambda em Java com arquivos .zip ou JAR

Go

- Implantar funções do Lambda em Go com arquivos .zip

C#

- Criar e implantar funções do Lambda em C# com arquivos .zip

PowerShell

- Implemente funções PowerShell Lambda com arquivos de arquivos.zip

Python: As bibliotecas carregam incorretamente

Problema: (Python) algumas bibliotecas não carregam corretamente do pacote de implantação

Bibliotecas com módulos de extensão escritos em C ou C++ devem ser compiladas em um ambiente com a mesma arquitetura de processador que o Lambda (Amazon Linux). Para ter mais informações, consulte Trabalhar com arquivos .zip para funções do Lambda em Python.

Solucionar problemas de redes no Lambda

Por padrão, o Lambda executa suas funções em uma Virtual Private Cloud (VPC) interna com conectividade aos serviços da AWS e à Internet. Para acessar os recursos da rede local, é possível configurar sua função para se conectar a uma VPC em sua conta. Ao usar esse recurso, você

gerencia o acesso da função à Internet e a conectividade de rede com recursos da Amazon Virtual Private Cloud (Amazon VPC).

Os erros de conectividade de rede podem ser originados de problemas na configuração de roteamento de sua VPC, regras de grupo de segurança, permissões de função do AWS Identity and Access Management (IAM), NAT ou disponibilidade de recursos, como endereços IP ou interfaces de rede. Conforme o problema, será possível ver um erro específico ou tempo limite se a solicitação não puder chegar ao destino.

VPC: A função perde o acesso à Internet ou atinge o tempo limite

Problema: a função do Lambda perde o acesso à Internet após se conectar a uma VPC.

Erro: Erro: conexão ETIMEDOUT 176.32.98.189:443

Erro: Erro: a tarefa expirou após 10,00 segundos

Erro: ReadTimeoutError: Read timed out. (read timeout=15) (Tempo limite excedeu)

Ao conectar uma função a uma VPC, todas as solicitações de saída passam pela VPC. Para se conectar à Internet, configure a VPC para enviar tráfego de saída da sub-rede da função para um gateway NAT em uma sub-rede pública. Para obter mais informações e exemplos de configurações de VPC, consulte the section called “Acesso à Internet para funções da VPC”.

Se algumas de suas conexões TCP estiverem expirando, pode ser devido à fragmentação de pacotes. As funções Lambda não podem lidar com o recebimento de solicitações TCP fragmentadas, pois o Lambda não oferece suporte à fragmentação de IP para TCP nem para ICMP.

VPC: a função precisa de acesso aos serviços da AWS sem usar a Internet

Problema: a função do Lambda precisa acessar os produtos da AWS sem usar a Internet.

Para conectar uma função aos produtos da AWS de uma sub-rede privada sem acesso à Internet, use os endpoints da VPC.

VPC: Elastic network interface limit reached (VPC: limite da interface de rede elástica atingido)

Erro: ENILimitReachedException: o limite de interface de rede elástica foi atingido para a VPC da função.

Quando você conecta uma função do Lambda a uma VPC, o Lambda cria uma interface de rede elástica para cada combinação de sub-rede e grupo de segurança anexado à função. A cota de serviço padrão é de 250 interfaces de rede por VPC. Para solicitar aumento de cota, use o console do Service Quotas.

EC2: interface de rede elástica com o tipo “lambda”

Código de erro: Client.OperationNotPermited

Mensagem de erro: o grupo de segurança não pode ser modificado para esse tipo de interface

Você receberá esse erro se tentar modificar uma interface de rede elástica (ENI) gerenciada pelo Lambda. O `ModifyNetworkInterfaceAttribute` não está incluído na API do Lambda para operações de atualização em interfaces de rede elástica criadas pela Lambda.

Aplicações do AWS Lambda

Uma aplicação do AWS Lambda é uma combinação de funções do Lambda, fontes de eventos e outros recursos que trabalham juntos para realizar tarefas. Você pode usar o AWS CloudFormation e outras ferramentas para coletar os componentes do aplicativo em um único pacote que pode ser implantado e gerenciado como um recurso. As aplicações tornam seus projetos do Lambda portáteis e permitem a integração com ferramentas de desenvolvedor adicionais, como o AWS CodePipeline, o AWS CodeBuild e a interface de linha de comando do AWS Serverless Application Model (CLI do AWS SAM).

O AWS Serverless Application Repository fornece uma coleção de aplicações do Lambda que você pode implantar na sua conta com alguns cliques. O repositório inclui aplicativos e amostras prontas para uso que você pode usar como ponto de partida para seus próprios projetos. Você também pode enviar seus próprios projetos para inclusão.

O AWS CloudFormation permite criar um modelo que define os recursos do aplicativo e permite gerenciar o aplicativo como uma pilha. Você pode adicionar ou modificar recursos com mais segurança na sua pilha de aplicativos. Se qualquer parte de uma atualização falhar, o AWS CloudFormation reverterá automaticamente para a configuração anterior. Com os parâmetros do AWS CloudFormation, é possível criar vários ambientes para as aplicações usando o mesmo modelo. O AWS SAM estende o AWS CloudFormation com uma sintaxe simplificada com foco no desenvolvimento de aplicações do Lambda.

A AWS CLI e a CLI do AWS SAM são ferramentas de linha de comando para gerenciamento de pilhas de aplicações do Lambda. Além de comandos para gerenciar pilhas de aplicações com a API do AWS CloudFormation, o suporte do AWS CLI a comandos de nível superior que simplificam tarefas como o upload de pacotes de implantação e a atualização de modelos. A CLI do AWS SAM fornece outras funcionalidades, como validação de modelos, testes localmente e integração com sistemas de CI/CD.

Ao criar uma aplicação, você pode criar seu repositório Git usando o CodeCommit ou uma conexão do AWS CodeStar com o GitHub. O CodeCommit permite que você use o console do IAM para gerenciar chaves SSH e credenciais HTTP para seus usuários. O CodeConnections permite que você se conecte à sua conta do GitHub. Para obter mais informações sobre conexões, consulte What are connections? no Developer Tools console User Guide (Guia do usuário do console de ferramentas do desenvolvedor).

Para obter mais informações sobre como projetar aplicações do Lambda, consulte Application design no Serverless Land.

Tópicos

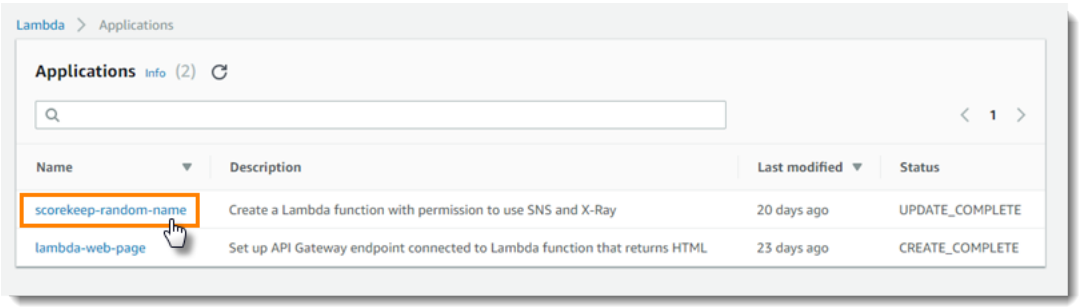
- Gerenciar aplicativos no console do AWS Lambda
- Criar implantações contínuas para funções do Lambda
- Usar o Lambda com o Kubernetes

Gerenciar aplicativos no console do AWS Lambda

O console do AWS Lambda ajuda a monitorar e gerenciar suas aplicações do Lambda. O menu Applications (Aplicações) lista pilhas do AWS CloudFormation com funções do Lambda. O menu inclui pilhas que você inicia no AWS CloudFormation usando o console do AWS CloudFormation, o AWS Serverless Application Repository, o AWS CLI ou a CLI do AWS SAM.

Para visualizar uma aplicação do Lambda

1. Abra a página Applications (Aplicações) do console do Lambda.
2. Escolha a aplicação.



A visão geral mostra as seguintes informações sobre o seu aplicativo.

- AWS CloudFormation template (Modelo do CloudFormation) ou SAM template (Modelo do SAM): o modelo que define sua aplicação.
- Resources (Recursos): os recursos da AWS definidos no modelo de sua aplicação. Para gerenciar as funções do Lambda de sua aplicação, escolha um nome de função na lista.

Monitorar aplicativos

A guia Monitoramento mostra um CloudWatch painel da Amazon com métricas agregadas para os recursos em seu aplicativo.

Para monitorar uma aplicação do Lambda

1. Abra a página Applications (Aplicações) do console do Lambda.
2. Escolha Monitoramento.

Por padrão, o console do Lambda mostra um painel básico. Você pode personalizar essa página definindo painéis personalizados em seu template de aplicativo. Quando seu template inclui um ou mais painéis, a página mostra seus painéis em vez do painel padrão. Você pode alternar entre painéis com o menu suspenso no canto superior direito da página.

Painéis de monitoramento personalizados

Personalize sua página de monitoramento de aplicativos adicionando um ou mais CloudWatch painéis da Amazon ao seu modelo de aplicativo com o tipo de AWS::CloudWatch::Dashboardrecurso. O exemplo a seguir cria um painel com um único widget que representa graficamente o número de invocações de uma função denominada `my-function`.

Example modelo de painel de função

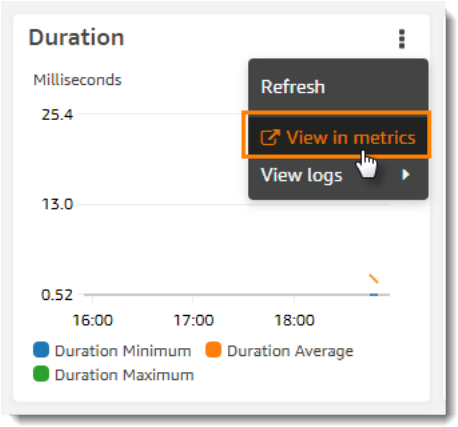
```
Resources:
  MyDashboard:
    Type: AWS::CloudWatch::Dashboard
    Properties:
      DashboardName: my-dashboard
      DashboardBody: |
        {
            "widgets": [
                {
                    "type": "metric",
                    "width": 12,
                    "height": 6,
                    "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
            }
        }
    ]
}
```

Você pode obter a definição para qualquer um dos widgets no painel de monitoramento padrão no console do CloudWatch.

Para visualizar uma definição de widget

1. Abra a página Applications (Aplicações) do console do Lambda.
2. Escolha um aplicativo que tenha o painel padrão.
3. Escolha Monitoramento.
4. Em qualquer widget, escolha View in metrics (Visualizar nas métricas) no menu suspenso.



5. Selecione Source.

Para obter mais informações sobre a criação de CloudWatch painéis e widgets, consulte Estrutura e sintaxe do corpo do painel na Amazon API Reference. CloudWatch

Criar implantações contínuas para funções do Lambda

Use implantações contínuas para controlar os riscos associados à introdução de novas versões da sua função do Lambda. Em uma implantação contínua, o sistema implanta automaticamente a nova versão da função e envia gradualmente uma quantidade crescente de tráfego para a nova versão. A quantidade de tráfego e a taxa de aumento são parâmetros que você pode configurar.

Configurar uma implementação sem interrupções usando o AWS CodeDeploy e o AWS SAM. CodeDeploy é um serviço que automatiza implantações de aplicativos para plataformas de computação da Amazon, como o Amazon EC2 eAWS Lambda. Para obter mais informações, consulte O que é o CodeDeploy?. Usando o CodeDeploy para implantar sua função do Lambda, é possível monitorar com facilidade o status da implantação e iniciar uma reversão se detectar algum problema.

O AWS SAM é um framework de código aberto para a criação de aplicações sem servidor. Crie um modelo de AWS SAM (no formato YAML) para especificar a configuração dos componentes necessários para a implantação contínua. O AWS SAM usa o modelo para criar e configurar os componentes. Para obter mais informações, consulte O que é o AWS SAM?.

Em uma implantação contínua, o AWS SAM executa estas tarefas:

- Configura sua função do Lambda e cria um alias.

  A configuração de roteamento de alias é o recurso subjacente que implementa a implantação contínua.
- Cria um aplicativo CodeDeploy e um grupo de implantação.

  O grupo de implantação gerencia a implantação contínua e a reversão (se necessário).
- Detecta quando você cria uma nova versão da sua função do Lambda.
- Aciona o CodeDeploy para iniciar a implantação da nova versão.

Exemplo de modelo do Lambda do AWS SAM

O exemplo a seguir mostra um Modelo do AWS SAM para uma implantação contínua simples.

```
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
```

Este modelo define uma função do Lambda denominada `myDateTimeFunction` com as propriedades a seguir.

AutoPublishAlias

A propriedade `AutoPublishAlias` cria um alias chamado `live`. Além disso, o framework AWS SAM detecta automaticamente quando você salva um código novo para a função. O framework publica uma nova versão da função e atualiza o alias do `live` para apontar para a nova versão.

DeploymentPreference

A propriedade `DeploymentPreference` determina a taxa na qual a aplicação do CodeDeploy muda o tráfego da versão original da função do Lambda para a nova versão. O valor `Linear10PercentEvery2Minutes` muda dez por cento adicionais do tráfego para a nova versão a cada dois minutos.

Para obter uma lista das configurações de implantação predefinidas, consulte Configurações de implantação.

Para obter um tutorial detalhado sobre como usar o CodeDeploy com funções do Lambda, consulte Implantar uma função do Lambda atualizada com o CodeDeploy.

Usar o Lambda com o Kubernetes

Você pode implantar e gerenciar funções do Lambda com a API do Kubernetes usando o AWS Controllers for Kubernetes (ACK) ou o Crossplane.

AWS Controlllers for Kubernetes (ACK)

Você pode usar o ACK para implantar e gerenciar recursos da AWS da API do Kubernetes. Por meio do ACK, AWS fornece controladores personalizados de código aberto para AWS serviços como Lambda, Amazon Elastic Container Registry (Amazon ECR), Amazon Simple Storage Service (Amazon S3) e Amazon. SageMaker Cada serviço da AWS compatível tem seu próprio controlador personalizado. Em seu cluster do Kubernetes, instale um controlador para cada serviço da AWS que você deseja usar. Em seguida, crie uma Definição de recursos personalizados (CRD) para definir os recursos da AWS.

Recomendamos que você use Helm 3.8 ou posterior para instalar controladores ACK. Todo controlador do ACK vem com seu próprio chart do Helm, que instala o controlador, os CRDs e as regras RBAC do Kubernetes. Para obter mais informações, consulte Install an ACK Controller na documentação do ACK.

Depois de criar o recurso do ACK personalizado, você pode usá-lo como qualquer outro objeto incorporado do Kubernetes. Por exemplo, você pode implantar e gerenciar funções do Lambda com suas cadeias de ferramentas do Kubernetes preferenciais, incluindo kubectl.

Aqui estão alguns exemplos de casos de uso para provisionar funções do Lambda por meio do ACK:

- Sua organização usa controle de acesso baseado em funções (RBAC) e perfis do IAM para contas de serviço para criar limites de permissões. Com o ACK, você pode reutilizar esse modelo de segurança para o Lambda sem precisar criar novos usuários e políticas.
- Sua organização tem um DevOps processo para implantar recursos em um cluster do Amazon Elastic Kubernetes Service (Amazon EKS) usando manifestos do Kubernetes. Com o ACK, você pode usar um manifesto para provisionar funções do Lambda sem criar uma infraestrutura separada como modelos de código.

Para obter mais informações sobre o uso do ACK, consulte Lambda tutorial in the ACK documentation.

Crossplane

Crossplane é um projeto de código aberto da Cloud Native Computing Foundation (CNCF) que usa o Kubernetes para gerenciar recursos de infraestrutura em nuvem. Com o Crossplane, os desenvolvedores podem solicitar infraestrutura sem precisar entender suas complexidades. As equipes da plataforma retêm o controle sobre como a infraestrutura é provisionada e gerenciada.

Usando o Crossplane, você pode implantar e gerenciar funções do Lambda com suas cadeias de ferramentas Kubernetes preferenciais, como kubectl, e qualquer pipeline de CI/CD que possa implantar manifestos no Kubernetes. Aqui estão alguns exemplos de casos de uso para provisionar funções do Lambda por meio do Crossplane:

- Sua organização deseja impor a conformidade garantindo que as funções do Lambda tenham as tags corretas. As equipes da plataforma podem usar Composições de Crossplane para definir essa política por meio de abstrações de API. Os desenvolvedores podem então usar essas abstrações para implantar funções do Lambda com tags.
- Seu projeto usa GitOps com o Kubernetes. Nesse modelo, o Kubernetes reconcilia continuamente o repositório git (estado desejado) com os recursos em execução dentro do cluster (estado atual). Se houver diferenças, o GitOps processo fará alterações automaticamente no cluster. Você pode usar GitOps com o Kubernetes para implantar e gerenciar funções Lambda por meio do Crossplane, usando ferramentas e conceitos familiares do Kubernetes, como CRDs e controladores.

Para obter mais informações sobre como usar o Crossplane com o Lambda, veja o seguinte:

- Esquemas para Crossplane da AWS: este repositório inclui exemplos de como usar o Crossplane para implantar recursos da AWS, incluindo funções do Lambda.

  > **Note**
  >
  > Os esquemas para Crossplane da AWS estão em desenvolvimento ativo e não devem ser usados na produção.

- Deploying Lambda with Amazon EKS and Crossplane: este vídeo demonstra um exemplo avançado de implantação de um arquitetura sem servidor da AWS com Crossplane, explorando o design tanto da perspectiva do desenvolvedor quanto da plataforma.

Aplicativos de exemplo do Lambda

O repositório do GitHub para este guia inclui aplicações de exemplo que demonstram o uso de várias linguagens e serviços da AWS. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS SAM e recursos de suporte.

Node.js

Aplicações de exemplo do Lambda em Node.js

- blank-nodejs: uma função do Node.js que mostra o uso do registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.
- nodejs-apig: uma função com endpoint de API pública que processa um evento do API Gateway e retorna uma resposta HTTP.
- efs-nodejs: uma função que usa um sistema de arquivos do Amazon EFS em uma Amazon VPC. Esse exemplo inclui uma VPC, um sistema de arquivos, destinos de montagem e ponto de acesso configurado para uso com o Lambda.

Python

Aplicativos do Lambda de exemplo do em Python

- blank-python: uma função Python que mostra o uso de registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.

Ruby

Aplicações de exemplo do Lambda em Ruby

- blank-ruby: uma função do Ruby que mostra o uso de registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.
- Ruby Code Samples for AWS Lambda: exemplos de código escritos em Ruby que demonstram como interagir com o AWS Lambda.

Java

Aplicações de exemplo do Lambda em Java

- java17-examples: uma função em Java que demonstra como usar um registro Java para representar um objeto de dados de evento de entrada.
- java-basic: uma coleção de funções Java mínimas com testes de unidade e configuração de registro em log variável.
- java-events: uma coleção de funções do Java contendo código básico sobre como lidar com eventos de vários serviços, como o Amazon API Gateway, o Amazon SQS e o Amazon Kinesis. Essas funções usam a versão mais recente da biblioteca aws-lambda-java-events (3.0.0 e versões mais recentes). Estes exemplos não exigem oAWS SDK como dependência.
- s3-java: uma função em Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.
- Use API Gateway to invoke a Lambda function (Usar o API Gateway para invocar uma função do Lambda): uma função Java que verifica uma tabela do Amazon DynamoDB contendo informações de funcionários. Em seguida, usa o Amazon Simple Notification Service para enviar uma mensagem de texto aos funcionários comemorando seus aniversários de empresa. Este exemplo usa o API Gateway para invocar a função.

Executar estruturas Java populares no Lambda

- spring-cloud-function-samples: um exemplo da Spring que mostra como usar a estrutura Spring Cloud Function para criar funções do AWS Lambda.
- Demonstração da aplicação Spring Boot sem servidor: um exemplo que mostra como configurar uma aplicação Spring Boot típica em um runtime Java gerenciado com e sem o SnapStart, ou como uma imagem nativa do GraalVM com um runtime personalizado.
- Demonstração da aplicação Micronaut sem servidor: um exemplo que mostra como usar o Micronaut em um runtime Java gerenciado com e sem o SnapStart, ou como uma imagem nativa do GraalVM com um runtime personalizado. Saiba mais nos guias do Micronaut/Lambda.
- Demonstração da aplicação Quarkus sem servidor: um exemplo que mostra como usar o Quarkus em um runtime Java gerenciado com e sem o SnapStart, ou como uma imagem nativa do GraalVM com um runtime personalizado. Saiba mais no guia do Quarkus/Lambda e no guia do Quarkus/SnapStart.

Go

O Lambda fornece as seguintes aplicações de exemplo para o runtime do Go:

Aplicativos do Lambda de exemplo do em Go

- go-al2: uma função olá, mundo que retorna o endereço IP público. Esta aplicação usa o runtime `provided.al2` personalizado.
- blank-go: uma função do Go que mostra o uso das bibliotecas do Go do Lambda, o registro em log, as variáveis de ambiente e o AWS SDK. Esta aplicação usa o runtime `go1.x`.

C#

Aplicativos do Lambda de exemplo do em C#

- blank-csharp: uma função em C# que mostra o uso das bibliotecas .NET do Lambda, do registro em log, das variáveis de ambiente, do rastreamento do AWS X-Ray, dos testes de unidade e do AWS SDK.
- blank-csharp-com-layer: uma função C# que usa a CLI do .NET para criar uma camada que empacota as dependências da função.
- ec2-spot: uma função que gerencia solicitações de instâncias spot no Amazon EC2.

PowerShell

O Lambda fornece as seguintes aplicações de amostra para PowerShell:

- blank-powershell: uma função do PowerShell que mostra o uso do registro em log, as variáveis de ambiente e o AWS SDK.

Para implantar um aplicativo de exemplo, siga as instruções no arquivo README. Para saber mais sobre a arquitetura e casos de uso de um aplicativo, leia os tópicos neste capítulo.

Tópicos

- Aplicativo de exemplo de função em branco para o AWS Lambda

Aplicativo de exemplo de função em branco para o AWS Lambda

A aplicação de exemplo de função em branco é uma aplicação inicial que demonstra operações comuns no Lambda com uma função que chama a API do Lambda. Ele mostra o uso de registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK. Explore essa aplicação para saber como criar funções do Lambda em sua linguagem de programação, ou usá-lo como ponto de partida para seus próprios projetos.

As variantes deste aplicativo de exemplo estão disponíveis nas seguintes linguagens:

Variantes

- Node.js: blank-nodejs.
- Python: blank-python.
- Ruby: blank-ruby.
- Java: blank-java.
- Go: blank-go.
- C#: blank-csharp.
- PowerShell – blank-powershell.

Os exemplos neste tópico destacam o código da versão Node.js, mas os detalhes são geralmente aplicáveis a todas as variantes.

Você pode implantar a amostra em alguns minutos com a AWS CLI e o AWS CloudFormation. Siga as instruções no README para fazer download, configurar e implantá-la na conta.

Seções

Arquitetura e código do manipulador

O aplicativo de exemplo consiste em um código de função, um modelo do AWS CloudFormation e recursos de suporte. Ao implantar o exemplo, você usa os seguintes serviços da AWS:

- AWS Lambda: executa o código da função, envia logs para o CloudWatch Logs e envia dados de rastreamento para o X-Ray. A função também chama a API do Lambda para obter detalhes sobre as cotas e o uso da conta na região atual.
- AWS X-Ray: coleta de dados de rastreamento, indexa rastreamentos para pesquisa e gera um mapa de serviço.
- Amazon CloudWatch— armazena logs e métricas.
- AWS Identity and Access Management(IAM)— concede permissão.
- Amazon Simple Storage Service (Amazon S3)— armazena o pacote de implantação da função durante a implantação.
- AWS CloudFormation: cria recursos da aplicação e implanta o código da função.

Aplicam-se taxas padrão para cada serviço. Para obter mais informações, consulte Preços do AWS.

O código da função mostra um fluxo de trabalho básico para processar um evento. O manipulador pega um evento do Amazon Simple Queue Service (Amazon SQS) como entrada e percorre pelos registros que ele contém, registrando em log o conteúdo de cada mensagem. Ele registra o conteúdo do evento, o objeto de contexto e as variáveis de ambiente. Depois, ele faz uma chamada com o AWS SDK e passa a resposta de volta para o tempo de execução do Lambda.

Example blank-nodejs/function/index.js: código do handler

```
// 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);
};
```

A amostra de aplicação não inclui uma fila do Amazon SQS para enviar eventos, mas usa um evento do Amazon SQS (event.json) para ilustrar como os eventos são processados. Para adicionar uma fila do Amazon SQS à aplicação, consulte Usar o Lambda com o Amazon SQS.

Automação de implantação com o AWS CloudFormation e a AWS CLI

Os recursos do aplicativo de exemplo são definidos em um modelo do AWS CloudFormation e implantados com a AWS CLI. O projeto inclui scripts de shell simples que automatizam o processo de configuração, implantação, invocação e destruição do aplicativo.

O modelo de aplicativo usa um tipo de recurso AWS Serverless Application Model (AWS SAM) para definir o modelo. O AWS SAM simplifica a criação de modelos para aplicativos sem servidor automatizando a definição de funções de execução, APIs e outros recursos.

O modelo define os recursos na pilha de aplicativos. Isso inclui a função, sua função de execução e uma camada do Lambda que fornece as dependências da biblioteca da função. A pilha não inclui o bucket que a AWS CLI usa durante a implantação nem o grupo de logs do CloudWatch Logs.

Example blank-nodejs/template.yml: recursos sem servidor

```
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 você implanta o aplicativo, o AWS CloudFormation aplica a transformação do AWS SAM ao modelo para gerar um modelo do AWS CloudFormation com tipos padrão, como `AWS::Lambda::Function` e `AWS::IAM::Role`.

Example modelo processado

```
{
  "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"
          }
```

```
        ],
```

Neste exemplo, a propriedade `Code` especifica um objeto em um bucket do Amazon S3. Isso corresponde ao caminho local na propriedade `CodeUri` no modelo de projeto:

```
      CodeUri: function/.
```

Para fazer upload dos arquivos do projeto no Amazon S3, o script de implantação usa comandos na AWS CLI. O comando `cloudformation package` pré-processa o modelo, faz upload de artefatos e substitui caminhos locais por locais de objetos do Amazon S3. O comando `cloudformation deploy` implanta o modelo processado com um conjunto de alterações do AWS CloudFormation.

Example blank-nodejs/3-deploy.sh: empacotar e implantar

```
#!/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
```

Da primeira vez que você executar esse script, ele criará uma pilha do AWS CloudFormation chamada `blank-nodejs`. Se fizer alterações no modelo ou no código da função, você poderá executá-la novamente para atualizar a pilha.

O script de limpeza (blank-nodejs/5-cleanup.sh) exclui a pilha e, opcionalmente, exclui o bucket de implantação e os logs de função.

Instrumentação com o AWS X-Ray

A função de exemplo é configurada para rastreamento com AWS X-Ray. Com o modo de rastreamento definido como ativo, o Lambda registra informações de temporização para um subconjunto de invocações e as envia para o X-Ray. O X-Ray processa os dados para gerar um Mapa de serviço que mostra um nó de cliente e dois nós de serviço.

O primeiro nó de serviço (`AWS::Lambda`) representa o serviço do Lambda, que valida a solicitação de invocação e o envia para a função. O segundo nó, `AWS::Lambda::Function`, representa a própria função.

Para registrar detalhes adicionais, a função de exemplo usa o X-Ray SDK. Com alterações mínimas no código de função, o X-Ray SDK registra detalhes sobre chamadas feitas com o AWS SDK para serviços da AWS.

Example blank-nodejs/function/index.js: instrumentação

```
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());
```

Instrumentar o cliente AWS SDK adiciona um nó extra ao mapa de serviços e mais detalhes em rastreamentos. Neste exemplo, o mapa de serviço mostra a função de exemplo chamando a API do Lambda para obter detalhes sobre armazenamento e uso de simultaneidade na região atual.

O rastreamento mostra detalhes de temporização para a invocação, com subsegmentos para inicialização da função, invocação e sobrecarga. O subsegmento de invocação tem um subsegmento para a chamada do AWS SDK para a operação de API `GetAccountSettings`.

Você pode incluir o X-Ray SDK e outras bibliotecas no pacote de implantação da função ou implantá-las separadamente em uma camada do Lambda. Para Node.js, Ruby e Python, o tempo de execução do Lambda inclui o AWS SDK no ambiente de execução.

Gerenciamento de dependências com camadas

Você pode instalar bibliotecas localmente e incluí-las no pacote de implantação que carregar no Lambda, mas isso tem suas desvantagens. Tamanhos de arquivo maiores geram tempos de implantação maiores e podem impedir que você teste alterações em seu código de função no console do Lambda. Para manter o pacote de implantação pequeno e evitar o upload de dependências que não foram alteradas, à aplicação de exemplo cria uma camada do Lambda e a associa à função.

Example blank-nodejs/template.yml: camada de dependência

```
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
```

O script `2-build-layer.sh` instala as dependências da função com o npm e as coloca em uma pasta com a estrutura exigida pelo tempo de execução do Lambda.

Example 2-build-layer.sh: preparar a camada

```
#!/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/
```

Ao implantar o aplicativo de exemplo pela primeira vez, a AWS CLI empacota a camada separadamente do código de função e implanta ambos. Para implantações subsequentes, o arquivo de camada só será carregado se o conteúdo da pasta `lib` tiver sido alterado.

Utilizar o Lambda com um AWS SDK

Os kits de desenvolvimento de software (SDKs) da AWS estão disponíveis para muitas linguagens de programação populares. Cada SDK fornece uma API, exemplos de código e documentação que facilitam a criação de aplicações em seu idioma preferido pelos desenvolvedores.

Documentação do SDK	Exemplos de código
AWS SDK for C++	Exemplos de código do AWS SDK for C++
AWS CLI	Exemplos de código do AWS CLI
AWS SDK for Go	Exemplos de código do AWS SDK for Go
AWS SDK for Java	Exemplos de código do AWS SDK for Java
AWS SDK for JavaScript	Exemplos de código do AWS SDK for JavaScript
AWS SDK para Kotlin	Exemplos de código do AWS SDK para Kotlin
AWS SDK for .NET	Exemplos de código do AWS SDK for .NET
AWS SDK for PHP	Exemplos de código do AWS SDK for PHP
AWS Tools for PowerShell	Tools for PowerShell code examples
AWS SDK for Python (Boto3)	Exemplos de código do AWS SDK for Python (Boto3)
AWS SDK for Ruby	Exemplos de código do AWS SDK for Ruby
AWS SDK para Rust	Exemplos de código do AWS SDK para Rust
SDK da AWS para SAP ABAP	Exemplos de código do SDK da AWS para SAP ABAP
AWS SDK for Swift	Exemplos de código do AWS SDK for Swift

Para obter exemplos específicos do Lambda, consulte Exemplos de código do Lambda usando AWS SDKs.

> **Exemplo de disponibilidade**
>
> Você não consegue encontrar o que precisa? Solicite um código de exemplo no link Fornecer feedback na parte inferior desta página.

Exemplos de código do Lambda usando AWS SDKs

Os exemplos de código a seguir mostram como usar o Lambda com um Kit de Desenvolvimento de Software (SDK) da AWS.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar funções de serviço específicas, é possível ver as ações contextualizadas em seus devidos cenários e exemplos entre serviços.

Cenários são exemplos de código que mostram como realizar uma tarefa específica chamando várias funções dentro do mesmo serviço.

Exemplos entre serviços são amostras de aplicações que funcionam em vários Serviços da AWS.

Para obter uma lista completa dos Guias do desenvolvedor do SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Conceitos básicos

Olá, Lambda

Os exemplos de código a seguir mostram como começar a usar o Lambda.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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}");
        });
    }
}
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK for .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Código para o arquivo CMakeLists.txt do CMake.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS lambda)

# Set this project's name.
project("hello_lambda")
```

```
# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_lambda.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Código para o arquivo de origem 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;
}
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK for C++.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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)
  }
```

```
 }
}
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK for Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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);
        }
    }
}
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
};
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK for JavaScript.

Exemplos de código

- Ações do Lambda usando AWS SDKs
  - Usar CreateAlias com o AWS SDK ou a CLI
  - Usar CreateFunction com o AWS SDK ou a CLI
  - Usar DeleteAlias com o AWS SDK ou a CLI
  - Usar DeleteFunction com o AWS SDK ou a CLI
  - Usar DeleteFunctionConcurrency com o AWS SDK ou a CLI
  - Usar DeleteProvisionedConcurrencyConfig com o AWS SDK ou a CLI
  - Usar GetAccountSettings com o AWS SDK ou a CLI
  - Usar GetAlias com o AWS SDK ou a CLI
  - Usar GetFunction com o AWS SDK ou a CLI
  - Usar GetFunctionConcurrency com o AWS SDK ou a CLI
  - Usar GetFunctionConfiguration com o AWS SDK ou a CLI
  - Usar GetPolicy com o AWS SDK ou a CLI
  - Usar GetProvisionedConcurrencyConfig com o AWS SDK ou a CLI
  - Usar Invoke com o AWS SDK ou a CLI
  - Usar ListFunctions com o AWS SDK ou a CLI
  - Usar ListProvisionedConcurrencyConfigs com o AWS SDK ou a CLI
  - Usar ListTags com o AWS SDK ou a CLI

- Usar ListVersionsByFunction com o AWS SDK ou a CLI
- Usar PublishVersion com o AWS SDK ou a CLI
- Usar PutFunctionConcurrency com o AWS SDK ou a CLI
- Usar PutProvisionedConcurrencyConfig com o AWS SDK ou a CLI
- Usar RemovePermission com o AWS SDK ou a CLI
- Usar TagResource com o AWS SDK ou a CLI
- Usar UntagResource com o AWS SDK ou a CLI
- Usar UpdateAlias com o AWS SDK ou a CLI
- Usar UpdateFunctionCode com o AWS SDK ou a CLI
- Usar UpdateFunctionConfiguration com o AWS SDK ou a CLI

- Cenários do Lambda usando AWS SDKs
  - Confirme automaticamente usuários conhecidos do Amazon Cognito com uma função do Lambda usando um AWS SDK
  - Migre automaticamente usuários conhecidos do Amazon Cognito com uma função do Lambda usando um AWS SDK
  - Começar a criar e invocar funções do Lambda usando um AWS SDK
  - Grave dados de atividades personalizados com uma função do Lambda após a autenticação do usuário do Amazon Cognito usando um AWS SDK
- Exemplos de tecnologia sem servidor para o Lambda usando SDKs da AWS
  - Como se conectar a um banco de dados do Amazon RDS em uma função do Lambda
  - Invocar uma função do Lambda em um trigger do Kinesis
  - Invocar uma função do Lambda em um gatilho do DynamoDB
  - Invocar uma função do Lambda de um acionador do Amazon DocumentDB
  - Invocar uma função do Lambda em um acionador do Amazon S3
  - Invocar uma função do Lambda em um acionador do Amazon SNS
  - Invocar uma função do Lambda em um trigger do Amazon SQS
  - Relatando falhas de itens em lote para funções do Lambda com um trigger do Kinesis
  - Relatar falhas de itens em lote para funções do Lambda com um gatilho do DynamoDB
  - Relatar falhas de itens em lote para funções do Lambda com um trigger do Amazon SQS
- Exemplos do Lambda entre serviços usando AWS SDKs
  - Criar uma API REST do API Gateway para monitorar dados da COVID-19

- Criar uma API REST de biblioteca de empréstimos
- Criar uma aplicação de mensageiro com o Step Functions
- Criar uma aplicação de gerenciamento de ativos de fotos que permita que os usuários gerenciem fotos usando rótulos
- Criar uma aplicação de chat websocket com o API Gateway
- Criar uma aplicação que analise o feedback dos clientes e sintetize o áudio
- Chamar uma função do Lambda em um navegador
- Como transformar dados para sua aplicação com o S3 Object Lambda
- Usar o API Gateway para invocar uma função do Lambda
- Usar Step Functions para invocar funções do Lambda
- Usar eventos programados para invocar uma função do Lambda

Ações do Lambda usando AWS SDKs

Os exemplos de código a seguir demonstram como realizar ações específicas do Lambda com AWS SDKs. Esses trechos chamam a API do Lambda e são trechos de código de programas maiores que devem ser executados no contexto. Cada exemplo inclui um link para o GitHub, onde você pode encontrar instruções para configurar e executar o código.

Os exemplos a seguir incluem apenas as ações mais utilizadas. Para obter uma lista completa, consulte a Referência de APIs do AWS Lambda.

Exemplos

- Usar CreateAlias com o AWS SDK ou a CLI
- Usar CreateFunction com o AWS SDK ou a CLI
- Usar DeleteAlias com o AWS SDK ou a CLI
- Usar DeleteFunction com o AWS SDK ou a CLI
- Usar DeleteFunctionConcurrency com o AWS SDK ou a CLI
- Usar DeleteProvisionedConcurrencyConfig com o AWS SDK ou a CLI
- Usar GetAccountSettings com o AWS SDK ou a CLI
- Usar GetAlias com o AWS SDK ou a CLI
- Usar GetFunction com o AWS SDK ou a CLI
- Usar GetFunctionConcurrency com o AWS SDK ou a CLI

- Usar GetFunctionConfiguration com o AWS SDK ou a CLI
- Usar GetPolicy com o AWS SDK ou a CLI
- Usar GetProvisionedConcurrencyConfig com o AWS SDK ou a CLI
- Usar Invoke com o AWS SDK ou a CLI
- Usar ListFunctions com o AWS SDK ou a CLI
- Usar ListProvisionedConcurrencyConfigs com o AWS SDK ou a CLI
- Usar ListTags com o AWS SDK ou a CLI
- Usar ListVersionsByFunction com o AWS SDK ou a CLI
- Usar PublishVersion com o AWS SDK ou a CLI
- Usar PutFunctionConcurrency com o AWS SDK ou a CLI
- Usar PutProvisionedConcurrencyConfig com o AWS SDK ou a CLI
- Usar RemovePermission com o AWS SDK ou a CLI
- Usar TagResource com o AWS SDK ou a CLI
- Usar UntagResource com o AWS SDK ou a CLI
- Usar UpdateAlias com o AWS SDK ou a CLI
- Usar UpdateFunctionCode com o AWS SDK ou a CLI
- Usar UpdateFunctionConfiguration com o AWS SDK ou a CLI

Usar **`CreateAlias`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `CreateAlias`.

CLI

AWS CLI

Para criar um alias para uma função do Lambda

O seguinte exemplo de `create-alias` cria um alias chamado `LIVE` que aponta para a versão 1 da função `my-function` do Lambda.

```
aws lambda create-alias \
    --function-name my-function \
    --description "alias for live version of function" \
    --function-version 1 \
```

```
    --name LIVE
```

Saída:

```
{
    "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"
}
```

Para obter mais informações, consulte Configurar aliases da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte CreateAlias na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo cria um novo alias do Lambda para versão e configuração de roteamento especificadas a fim de indicar o percentual de solicitações de invocação que ele receberá.

```
New-LMAlias -FunctionName "MylambdaFunction123" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6} -Description "Alias
 for version 4" -FunctionVersion 4 -Name "PowershellAlias"
```

- Para obter detalhes da API, consulte CreateAlias na Referência do Cmdlet das AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`CreateFunction`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `CreateFunction`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação no contexto no seguinte exemplo de código:

- Conceitos básicos de funções

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
    }
```

- Para obter detalhes da API, consulte CreateFunction na Referência da API AWS SDK for .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
        }
```

- Para obter detalhes da API, consulte CreateFunction na Referência da API AWS SDK for C++.

CLI

AWS CLI

Criar uma função do Lambda

O exemplo `create-function` a seguir cria uma função do Lambda denominada `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
```

Conteúdo de `my-function.zip`:

```
This file is a deployment package that contains your function code and any
 dependencies.
```

Saída:

```
{
    "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": ""
}
```

Para obter mais informações, consulte Configurar opções da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte CreateFunction em AWS CLI Command Reference.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
}
```

- Para obter detalhes da API, consulte CreateFunction na Referência da API AWS SDK for Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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);
        }
    }
}
```

- Para obter detalhes da API, consulte CreateFunction na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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);
};
```

- Para obter detalhes da API, consulte CreateFunction na Referência da API AWS SDK for JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
    }
}
```

- Para obter detalhes da API, consulte CreateFunction na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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",
            ]);
        });
    }
```

- Para obter detalhes da API, consulte CreateFunction na Referência da API AWS SDK for PHP.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo cria uma nova função em C# (runtime dotnetcore1.0) chamada MyFunction no AWS Lambda, fornecendo os binários compilados para a função com base em um arquivo zip no sistema local de arquivos (é possível usar caminhos relativos ou absolutos). As funções do Lambda em C# especificam o manipulador da função usando a designação AssemblyName::Namespace.ClassName::MethodName. É necessário substituir adequadamente o nome da montagem (sem o sufixo .dll), o namespace, o nome da classe e o nome do método da especificação do manipulador. A nova função terá as variáveis de ambiente “envvar1” e “envvar2” configuradas com base nos valores fornecidos.

```
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" }
```

Saída:

```
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        :
```

Exemplo 2: este exemplo é semelhante ao anterior, com a exceção de que os binários da função são carregados primeiramente em um bucket do Amazon S3 (que deve estar na mesma região da função do Lambda desejada) e o objeto resultante do S3 será referenciado ao criar a função.

```
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" }
```

- Para obter detalhes da API, consulte CreateFunction na Referência do Cmdlet das AWS Tools for PowerShell.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte CreateFunction na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte CreateFunction na Referência da API AWS SDK for Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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())
    }
```

- Para obter detalhes da API, consulte CreateFunction na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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.
```

- Para obter detalhes da API, consulte CreateFunction na Referência da API do AWS SDK para SAP ABAP.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`DeleteAlias`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `DeleteAlias`.

CLI

AWS CLI

Para excluir um alias de uma função do Lambda

O seguinte exemplo de `delete-alias` exclui o alias chamado `LIVE` da função `my-function` do Lambda.

```
aws lambda delete-alias \
    --function-name my-function \
    --name LIVE
```

Este comando não produz saída.

Para obter mais informações, consulte Configurar aliases da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte DeleteAlias na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo exclui o alias da função do Lambda mencionado no comando.

```
Remove-LMAlias -FunctionName "MylambdaFunction123" -Name "NewAlias"
```

- Para obter detalhes da API, consulte DeleteAlias na Referência do Cmdlet das AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`DeleteFunction`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `DeleteFunction`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação no contexto no seguinte exemplo de código:

- Conceitos básicos de funções

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
    }
```

- Para obter detalhes da API, consulte DeleteFunction na Referência da API AWS SDK for .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
    }
```

- Para obter detalhes da API, consulte DeleteFunction na Referência da API AWS SDK for C++.

CLI

AWS CLI

Exemplo 1: excluir uma função do Lambda pelo nome da função

O exemplo de `delete-function` a seguir exclui a função do Lambda denominada `my-function` ao especificar o nome da função.

```
aws lambda delete-function \
    --function-name my-function
```

Este comando não produz saída.

Exemplo 2: excluir uma função do Lambda pelo ARN da função

O exemplo `delete-function` a seguir exclui a função do Lambda denominada `my-function` ao especificar o ARN da função.

```
aws lambda delete-function \
    --function-name arn:aws:lambda:us-west-2:123456789012:function:my-function
```

Este comando não produz saída.

Exemplo 3: excluir uma função do Lambda pelo ARN parcial da função

O exemplo `delete-function` a seguir exclui a função do Lambda denominada `my-function` ao especificar o ARN parcial da função.

```
aws lambda delete-function \
    --function-name 123456789012:function:my-function
```

Este comando não produz saída.

Para obter mais informações, consulte Configurar opções da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte DeleteFunction em Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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)
 }
}
```

- Para obter detalhes da API, consulte DeleteFunction na Referência da API AWS SDK for Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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);
        }
    }
}
```

- Para obter detalhes da API, consulte DeleteFunction na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
/**
 * @param {string} funcName
```

```
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Para obter detalhes da API, consulte DeleteFunction na Referência da API AWS SDK for JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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")
    }
}
```

- Para obter detalhes da API, consulte DeleteFunction na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
    public function deleteFunction($functionName)
    {
        return $this->lambdaClient->deleteFunction([
            'FunctionName' => $functionName,
        ]);
    }
```

- Para obter detalhes da API, consulte DeleteFunction na Referência da API AWS SDK for PHP.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo exclui uma versão específica de uma função do Lambda

```
Remove-LMFunction -FunctionName "MylambdaFunction123" -Qualifier '3'
```

- Para obter detalhes da API, consulte DeleteFunction na Referência do Cmdlet das AWS Tools for PowerShell.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte DeleteFunction na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte DeleteFunction na Referência da API AWS SDK for Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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)
    }
```

- Para obter detalhes da API, consulte DeleteFunction na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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.
```

- Para obter detalhes da API, consulte DeleteFunction na Referência da API do AWS SDK para SAP ABAP.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`DeleteFunctionConcurrency`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `DeleteFunctionConcurrency`.

CLI

AWS CLI

Para remover o limite reservado de execução simultânea de uma função

O seguinte exemplo de `delete-function-concurrency` exclui o limite reservado de execução simultânea da função `my-function`.

```
aws lambda delete-function-concurrency \
    --function-name  my-function
```

Este comando não produz saída.

Para obter mais informações, consulte Como reservar simultaneidade para uma função do Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte DeleteFunctionConcurrency na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo remove a simultaneidade de função da função do Lambda.

```
Remove-LMFunctionConcurrency -FunctionName "MylambdaFunction123"
```

- Para obter detalhes da API, consulte DeleteFunctionConcurrency na Referência de Cmdlet da AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`DeleteProvisionedConcurrencyConfig`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `DeleteProvisionedConcurrencyConfig`.

CLI

AWS CLI

Para excluir uma configuração de simultaneidade provisionada

O seguinte exemplo de `delete-provisioned-concurrency-config` exclui a configuração de simultaneidade provisionada para o alias `GREEN` da função especificada.

```
aws lambda delete-provisioned-concurrency-config \
    --function-name my-function \
    --qualifier GREEN
```

- Para obter detalhes da API, consulte DeleteProvisionedConcurrencyConfig na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo remove a configuração de simultaneidade provisionada para um alias específico.

```
Remove-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
Qualifier "NewAlias1"
```

- Para obter detalhes da API, consulte DeleteProvisionedConcurrencyConfig na Referência de Cmdlet da AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`GetAccountSettings`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `GetAccountSettings`.

CLI

AWS CLI

Para recuperar detalhes sobre sua conta em uma região da AWS

O seguinte exemplo de `get-account-settings` mostra os limites do Lambda e as informações de uso da sua conta.

```
aws lambda get-account-settings
```

Saída:

```
{
    "AccountLimit": {
       "CodeSizeUnzipped": 262144000,
       "UnreservedConcurrentExecutions": 1000,
       "ConcurrentExecutions": 1000,
       "CodeSizeZipped": 52428800,
```

```
        "TotalCodeSize": 80530636800
    },
    "AccountUsage": {
        "FunctionCount": 4,
        "TotalCodeSize": 9426
    }
}
```

Para obter mais informações, consulte Limites do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte GetAccountSettings na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo é apresentado para comparar o limite da conta e o uso da conta

```
Get-LMAccountSetting | Select-Object
 @{Name="TotalCodeSizeLimit";Expression={$_.AccountLimit.TotalCodeSize}},
 @{Name="TotalCodeSizeUsed";Expression={$_.AccountUsage.TotalCodeSize}}
```

Saída:

```
TotalCodeSizeLimit TotalCodeSizeUsed
------------------ -----------------
       80530636800          15078795
```

- Para obter detalhes da API, consulte GetAccountSettings na Referência de Cmdlet da AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`GetAlias`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `GetAlias`.

CLI

AWS CLI

Para recuperar detalhes sobre um alias de função

O seguinte exemplo de `get-alias` mostra os detalhes do alias chamado `LIVE` na função `my-function` do Lambda.

```
aws lambda get-alias \
    --function-name my-function \
    --name LIVE
```

Saída:

```
{
    "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"
}
```

Para obter mais informações, consulte Configurar aliases da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte GetAlias na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo recupera os pesos da configuração de roteamento para o alias de uma função do Lambda específica.

```
Get-LMAlias -FunctionName "MylambdaFunction123" -Name "newlabel1" -Select
 RoutingConfig
```

Saída:

```
AdditionalVersionWeights
------------------------
{[1, 0.6]}
```

- Para obter detalhes da API, consulte GetAlias na Referência do Cmdlet das AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`GetFunction`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `GetFunction`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação no contexto no seguinte exemplo de código:

- Conceitos básicos de funções

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
    }
```

- Para obter detalhes da API, consulte GetFunction na Referência da API AWS SDK for .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
        }
```

- Para obter detalhes da API, consulte GetFunction na Referência da API AWS SDK for C++.

CLI

AWS CLI

Recuperar informações sobre uma função

O exemplo `get-function` a seguir mostra informações sobre a função `my-function`.

```
aws lambda get-function \
    --function-name  my-function
```

Saída:

```
{
    "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": ""
    }
}
```

Para obter mais informações, consulte Configurar opções da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte GetFunction na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
}
```

- Para obter detalhes da API, consulte GetFunction na Referência da API AWS SDK for Go.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Para obter detalhes da API, consulte GetFunction na Referência da API AWS SDK for JavaScript.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
    public function getFunction($functionName)
    {
        return $this->lambdaClient->getFunction([
            'FunctionName' => $functionName,
        ]);
    }
```

- Para obter detalhes da API, consulte GetFunction na Referência da API AWS SDK for PHP.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte GetFunction na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte GetFunction na Referência da API AWS SDK for Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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)
    }
```

- Para obter detalhes da API, consulte GetFunction na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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.
```

- Para obter detalhes da API, consulte GetFunction na Referência da API do AWS SDK para SAP ABAP.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`GetFunctionConcurrency`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `GetFunctionConcurrency`.

CLI

AWS CLI

Para visualizar a configuração de simultaneidade reservada para uma função

O seguinte exemplo de `get-function-concurrency` recupera a configuração de simultaneidade reservada para a função especificada.

```
aws lambda get-function-concurrency \
    --function-name my-function
```

Saída:

```
{
    "ReservedConcurrentExecutions": 250
}
```

- Para obter detalhes da API, consulte GetFunctionConcurrency na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo obtém a simultaneidade reservada para a função do Lambda

```
Get-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -Select *
```

Saída:

```
ReservedConcurrentExecutions
----------------------------
100
```

- Para obter detalhes da API, consulte GetFunctionConcurrency na Referência de Cmdlet da AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`GetFunctionConfiguration`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `GetFunctionConfiguration`.

CLI

AWS CLI

Para recuperar as configurações específicas da versão de uma função do Lambda

O seguinte exemplo de `get-function-configuration` mostra as configurações da versão 2 da função `my-function`.

```
aws lambda get-function-configuration \
    --function-name  my-function:2
```

Saída:

```
{
    "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"
}
```

Para obter mais informações, consulte Configurar opções da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte GetFunctionConfiguration na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo retorna a configuração específica de versão de uma função do Lambda.

```
Get-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Qualifier
 "PowershellAlias"
```

Saída:

```
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
```

- Para obter detalhes da API, consulte GetFunctionConfiguration na Referência de Cmdlet da AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`GetPolicy`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `GetPolicy`.

CLI

AWS CLI

Para recuperar a política do IAM baseada em recursos para uma função, versão ou alias

O seguinte exemplo de `get-policy` mostra informações de política sobre a função `my-function` do Lambda.

```
aws lambda get-policy \
    --function-name my-function
```

Saída:

```
{
    "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"
}
```

Para obter mais informações, consulte Como usar políticas baseadas em recursos para o AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte GetPolicy na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo mostra a política de função da função do Lambda

```
Get-LMPolicy -FunctionName test -Select Policy
```

Saída:

```
{"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"}]}
```

- Para obter detalhes da API, consulte GetPolicy na Referência do Cmdlet das AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **GetProvisionedConcurrencyConfig** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `GetProvisionedConcurrencyConfig`.

CLI

AWS CLI

Para visualizar uma configuração de simultaneidade provisionada

O seguinte exemplo de `get-provisioned-concurrency-config` mostra detalhes da configuração de simultaneidade provisionada para o alias `BLUE` da função especificada.

```
aws lambda get-provisioned-concurrency-config \
    --function-name my-function \
    --qualifier BLUE
```

Saída:

```
{
    "RequestedProvisionedConcurrentExecutions": 100,
    "AvailableProvisionedConcurrentExecutions": 100,
    "AllocatedProvisionedConcurrentExecutions": 100,
    "Status": "READY",
    "LastModified": "2019-12-31T20:28:49+0000"
}
```

- Para obter detalhes da API, consulte GetProvisionedConcurrencyConfig na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo obtém a configuração de simultaneidade provisionada para o alias especificado da função do Lambda.

```
C:\>Get-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
Qualifier "NewAlias1"
```

Saída:

```
AllocatedProvisionedConcurrentExecutions : 0
AvailableProvisionedConcurrentExecutions : 0
LastModified                             : 2020-01-15T03:21:26+0000
RequestedProvisionedConcurrentExecutions : 70
Status                                   : IN_PROGRESS
StatusReason                             :
```

- Para obter detalhes da API, consulte GetProvisionedConcurrencyConfig na Referência de Cmdlet da AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`Invoke`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `Invoke`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação no contexto no seguinte exemplo de código:

- Conceitos básicos de funções

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
   }
```

- Para obter detalhes da API, consulte Invoke, na Referência da API AWS SDK for .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
        }
```

- Para obter detalhes da API, consulte Invoke, na Referência da API AWS SDK for C++.

CLI

AWS CLI

Exemplo 1: invocar uma função do Lambda de forma síncrona

O exemplo de `invoke` a seguir invoca a função `my-function` de forma síncrona. A opção `cli-binary-format` será necessária se você estiver usando a AWS CLI versão 2. Para obter mais informações, consulte Opções de linha de comando globais compatíveis da AWS CLI no Guia do usuário da AWS Command Line Interface.

```
aws lambda invoke \
    --function-name my-function \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "name": "Bob" }' \
```

```
    response.json
```

Saída:

```
{
    "ExecutedVersion": "$LATEST",
    "StatusCode": 200
}
```

Para obter mais informações, consulte Invocação síncrona no Guia do desenvolvedor do AWS Lambda.

Exemplo 2: invocar uma função do Lambda de forma assíncrona

O exemplo `invoke` a seguir invoca a função `my-function` de forma assíncrona. A opção `cli-binary-format` será necessária se você estiver usando a AWS CLI versão 2. Para obter mais informações, consulte Opções de linha de comando globais compatíveis da AWS CLI no Guia do usuário da AWS Command Line Interface.

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

Saída:

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

Para obter mais informações, consulte Invocação assíncrona no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte Invocar na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
}
```

- Para obter detalhes da API, consulte Invoke, na Referência da API AWS SDK for Go.

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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);
        }
    }
}
```

- Para obter detalhes da API, consulte Invoke, na Referência da API AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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 };
};
```

- Para obter detalhes da API, consulte Invoke, na Referência da API AWS SDK for JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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}")
    }
}
```

- Para obter detalhes da API, consulte Invoke na Referência da API AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
    public function invoke($functionName, $params, $logType = 'None')
    {
        return $this->lambdaClient->invoke([
            'FunctionName' => $functionName,
            'Payload' => json_encode($params),
            'LogType' => $logType,
        ]);
    }
```

- Para obter detalhes da API, consulte Invoke, na Referência da API AWS SDK for PHP.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte Invoke, na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte Invoke, na Referência da API AWS SDK for Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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")
    }
}
```

- Para obter detalhes da API, consulte Invoke na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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.
```

- Para obter os detalhes da API, consulte Invoke na Referência da API do AWS SDK para SAP ABAP.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`ListFunctions`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `ListFunctions`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação no contexto no seguinte exemplo de código:

- Conceitos básicos de funções

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
}
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK for .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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());
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK for C++.

CLI

AWS CLI

Recuperar uma lista de funções do Lambda

O exemplo de `list-functions` a seguir exibe uma lista de todas as funções do usuário atual.

```
aws lambda list-functions
```

Saída:

```
{
    "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": ""
        }
    ]
}
```

Para obter mais informações, consulte Configurar opções da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte ListFunctions na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
}
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK for Go.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK for JavaScript.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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,
        ]);
    }
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK for PHP.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo mostra todas as funções do Lambda com tamanho de código classificado

```
Get-LMFunctionList | Sort-Object -Property CodeSize | Select-Object FunctionName,
 RunTime, Timeout, CodeSize
```

Saída:

```
FunctionName                                                   Runtime   Timeout
 CodeSize
------------                                                   -------   -------
 --------
test                                                           python2.7       3
   243
MylambdaFunction123                                            python3.8     600
   659
myfuncpython1                                                  python3.8     303
   675
```

- Para obter detalhes da API, consulte ListFunctions na Referência de Cmdlet do AWS Tools for PowerShell.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK for Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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)
   }
```

- Para obter detalhes da API, consulte ListFunctions na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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.
```

- Para obter detalhes da API, consulte ListFunctions na Referência de APIs do AWS SDK para SAP ABAP.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`ListProvisionedConcurrencyConfigs`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `ListProvisionedConcurrencyConfigs`.

CLI

AWS CLI

Para obter uma lista de configurações de simultaneidade provisionada

O seguinte exemplo de `list-provisioned-concurrency-configs` lista as configurações de simultaneidade provisionada para a função especificada.

```
aws lambda list-provisioned-concurrency-configs \
    --function-name my-function
```

Saída:

```
{
    "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"
        }
    ]
}
```

- Para obter detalhes da API, consulte ListProvisionedConcurrencyConfigs na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo recupera a lista de configurações de simultaneidade provisionada para uma função do Lambda.

```
Get-LMProvisionedConcurrencyConfigList -FunctionName "MylambdaFunction123"
```

- Para obter detalhes da API, consulte ListProvisionedConcurrencyConfigs na Referência de Cmdlet do AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`ListTags`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `ListTags`.

CLI

AWS CLI

Para recuperar a lista de tags para uma função do Lambda

O seguinte exemplo de `list-tags` mostra as tags anexadas à função `my-function` do Lambda.

```
aws lambda list-tags \
    --resource arn:aws:lambda:us-west-2:123456789012:function:my-function
```

Saída:

```
{
    "Tags": {
        "Category": "Web Tools",
        "Department": "Sales"
    }
}
```

Para obter mais informações, consulte Como marcar funções do Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte ListTags na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: recupera as tags e seus valores atualmente definidos na função especificada.

```
Get-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction"
```

Saída:

```
Key        Value
---        -----
California Sacramento
Oregon     Salem
```

```
Washington Olympia
```

- Para obter detalhes da API, consulte ListTags na Referência de Cmdlet do AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`ListVersionsByFunction`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `ListVersionsByFunction`.

CLI

AWS CLI

Para recuperar uma lista de versões de uma função

O seguinte exemplo de `list-versions-by-function` mostra a lista de versões da função `my-function` do Lambda.

```
aws lambda list-versions-by-function \
    --function-name my-function
```

Saída:

```
{
    "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"
        }
    ]
}
```

Para obter mais informações, consulte Configurar aliases da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte ListVersionsByFunction na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo retorna a lista de configurações específicas de versão para cada versão da função do Lambda.

```
Get-LMVersionsByFunction -FunctionName "MylambdaFunction123"
```

Saída:

```
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
```

- Para obter detalhes da API, consulte ListVersionsByFunction na Referência de Cmdlet do AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`PublishVersion`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `PublishVersion`.

CLI

AWS CLI

Para publicar uma nova versão de uma função

O seguinte exemplo de `publish-version` publica uma nova versão da função `my-function` do Lambda.

```
aws lambda publish-version \
    --function-name my-function
```

Saída:

```
{
    "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": ""
}
```

Para obter mais informações, consulte Configurar aliases da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte PublishVersion na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo cria uma versão para o snapshot existente do código da função do Lambda

```
Publish-LMVersion -FunctionName "MylambdaFunction123" -Description "Publishing
 Existing Snapshot of function code as a  new version through Powershell"
```

- Para obter detalhes da API, consulte PublishVersion na Referência de Cmdlet do AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`PutFunctionConcurrency`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `PutFunctionConcurrency`.

CLI

AWS CLI

Para configurar um limite de simultaneidade reservado para uma função

O seguinte exemplo de `put-function-concurrency` configura 100 execuções simultâneas reservadas para a função `my-function`.

```
aws lambda put-function-concurrency \
    --function-name  my-function  \
    --reserved-concurrent-executions 100
```

Saída:

```
{
    "ReservedConcurrentExecutions": 100
}
```

Para obter mais informações, consulte Como reservar simultaneidade para uma função do Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte PutFunctionConcurrency na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo aplica as configurações de simultaneidade para a função de maneira geral.

```
Write-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -
ReservedConcurrentExecution 100
```

- Para obter detalhes da API, consulte PutFunctionConcurrency na Referência de Cmdlet do AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`PutProvisionedConcurrencyConfig`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `PutProvisionedConcurrencyConfig`.

CLI

AWS CLI

Para alocar a simultaneidade provisionada

O seguinte exemplo de `put-provisioned-concurrency-config` aloca 100 simultaneidades provisionadas para o alias `BLUE` da função especificada.

```
aws lambda put-provisioned-concurrency-config \
    --function-name my-function \
    --qualifier BLUE \
    --provisioned-concurrent-executions 100
```

Saída:

```
{
    "Requested ProvisionedConcurrentExecutions": 100,
    "Allocated ProvisionedConcurrentExecutions": 0,
    "Status": "IN_PROGRESS",
    "LastModified": "2019-11-21T19:32:12+0000"
}
```

- Para obter detalhes da API, consulte PutProvisionedConcurrencyConfig na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo adiciona uma configuração de simultaneidade provisionada ao alias de uma função

```
Write-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
ProvisionedConcurrentExecution 20 -Qualifier "NewAlias1"
```

- Para obter detalhes da API, consulte PutProvisionedConcurrencyConfig na Referência de Cmdlet do AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`RemovePermission`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `RemovePermission`.

CLI

AWS CLI

Para remover permissões de uma função do Lambda existente

O seguinte exemplo de `remove-permission` remove permissão para invocar uma função chamada `my-function`.

```
aws lambda remove-permission \
    --function-name my-function \
    --statement-id sns
```

Este comando não produz saída.

Para obter mais informações, consulte Como usar políticas baseadas em recursos para o AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte RemovePermission na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo remove a política de função para o StatementId especificado de uma função do Lambda.

```
$policy =  Get-LMPolicy -FunctionName "MylambdaFunction123" -Select Policy |
 ConvertFrom-Json| Select-Object -ExpandProperty Statement
Remove-LMPermission -FunctionName "MylambdaFunction123" -StatementId
 $policy[0].Sid
```

- Para obter detalhes da API, consulte RemovePermission na Referência de Cmdlet do AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`TagResource`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `TagResource`.

CLI

AWS CLI

Para adicionar tags a uma função do Lambda existente

O seguinte exemplo de `tag-resource` adiciona uma tag com o nome de chave `DEPARTMENT` e um valor de `Department A` à função do Lambda especificada.

```
aws lambda tag-resource \
    --resource arn:aws:lambda:us-west-2:123456789012:function:my-function \
    --tags "DEPARTMENT=Department A"
```

Este comando não produz saída.

Para obter mais informações, consulte Como marcar funções do Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte TagResource na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: adiciona as três tags (Washington, Oregon e Califórnia) e seus valores associados à função especificada identificada por seu ARN.

```
Add-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction" -Tag @{ "Washington" = "Olympia";
 "Oregon" = "Salem"; "California" = "Sacramento" }
```

- Para obter detalhes da API, consulte TagResource na Referência de Cmdlet do AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`UntagResource`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `UntagResource`.

CLI

AWS CLI

Para remover tags de uma função do Lambda existente

O seguinte exemplo de `untag-resource` remove a tag com o nome de chave `DEPARTMENT` da função `my-function` do Lambda.

```
aws lambda untag-resource \
    --resource arn:aws:lambda:us-west-2:123456789012:function:my-function \
    --tag-keys DEPARTMENT
```

Este comando não produz saída.

Para obter mais informações, consulte Como marcar funções do Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes sobre a API, consulte UntagResource na AWS CLI Command Reference.

PowerShell

Tools for PowerShell

Exemplo 1: remove as tags fornecidas de uma função. A menos que a opção -Force esteja especificada, o cmdlet solicitará a confirmação antes de continuar. Uma única chamada será feita para o serviço a fim de remover as tags.

```
Remove-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction" -TagKey
 "Washington","Oregon","California"
```

Exemplo 2: remove as tags fornecidas de uma função. A menos que a opção -Force esteja especificada, o cmdlet solicitará a confirmação antes de continuar. Isso acontece porque a chamada para o serviço é feita pela tag fornecida.

```
"Washington","Oregon","California" | Remove-LMResourceTag -Resource
 "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
```

- Para obter detalhes da API, consulte UntagResource na Referência do Cmdlet das AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`UpdateAlias`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `UpdateAlias`.

CLI

AWS CLI

Para atualizar um alias de função

O seguinte exemplo de `update-alias` atualiza um alias chamado `LIVE` que aponta para a versão 3 da função `my-function` do Lambda.

```
aws lambda update-alias \
```

```
    --function-name my-function \
    --function-version 3 \
    --name LIVE
```

Saída:

```
{
    "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"
}
```

Para obter mais informações, consulte Configurar aliases da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte UpdateAlias na Referência de comandos da AWS CLI.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo atualiza a configuração de um alias de função do Lambda existente. Ele atualiza o valor da configuração de RoutingConfiguration para transferir 60% (0,6) do tráfego para a versão 1

```
Update-LMAlias -FunctionName "MylambdaFunction123" -Description
 " Alias for version 2" -FunctionVersion 2 -Name "newlabel1" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6}
```

- Para obter detalhes da API, consulte UpdateAlias na Referência de Cmdlet do AWS Tools for PowerShell.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **UpdateFunctionCode** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `UpdateFunctionCode`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação no contexto no seguinte exemplo de código:

- Conceitos básicos de funções

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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}.");
    }
```

- Para obter detalhes sobre a API, consulte UpdateFunctionCode na Referência da API AWS SDK for .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
        }
```

- Para obter detalhes sobre a API, consulte UpdateFunctionCode na Referência da API AWS SDK for C++.

CLI

AWS CLI

Atualizar o código de uma função do Lambda

O exemplo de `update-function-code` a seguir substitui o código da versão não publicada ($LATEST) da função `my-function` pelo conteúdo do arquivo zip especificado.

```
aws lambda update-function-code \
    --function-name  my-function \
```

```
    --zip-file fileb://my-function.zip
```

Saída:

```
{
    "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"
}
```

Para obter mais informações, consulte Configurar opções da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte UpdateFunctionCode na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
}
```

- Para obter detalhes sobre a API, consulte UpdateFunctionCode na Referência da API AWS SDK for Go.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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);
};
```

- Para obter detalhes sobre a API, consulte UpdateFunctionCode na Referência da API AWS SDK for JavaScript.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
public function updateFunctionCode($functionName, $s3Bucket, $s3Key)
{
    return $this->lambdaClient->updateFunctionCode([
        'FunctionName' => $functionName,
        'S3Bucket' => $s3Bucket,
        'S3Key' => $s3Key,
    ]);
}
```

- Para obter detalhes sobre a API, consulte UpdateFunctionCode na Referência da API AWS SDK for PHP.

PowerShell

Tools for PowerShell

Exemplo 1: atualiza a função chamada “MyFunction” com o novo conteúdo do arquivo zip especificado. Para uma função do Lambda em C# .NET Core, o arquivo zip deve conter a montagem compilada.

```
Update-LMFunctionCode -FunctionName MyFunction -ZipFilename .\UpdatedCode.zip
```

Exemplo 2: este exemplo é semelhante ao anterior, mas usa um objeto do Amazon S3 contendo o código atualizado para atualizar a função.

```
Update-LMFunctionCode -FunctionName MyFunction -BucketName mybucket -Key
 UpdatedCode.zip
```

- Para obter detalhes da API, consulte UpdateFunctionCode na Referência de Cmdlet do AWS Tools for PowerShell.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte UpdateFunctionCode na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes sobre a API, consulte UpdateFunctionCode na Referência da API AWS SDK for Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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())
    }
```

- Para obter detalhes da API, consulte UpdateFunctionCode na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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.
```

- Para obter os detalhes da API, consulte UpdateFunctionCode na Referência da API do AWS SDK para SAP ABAP.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar **`UpdateFunctionConfiguration`** com o AWS SDK ou a CLI

Os exemplos de código a seguir mostram como usar o `UpdateFunctionConfiguration`.

Exemplos de ações são trechos de código de programas maiores e devem ser executados em contexto. É possível ver essa ação no contexto no seguinte exemplo de código:

- Conceitos básicos de funções

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
    }
```

- Para obter detalhes da API, consulte UpdateFunctionConfiguration na Referência da API AWS SDK for .NET.

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
        }
```

- Para obter detalhes da API, consulte UpdateFunctionConfiguration na Referência da API AWS SDK for C++.

CLI

AWS CLI

Modificar a configuração de uma função

O exemplo de `update-function-configuration` a seguir modifica o tamanho da memória para 256 MB para a versão não publicada ($LATEST) da função `my-function`.

```
aws lambda update-function-configuration \
```

```
    --function-name  my-function \
    --memory-size 256
```

Saída:

```
{
    "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"
}
```

Para obter mais informações, consulte Configurar opções da função do AWS Lambda no Guia do desenvolvedor do AWS Lambda.

- Para obter detalhes da API, consulte UpdateFunctionConfiguration na Referência de comandos da AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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)
 }
}
```

- Para obter detalhes da API, consulte UpdateFunctionConfiguration na Referência da API AWS SDK for Go.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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);
};
```

- Para obter detalhes da API, consulte UpdateFunctionConfiguration na Referência da API AWS SDK for JavaScript.

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

```
    public function updateFunctionConfiguration($functionName, $handler,
 $environment = '')
    {
        return $this->lambdaClient->updateFunctionConfiguration([
            'FunctionName' => $functionName,
            'Handler' => "$handler.lambda_handler",
```

```
            'Environment' => $environment,
        ]);
    }
```

- Para obter detalhes da API, consulte UpdateFunctionConfiguration na Referência da API AWS SDK for PHP.

PowerShell

Tools for PowerShell

Exemplo 1: este exemplo atualiza a configuração da função do Lambda existente

```
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
```

- Para obter detalhes da API, consulte UpdateFunctionConfiguration na Referência de Cmdlet do AWS Tools for PowerShell.

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte UpdateFunctionConfiguration na Referência da API AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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
```

- Para obter detalhes da API, consulte UpdateFunctionConfiguration na Referência da API AWS SDK for Ruby.

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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)
}
```

- Para obter detalhes da API, consulte UpdateFunctionConfiguration na Referência da API AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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.
```

- Para obter os detalhes da API, consulte UpdateFunctionConfiguration na Referência da API do AWS SDK para SAP ABAP.

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Cenários do Lambda usando AWS SDKs

Os exemplos de código a seguir mostram como implementar cenários comuns no Lambda com AWS SDKs. Esses cenários mostram como realizar tarefas específicas chamando várias funções no Lambda. Cada exemplo inclui um link para o GitHub, em que é possível encontrar instruções sobre como configurar e executar o código.

Exemplos

- Confirme automaticamente usuários conhecidos do Amazon Cognito com uma função do Lambda usando um AWS SDK
- Migre automaticamente usuários conhecidos do Amazon Cognito com uma função do Lambda usando um AWS SDK
- Começar a criar e invocar funções do Lambda usando um AWS SDK
- Grave dados de atividades personalizados com uma função do Lambda após a autenticação do usuário do Amazon Cognito usando um AWS SDK

Confirme automaticamente usuários conhecidos do Amazon Cognito com uma função do Lambda usando um AWS SDK

O exemplo de código a seguir mostra como confirmar automaticamente usuários conhecidas do Amazon Cognito com uma função do Lambda.

- Configure um grupo de usuários para chamar uma função do Lambda para o acionador `PreSignUp`.
- Inscreva-se para ser um usuário no Amazon Cognito.
- A função do Lambda verifica uma tabela do DynamoDB e confirma automaticamente os usuários conhecidos.
- Faça login como o novo usuário e, em seguida, limpe os recursos.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Execute um cenário interativo em um prompt de comando.

```
// 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))
}
```

Aborde o acionador `PreSignUp` com uma função do 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)
}
```

Crie uma struct que realize tarefas comuns.

```
// 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))
}
```

Crie uma struct que encapsule ações do 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
}
```

Crie uma struct que encapsule ações do 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
}
```

Crie uma struct que encapsule ações do CloudWatch Logs.

```
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
}
```

Crie uma struct que encapsule ações do 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
}
```

Limpar recursos.

```
// 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!")
 }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Go.
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Migre automaticamente usuários conhecidos do Amazon Cognito com uma função do Lambda usando um AWS SDK

O exemplo de código a seguir mostra como migrar automaticamente usuários conhecidas do Amazon Cognito com uma função do Lambda.

- Configure um grupo de usuários para chamar uma função do Lambda para o acionador `MigrateUser`.
- Faça login no Amazon Cognito com um nome de usuário e e-mail que não estejam no grupo de usuários.
- A função do Lambda verifica uma tabela do DynamoDB e migra automaticamente os usuários conhecidos para o grupo de usuários.
- Realize um fluxo de senha esquecida para redefinir a senha para o usuário migrado.
- Faça login como o novo usuário e, em seguida, limpe os recursos.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Execute um cenário interativo em um prompt de comando.

```
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))
}
```

Aborde o acionador `MigrateUser` com uma função do 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)
}
```

Crie uma struct que realize tarefas comuns.

```
// 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))
}
```

Crie uma struct que encapsule ações do 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
}
```

Crie uma struct que encapsule ações do 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
}
```

Crie uma struct que encapsule ações do CloudWatch Logs.

```
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
}
```

Crie uma struct que encapsule ações do 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
}
```

Limpar recursos.

```
// 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!")
 }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Go.
  - ConfirmForgotPassword
  - DeleteUser
  - ForgotPassword
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Começar a criar e invocar funções do Lambda usando um AWS SDK

Os exemplos de código a seguir mostram como:

- Crie um perfil do IAM e uma função do Lambda e carregue o código de manipulador.
- Invocar essa função com um único parâmetro e receber resultados.
- Atualizar o código de função e configurar usando uma variável de ambiente.
- Invocar a função com novos parâmetros e receber resultados. Exibir o log de execução retornado.
- Liste as funções para sua conta e limpe os recursos.

Para obter mais informações, consulte Criar uma função do Lambda no console.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie métodos que executem ações do 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;
    }


}
```

Crie uma função que execute o cenário.

```
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();
    }
}
```

Defina um manipulador do Lambda que aumente um número.

```
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;
        }
    }
}
```

Defina um segundo manipulador do Lambda que faça operações aritméticas.

```
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;
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for .NET.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

C++

SDK para C++

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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;
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for C++.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie um cenário interativo para mostrar como começar a usar funções do 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.")
 }
}
```

Crie uma estrutura que encapsule ações individuais do 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"`
}
```

Crie uma estrutura que implemente funções para ajudar a executar o cenário.

```
// 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
}
```

Defina um manipulador do Lambda que aumente um número.

```
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
```

Defina um segundo manipulador do Lambda que faça operações aritméticas.

```
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
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Go.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Java

SDK para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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);
        }
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Java 2.x.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Crie um perfil do AWS Identity and Access Management (IAM) que concede permissão ao Lambda para gravar em logs.

```
    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);
};
```

Crie uma função do Lambda e carregue o código de manipulador.

```
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);
};
```

Invoque essa função com um único parâmetro e receba resultados.

```
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 };
};
```

Atualize o código de função e configure seu ambiente do Lambda usando uma variável de 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);
};
```

Liste as funções para a sua conta.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

Exclua o perfil do IAM e a função do 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);
};
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for JavaScript.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Kotlin

SDK para Kotlin

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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")
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para Kotlin.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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";
    }
```

```
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for PHP.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Python

SDK para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Defina um manipulador do Lambda que aumente um número.

```
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
```

Defina um segundo manipulador do Lambda que faça operações aritméticas.

```
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
```

Crie funções que envolvam ações do 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
```

Crie uma função que execute o cenário.

```
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!")
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para Python (Boto3).
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode

- UpdateFunctionConfiguration

Ruby

SDK para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Configure as permissões de pré-requisitos do IAM para uma função do Lambda capaz de gravar logs.

```
# 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
```

Defina um manipulador do Lambda que incremente um número fornecido como um parâmetro de invocação.

```
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
```

Compacte a função do Lambda em um pacote de implantação.

```
  # 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
```

Crie uma nova função do 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
```

Invoque a função do Lambda com parâmetros de runtime opcionais.

```
 # 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
```

Atualize a configuração da função do Lambda para injetar uma nova variável de 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
```

Atualize o código da função do Lambda com um pacote de implantação diferente que contenha um código diferente.

```
  # 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
```

Liste todas as funções do Lambda existentes usando o paginador integrado.

```
  # 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
```

Exclua uma função do Lambda específica.

```
  # 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
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Ruby.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Rust

SDK para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

O Cargo.toml com dependências usadas neste cenário.

```
[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"
```

Uma coleção de utilitários que simplificam chamar o Lambda para este cenário. Este arquivo está como src/ations.rs no crate.

```
// 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(),
        );
    }
}
```

Um binário para executar o cenário do início ao fim usando sinalizadores de linha de comando para controlar algum comportamento. Este arquivo está como src/bin/scenario.rs no crate.

```
// 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");
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para Rust.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da 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.
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para SAP ABAP.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Grave dados de atividades personalizados com uma função do Lambda após a autenticação do usuário do Amazon Cognito usando um AWS SDK

O exemplo de código a seguir mostra como gravar dados de atividade personalizados com uma função do Lambda depois da autenticação do usuário do Amazon Cognito.

- Use as funções de administrador para adicionar um usuário a um grupo de usuários.
- Configure um grupo de usuários para chamar uma função do Lambda para o acionador `PostAuthentication`.
- Faça login do novo usuário no Amazon Cognito.
- A função do Lambda grava informações personalizadas no CloudWatch Logs e em uma tabela do DynamoDB.
- Obtenha e veja dados personalizados da tabela do DynamoDB e, em seguida, limpe os recursos.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no Repositório de exemplos de código da AWS.

Execute um cenário interativo em um prompt de comando.

```
// 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))
}
```

Aborde o acionador `PostAuthentication` com uma função do 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)
}
```

Crie uma struct que realize tarefas comuns.

```
// 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))
}
```

Crie uma struct que encapsule ações do 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
}
```

Crie uma struct que encapsule ações do 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
}
```

Crie uma struct que encapsule ações do CloudWatch Logs.

```
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
}
```

Crie uma struct que encapsule ações do 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
}
```

Limpar recursos.

```
// 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!")
 }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK for Go.
  - AdminCreateUser
  - AdminSetUserPassword
  - DeleteUser
  - InitiateAuth
  - UpdateUserPool

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Exemplos de tecnologia sem servidor para o Lambda usando SDKs da AWS

Os exemplos de código a seguir mostram como usar o Lambda com SDKs da AWS.

Exemplos

- Como se conectar a um banco de dados do Amazon RDS em uma função do Lambda
- Invocar uma função do Lambda em um trigger do Kinesis
- Invocar uma função do Lambda em um gatilho do DynamoDB
- Invocar uma função do Lambda de um acionador do Amazon DocumentDB
- Invocar uma função do Lambda em um acionador do Amazon S3
- Invocar uma função do Lambda em um acionador do Amazon SNS
- Invocar uma função do Lambda em um trigger do Amazon SQS
- Relatando falhas de itens em lote para funções do Lambda com um trigger do Kinesis
- Relatar falhas de itens em lote para funções do Lambda com um gatilho do DynamoDB
- Relatar falhas de itens em lote para funções do Lambda com um trigger do Amazon SQS

Como se conectar a um banco de dados do Amazon RDS em uma função do Lambda

Os exemplos de código a seguir mostram como implementar uma função do Lambda que se conecte a um banco de dados do RDS. A função faz uma solicitação simples ao banco de dados e exibe o resultado.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Conectar-se a um banco de dados do Amazon RDS em uma função do Lambda usando 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 para JavaScript (v2)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Conectar-se a um banco de dados do Amazon RDS em uma função do Lambda usando 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)
  }
};
```

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Invocar uma função do Lambda em um trigger do Kinesis

Os exemplos de código a seguir mostram como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de mensagens de stream do Kinesis. A função recupera a carga útil do Kinesis, decodifica do Base64 e registra o conteúdo do registro em log.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando 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;
}
```

Consumir um evento do Kinesis com o Lambda usando 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 para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Kinesis com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consuma um evento do Kinesis com o Lambda usando 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
}
```

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Invocar uma função do Lambda em um gatilho do DynamoDB

Os exemplos de código a seguir mostram como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de registros de um fluxo do DynamoDB. A função recupera a carga útil do DynamoDB e registra em log o conteúdo do registro.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do DynamoDB com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do DynamoDB com o Lambda usando 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)}`);
};
```

Consumir um evento do DynamoDB com o Lambda usando 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 para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do DynamoDB com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do DynamoDB com o Lambda usando 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(())

}
```

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Invocar uma função do Lambda de um acionador do Amazon DocumentDB

Os exemplos de código a seguir mostram como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de registros de um fluxo de alterações do DocumentDB. A função recupera a carga útil do DocumentDB e registra em log o conteúdo do registro.

Go

SDK para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Amazon DocumentDB com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Amazon DocumentDB com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Amazon DocumentDB com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do Amazon DocumentDB com o Lambda usando 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
```

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Invocar uma função do Lambda em um acionador do Amazon S3

Os exemplos de código a seguir mostram como implementar uma função do Lambda que recebe um evento acionado pelo upload de um objeto em um bucket do S3. A função recupera o nome do bucket do S3 e a chave do objeto do parâmetro de evento e chama a API do Amazon S3 para recuperar e registrar em log o tipo de conteúdo do objeto.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do S3 com o Lambda usando .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do S3 com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do S3 com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do S3 com o Lambda usando JavaScript.

```
// 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);
    }
};
```

Consumir um evento do S3 com o Lambda usando TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure
 they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do S3 com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do S3 com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como consumir um evento do S3 com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do S3 com o Lambda usando 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(())
}
```

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Invocar uma função do Lambda em um acionador do Amazon SNS

Os exemplos de código a seguir mostram como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de mensagens de um tópico do SNS. A função recupera as mensagens do parâmetro event e registra o conteúdo de cada mensagem.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório de Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consumir um evento do SNS com o Lambda usando TypeScript.

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

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
```

```
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SNS com o Lambda usando 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
}
```

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Invocar uma função do Lambda em um trigger do Amazon SQS

Os exemplos de código a seguir mostram como implementar uma função do Lambda que recebe um evento acionado pelo recebimento de mensagens de uma fila do SQS. A função recupera as mensagens do parâmetro event e registra o conteúdo de cada mensagem.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Consumir um evento do SQS com o Lambda usando TypeScript.

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

```
export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

SDK para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consumir um evento do SQS com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Consuma um evento do SQS com o Lambda usando 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
}
```

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Relatando falhas de itens em lote para funções do Lambda com um trigger do Kinesis

Os exemplos de código a seguir mostram como implementar uma resposta parcial em lote para funções do Lambda que recebem eventos de um stream do Kinesis. A função relata as falhas do item em lote na resposta, sinalizando para o Lambda tentar novamente essas mensagens posteriormente.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do Kinesis com o Lambda usando o .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do Kinesis com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do Kinesis com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do Kinesis com o Lambda usando 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;
}
```

Relatar falhas de itens em lote do Kinesis com o Lambda usando 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 para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do Kinesis com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do Kinesis com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de item em lote do Kinesis com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relate falhas de itens em lote do Kinesis com o Lambda usando 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
}
```

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Relatar falhas de itens em lote para funções do Lambda com um gatilho do DynamoDB

Os exemplos de código a seguir mostram como implementar uma resposta parcial em lote para funções do Lambda que recebem eventos de um fluxo do DynamoDB. A função relata as falhas do item em lote na resposta, sinalizando para o Lambda tentar novamente essas mensagens posteriormente.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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: [] };
};
```

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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 para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do DynamoDB com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Como relatar falhas de itens em lote do DynamoDB com o Lambda usando 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
}
```

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Relatar falhas de itens em lote para funções do Lambda com um trigger do Amazon SQS

Os exemplos de código a seguir mostram como implementar uma resposta parcial em lote para funções do Lambda que recebem eventos de uma fila do SQS. A função relata as falhas do item em lote na resposta, sinalizando para o Lambda tentar novamente essas mensagens posteriormente.

.NET

AWS SDK for .NET

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando o .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 para Go V2

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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 para Java 2.x

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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 para JavaScript (v3)

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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}`);
}
```

Relatar falhas de item em lote do SQS com o Lambda usando 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 para PHP

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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 para Python (Boto3).

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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 para Ruby

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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 para Rust

> **Note**
>
> Há mais no GitHub. Encontre o exemplo completo e saiba como configurar e executar no repositório dos Exemplos sem servidor.

Relatar falhas de itens em lote do SQS com o Lambda usando 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
}
```

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Exemplos do Lambda entre serviços usando AWS SDKs

As amostras de aplicações a seguir usam AWS SDKs para combinar o Lambda com outros Serviços da AWS. Cada exemplo inclui um link para o GitHub, em que é possível encontrar instruções sobre como configurar e executar a aplicação.

Exemplos

- Criar uma API REST do API Gateway para monitorar dados da COVID-19
- Criar uma API REST de biblioteca de empréstimos
- Criar uma aplicação de mensageiro com o Step Functions
- Criar uma aplicação de gerenciamento de ativos de fotos que permita que os usuários gerenciem fotos usando rótulos
- Criar uma aplicação de chat websocket com o API Gateway
- Criar uma aplicação que analise o feedback dos clientes e sintetize o áudio
- Chamar uma função do Lambda em um navegador
- Como transformar dados para sua aplicação com o S3 Object Lambda

- Usar o API Gateway para invocar uma função do Lambda
- Usar Step Functions para invocar funções do Lambda
- Usar eventos programados para invocar uma função do Lambda

Criar uma API REST do API Gateway para monitorar dados da COVID-19

O exemplo de código a seguir mostra como criar uma API REST que simula um sistema para monitorar casos diários de COVID-19 nos Estados Unidos, usando dados fictícios.

Python

SDK para Python (Boto3)

Mostra como usar o AWS Chalice com o AWS SDK for Python (Boto3) para criar uma API REST sem servidor que usa o Amazon API Gateway, o AWS Lambda e o Amazon DynamoDB. A API REST simula um sistema que monitora casos diários de COVID-19 nos Estados Unidos, usando dados fictícios. Aprenda como:

- Usar o AWS Chalice para definir rotas nas funções do Lambda que são chamadas para lidar com solicitações REST provenientes do API Gateway.
- Usar as funções do Lambda para recuperar e armazenar dados em uma tabela do DynamoDB para atender a solicitações REST.
- Definir recursos de função de segurança e estrutura de tabela em um modelo do AWS CloudFormation.
- Usar o AWS Chalice e o CloudFormation para empacotar e implantar todos os recursos necessários.
- Usar o CloudFormation para limpar todos os recursos criados.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- AWS CloudFormation
- DynamoDB
- Lambda

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar uma API REST de biblioteca de empréstimos

O exemplo de código abaixo mostra como criar uma biblioteca de empréstimos na qual os clientes possam pegar e devolver livros emprestados usando uma API REST com suporte por um banco de dados do Amazon Aurora.

Python

SDK para Python (Boto3)

Mostra como usar o AWS SDK for Python (Boto3) com a API do Amazon Relational Database Service (Amazon RDS) e o AWS Chalice a fim de criar uma API REST com suporte por um banco de dados do Amazon Aurora. O serviço da Web é uma tecnologia sem servidor e representa uma biblioteca de empréstimos simples, na qual os clientes podem pegar e devolver livros emprestados. Aprenda como:

- Crie e gerencie um cluster de banco de dados Aurora com tecnologia sem servidor.
- Utilize o AWS Secrets Manager para gerenciar credenciais de bancos de dados.
- Implemente uma camada de armazenamento de dados que use o Amazon RDS para mover dados para dentro e fora do banco de dados.
- Use o AWS Chalice para implantar uma API REST com tecnologia sem servidor no Amazon API Gateway e no AWS Lambda.
- Use o pacote Requests para enviar solicitações ao serviço Web.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- Aurora
- Lambda
- Secrets Manager

Para ver uma lista completa dos Guias do desenvolvedor de SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar uma aplicação de mensageiro com o Step Functions

O exemplo de código a seguir mostra como criar uma aplicação de mensageiro do AWS Step Functions que recupera registros de mensagens de uma tabela de banco de dados.

Python

SDK para Python (Boto3)

Mostra como usar o AWS SDK for Python (Boto3) com o AWS Step Functions para criar uma aplicação de mensageiro que recupere registros de mensagens de uma tabela do Amazon DynamoDB e os envia com o Amazon Simple Queue Service (Amazon SQS). A máquina de estado se integra a uma função do AWS Lambda para verificar o banco de dados em busca de mensagens não enviadas.

- Crie uma máquina de estado que recupere e atualize registros de mensagens de uma tabela do Amazon DynamoDB.
- Atualize a definição de máquina de estado para enviar mensagens ao Amazon Simple Queue Service (Amazon SQS).
- Inicie e interrompa execuções da máquina de estado.
- Conecte-se ao Lambda, ao DynamoDB e ao Amazon SQS por meio de uma máquina de estado usando integrações de serviço.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar uma aplicação de gerenciamento de ativos de fotos que permita que os usuários gerenciem fotos usando rótulos

O exemplo de código a seguir mostra como criar uma aplicação com tecnologia sem servidor que permite que os usuários gerenciem fotos usando rótulos.

.NET

AWS SDK for .NET

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK para C++

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK para Java 2.x

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway

- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK para JavaScript (v3)

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDK para Kotlin

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK para PHP

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK para Rust

Mostra como desenvolver uma aplicação de gerenciamento de ativos fotográficos que detecta rótulos em imagens usando o Amazon Rekognition e os armazena para recuperação posterior.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Para uma análise detalhada da origem desse exemplo, veja a publicação na Comunidade da AWS.

Serviços utilizados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar uma aplicação de chat websocket com o API Gateway

O exemplo de código a seguir mostra como criar uma aplicação de chat que é atendido por uma API de Websocket criada no Amazon API Gateway.

Python

SDK para Python (Boto3)

Mostra como usar o AWS SDK for Python (Boto3) com o Amazon API Gateway V2 para criar uma API de Websocket que se integra ao AWS Lambda e ao Amazon DynamoDB.

- Crie uma API de Websocket atendida pelo API Gateway.
- Defina um manipulador do Lambda que armazena conexões no DynamoDB e publica mensagens para outros participantes do chat.
- Conecte-se à aplicação de chat websocket e envie mensagens com o pacote Websockets.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Criar uma aplicação que analise o feedback dos clientes e sintetize o áudio

Os exemplos de código a seguir mostram como criar uma aplicação que analisa os cartões de comentários dos clientes, os traduz do idioma original, determina seus sentimentos e gera um arquivo de áudio do texto traduzido.

.NET

AWS SDK for .NET

Esta aplicação de exemplo analisa e armazena cartões de feedback de clientes. Especificamente, ela atende à necessidade de um hotel fictício na cidade de Nova York. O hotel recebe feedback dos hóspedes em vários idiomas na forma de cartões de comentários físicos. Esse feedback é enviado para a aplicação por meio de um cliente web. Depois de fazer upload da imagem de um cartão de comentário, ocorrem as seguintes etapas:

- O texto é extraído da imagem usando o Amazon Textract.
- O Amazon Comprehend determina o sentimento do texto extraído e o idioma.
- O texto extraído é traduzido para o inglês com o Amazon Translate.
- O Amazon Polly sintetiza um arquivo de áudio do texto extraído.

A aplicação completa pode ser implantada com o AWS CDK. Para obter o código-fonte e as instruções de implantação, consulte o projeto em GitHub.

Serviços utilizados neste exemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Java

SDK para Java 2.x

Esta aplicação de exemplo analisa e armazena cartões de feedback de clientes. Especificamente, ela atende à necessidade de um hotel fictício na cidade de Nova York. O hotel recebe feedback dos hóspedes em vários idiomas na forma de cartões de comentários físicos. Esse feedback é enviado para a aplicação por meio de um cliente web. Depois de fazer upload da imagem de um cartão de comentário, ocorrem as seguintes etapas:

- O texto é extraído da imagem usando o Amazon Textract.
- O Amazon Comprehend determina o sentimento do texto extraído e o idioma.
- O texto extraído é traduzido para o inglês com o Amazon Translate.
- O Amazon Polly sintetiza um arquivo de áudio do texto extraído.

A aplicação completa pode ser implantada com o AWS CDK. Para obter o código-fonte e as instruções de implantação, consulte o projeto em GitHub.

Serviços utilizados neste exemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

JavaScript

SDK para JavaScript (v3)

Esta aplicação de exemplo analisa e armazena cartões de feedback de clientes. Especificamente, ela atende à necessidade de um hotel fictício na cidade de Nova York. O hotel recebe feedback dos hóspedes em vários idiomas na forma de cartões de comentários físicos. Esse feedback é enviado para a aplicação por meio de um cliente web. Depois de fazer upload da imagem de um cartão de comentário, ocorrem as seguintes etapas:

- O texto é extraído da imagem usando o Amazon Textract.
- O Amazon Comprehend determina o sentimento do texto extraído e o idioma.
- O texto extraído é traduzido para o inglês com o Amazon Translate.
- O Amazon Polly sintetiza um arquivo de áudio do texto extraído.

A aplicação completa pode ser implantada com o AWS CDK. Para obter o código-fonte e as instruções de implantação, consulte o projeto em GitHub. Os trechos a seguir mostram como o AWS SDK for JavaScript é usado nas funções do 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 };
};
```

Serviços utilizados neste exemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Ruby

SDK para Ruby

Esta aplicação de exemplo analisa e armazena cartões de feedback de clientes. Especificamente, ela atende à necessidade de um hotel fictício na cidade de Nova York. O hotel recebe feedback dos hóspedes em vários idiomas na forma de cartões de comentários físicos. Esse feedback é enviado para a aplicação por meio de um cliente web. Depois de fazer upload da imagem de um cartão de comentário, ocorrem as seguintes etapas:

- O texto é extraído da imagem usando o Amazon Textract.
- O Amazon Comprehend determina o sentimento do texto extraído e o idioma.
- O texto extraído é traduzido para o inglês com o Amazon Translate.
- O Amazon Polly sintetiza um arquivo de áudio do texto extraído.

A aplicação completa pode ser implantada com o AWS CDK. Para obter o código-fonte e as instruções de implantação, consulte o projeto em GitHub.

Serviços utilizados neste exemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Para obter uma lista completa dos Guias do desenvolvedor do SDK da AWS e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Chamar uma função do Lambda em um navegador

O exemplo de código a seguir mostra como chamar uma função do AWS Lambda em um navegador.

JavaScript

SDK para JavaScript (v2)

É possível criar uma aplicação baseada em navegador que usa uma função do AWS Lambda para atualizar uma tabela do Amazon DynamoDB com seleções de usuário.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Lambda

SDK para JavaScript (v3)

É possível criar uma aplicação baseada em navegador que usa uma função do AWS Lambda para atualizar uma tabela do Amazon DynamoDB com seleções de usuário. Essa aplicação usa o AWS SDK for JavaScript v3.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Lambda

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Como transformar dados para sua aplicação com o S3 Object Lambda

O exemplo de código a seguir mostra como transformar dados para sua aplicação com o S3 Object Lambda.

.NET

AWS SDK for .NET

Mostra como adicionar código personalizado a solicitações GET padrão do S3 para modificar o objeto solicitado e recuperado do S3 e possibilitar que o objeto atenda às necessidades do cliente ou aplicação solicitante.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços utilizados neste exemplo

- Lambda
- Amazon S3

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar o API Gateway para invocar uma função do Lambda

Os exemplos de código a seguir mostram como criar uma função do AWS Lambda invocada pelo Amazon API Gateway.

Java

SDK para Java 2.x

Mostra como criar uma função do AWS Lambda usando a API de runtime de Java do Lambda. Este exemplo invoca diferentes serviços da AWS para lidar com um caso de uso específico. Este exemplo mostra como criar uma função do Lambda invocada pelo Amazon API Gateway que verifica uma tabela do Amazon DynamoDB em busca de aniversários de trabalho e usa o Amazon Simple Notification Service (Amazon SNS) para enviar uma mensagem de texto aos seus funcionários que os parabeniza em sua data de aniversário de um ano.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK para JavaScript (v3)

Mostra como criar uma função do AWS Lambda usando a API de runtime de JavaScript do Lambda. Este exemplo invoca diferentes serviços da AWS para lidar com um caso de uso específico. Este exemplo mostra como criar uma função do Lambda invocada pelo Amazon API Gateway que verifica uma tabela do Amazon DynamoDB em busca de aniversários de trabalho e usa o Amazon Simple Notification Service (Amazon SNS) para enviar uma mensagem de texto aos seus funcionários que os parabeniza em sua data de aniversário de um ano.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Esse exemplo também está disponível no Guia do desenvolvedor do AWS SDK for JavaScript v3.

Serviços usados neste exemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Python

SDK para Python (Boto3).

Este exemplo mostra como criar e usar uma API REST do Amazon API Gateway cujo alvo é uma função do AWS Lambda. O manipulador do Lambda mostra como rotear com base em métodos HTTP; como obter dados da string de consulta, do cabeçalho e do corpo e como retornar uma resposta JSON.

- Implante uma função do Lambda.
- Crie uma API REST do API Gateway.
- Criar um recurso REST cujo alvo seja a função do Lambda.
- Conceda permissão para que o API Gateway possa invocar a função do Lambda.

- Use o pacote Requests para enviar solicitações à API REST.
- Limpe todos os recursos criados durante a demonstração.

Este exemplo é mais bem visualizado no GitHub. Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- API Gateway
- Lambda

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar Step Functions para invocar funções do Lambda

Os exemplos de código a seguir mostram como criar um máquina de estado do AWS Step Functions que invoca a funções do AWS Lambda em sequência.

Java

SDK para Java 2.x

Mostra como criar um AWS fluxo de trabalho sem servidor usando o AWS Step Functions e o AWS SDK for Java 2.x. Cada etapa do fluxo de trabalho é implementada usando uma função do AWS Lambda.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

JavaScript

SDK para JavaScript (v3)

Mostra como criar um AWS fluxo de trabalho sem servidor usando o AWS Step Functions e o AWS SDK for JavaScript. Cada etapa do fluxo de trabalho é implementada usando uma função do AWS Lambda.

O Lambda é um serviço computacional que permite executar código sem provisionar ou gerenciar servidores. O Step Functions é um serviço de orquestração sem servidor que permite combinar funções do Lambda e outros serviços da AWS para criar aplicações essenciais aos negócios.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Esse exemplo também está disponível no Guia do desenvolvedor do AWS SDK for JavaScript v3.

Serviços usados neste exemplo

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Usar eventos programados para invocar uma função do Lambda

Os exemplos de código a seguir mostram como criar uma função do AWS Lambda invocada por um evento programado do Amazon EventBridge.

Java

SDK para Java 2.x

Mostra como criar um evento programado do Amazon EventBridge que invoca uma função do AWS Lambda. Configure o EventBridge para usar uma expressão cron para programar

o momento em que a função do Lambda é invocada. Neste exemplo, você cria uma função do Lambda usando a API de runtime de Java do Lambda. Este exemplo invoca diferentes serviços da AWS para lidar com um caso de uso específico. Este exemplo mostra como criar uma aplicação que envia uma mensagem de texto móvel para seus funcionários que os parabeniza na data de aniversário de um ano.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços usados neste exemplo

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK para JavaScript (v3)

Mostra como criar um evento programado do Amazon EventBridge que invoca uma função do AWS Lambda. Configure o EventBridge para usar uma expressão cron para programar o momento em que a função do Lambda é invocada. Neste exemplo, você cria uma função do Lambda usando a API de runtime de JavaScript do Lambda. Este exemplo invoca diferentes serviços da AWS para lidar com um caso de uso específico. Este exemplo mostra como criar uma aplicação que envia uma mensagem de texto móvel para seus funcionários que os parabeniza na data de aniversário de um ano.

Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Esse exemplo também está disponível no Guia do desenvolvedor do AWS SDK for JavaScript v3.

Serviços usados neste exemplo

- DynamoDB
- EventBridge
- Lambda

- Amazon SNS

Python

SDK para Python (Boto3).

Este exemplo mostra como registrar uma função do AWS Lambda como o destino de um evento do Amazon EventBridge programado. O handler do Lambda grava uma mensagem amigável e os dados completos do evento no Amazon CloudWatch Logs para recuperação posterior.

- Implanta uma função do Lambda.
- Cria um evento agendado do EventBridge e faz da função do Lambda o destino.
- Concede permissão para que o EventBridge invoque uma função do Lambda.
- Imprime os dados mais recentes do CloudWatch Logs para mostrar o resultado das invocações agendadas.
- Limpa todos os recursos criados durante a demonstração.

Este exemplo é mais bem visualizado no GitHub. Para obter o código-fonte completo e instruções sobre como configurar e executar o exemplo, consulte o exemplo completo no GitHub.

Serviços utilizados neste exemplo

- CloudWatch Logs
- EventBridge
- Lambda

Para obter uma lista completa dos Guias do desenvolvedor do AWS SDK e exemplos de código, consulte Utilizar o Lambda com um AWS SDK. Este tópico também inclui informações sobre como começar e detalhes sobre versões anteriores do SDK.

Cotas Lambda

> **Important**
>
> Novas Contas da AWS reduziram as cotas de simultaneidade e de memória. A AWS aumenta essas cotas automaticamente com base na utilização.

Computação e armazenamento

O Lambda define cotas para a quantidade de recursos computacionais e de armazenamento que você pode usar para executar e armazenar funções. As cotas para execuções simultâneas e armazenamento são aplicadas por Região da AWS. As cotas de interface de rede elástica (ENI) se aplicam por nuvem privada virtual (VPC), qualquer que seja a região. As cotas a seguir podem ser aumentadas dos valores padrão. Para obter mais informações, consulte Solicitar um aumento da cota no Manual do usuário do Service Quotas.

Recurso	Cota padrão	Pode ser aumentado até
Execuções simultâneas	1.000	Dezenas de milhares
Armazenamento para funções carregadas (arquivos .zip) e camadas. Cada versão da função e versão da camada consome armazenamento. Para práticas recomendadas para o gerenciamento do armazenamento do seu código, consulte Monitoring Lambda code storage no Serverless Land.	75 GB	Terabytes
Armazenamento para funções definidas como imagens de contêiner Essas imagens são armazenadas no Amazon ECR.	Consulte Cotas de serviço do Amazon ECR.	

Recurso	Cota padrão	Pode ser aumentado até
Interfaces de rede elásticas por Virtual Private Cloud (VPC) **Note** Esta cota é compartilhada com outros serviços, como o Amazon Elastic File System (Amazon EFS). Consulte Cotas da Amazon VPC.	250	Milhares

Para obter detalhes sobre a simultaneidade e sobre como o Lambda dimensiona a simultaneidade da função em resposta ao tráfego, consulte Como entender a escalabilidade da função do Lambda.

Configuração, implantação e execução de funções

As cotas a seguir se aplicam à configuração de funções, às implantações e à execução. Exceto quando indicado, elas não podem ser alteradas.

Note

A documentação do Lambda, as mensagens de log e o console usam a abreviatura MB (em vez de MiB) para se referir a 1.024 KB.

Recurso	Cota
Alocação de memória da função	128 MB a 10.240 MB, em incrementos de 1 MB. Observação: o Lambda aloca capacidade da CPU na proporção da quantidade de memória configura

Recurso	Cota
	da. Você pode aumentar ou diminuir a memória e a potência da CPU alocada para a função usando a configuração Memória (MB). Com 1.769 MB, uma função tem o equivalente a uma vCPU.
Tempo limite da função	900 segundos (15 minutos)
Variáveis de ambiente da função	4 KB, para todas as variáveis de ambiente associadas à função, em agregado
Política baseada em recursos da função	20 KB
Função camadas	cinco camadas
Limite de escalabilidade de simultaneidade de funções	Para cada função, mil ambientes de execução a cada dez segundos
Carga da invocação (solicitação e resposta)	6 MB cada para solicitação e resposta (síncrona) 20 MB para cada resposta transmitida (Síncrono. O tamanho da carga útil para respostas transmitidas pode ser aumentado em relação aos valores padrão. Entre em contato com o AWS Support para saber mais.) 256 KB (assíncrona) 1 MB para o tamanho total combinado dos valores de linha e de cabeçalho da solicitação

Recurso	Cota
Largura de banda para respostas enviadas por streamig	Ilimitada para os primeiros 6 MB da resposta da função Para respostas maiores que 6 MB, 2 MBps para o restante da resposta
Tamanho dopacote de implantação (arquivo .zip)	50 MB (compactado, para upload direto) 250 MB (descompactado) Essa cota se aplica a todos os arquivos que você carrega, inclusive camadas e tempos de execução personalizados. 3 MB (editor de console)
Tamanho das configurações de imagem de contêiner	16 KB
Tamanho do pacote do código da imagem do contêiner	10 GB (tamanho máximo de imagem descompactada, incluindo todas as camadas)
Eventos de teste (editor de console)	10
Armazenamento do diretório do `/tmp`	Entre 512 MB e 10.240 MB, em incrementos de 1 MB
Descrições do arquivo	1,024
Processos de execução/threads	1,024

Solicitações da API do Lambda

As cotas a seguir estão associadas a solicitações de API do Lambda.

Recurso	Cota
Solicitações de invocação por função por região (síncronas)	Cada instância do ambiente de execução pode atender até dez solicitações por segundo. Em outras palavras, o limite total de invocação corresponde a dez vezes o limite de simultaneidade. Consulte Como entender a escalabilidade da função do Lambda.
Solicitações de invocação por função por região (assíncronas)	Cada instância do ambiente de execução pode atender a um número ilimitado de solicitações. Em outras palavras, o limite total de invocação é baseado somente na simultaneidade disponível para a função. Consulte Como entender a escalabilidade da função do Lambda.
Solicitações de invocação por alias ou versão de função (solicitações por segundo)	10 x simultaneidade provisionada alocada **Note** Essa cota se aplica somente às funções que usam simultaneidade provisionada.
Solicitações da API GetFunction	100 solicitações por segundo. Não pode ser aumentado.
Solicitações de API GetPolicy	15 solicitações por segundo. Não pode ser aumentado.

Recurso	Cota
Restante das solicitações da API do ambiente de gerenciamento (exclui solicitações de invocação, GetFunction e GetPolicy)	15 solicitações por segundo em todas as APIs (não 15 solicitações por segundo por API). Não pode ser aumentado.

Outros serviços

Cotas para outros serviços, como o AWS Identity and Access Management (IAM), Amazon CloudFront (Lambda @Edge) e Amazon Virtual Private Cloud (Amazon VPC) podem afetar as funções do Lambda. Para obter mais informações, consulte AWS service (Serviço da AWS) quotas, na Referência geral da Amazon Web Services, e Invocando o Lambda com eventos de outros serviços da AWS.

Histórico do documento

A tabela a seguir descreve as alterações importantes feitas no Guia do desenvolvedor do AWS Lambda desde maio de 2018. Para receber notificações sobre atualizações dessa documentação, inscreva-se no feed RSS.

Alteração	Descrição	Data
Compatibilidade com o SnapStart em novas regiões	O SnapStart do Lambda agora está disponível nas seguintes regiões: Europa (Espanha), Europa (Zurique), Ásia-Pacífico (Melbourne), Ásia-Pacífico (Hyderabad) e Oriente Médio (EAU).	12 de janeiro de 2024
Atualizações de política gerenciada do AWS	O Service Quotas atualizou uma política gerenciada existente da AWS (`AWSLambdaVPCAccessExecutionRole`).	5 de janeiro de 2024
Runtime do Python 3.12	O Lambda agora é compatível com o Python 3.12 como um runtime gerenciado e uma imagem base de contêiner. Para obter mais informações, consulte Python 3.12 runtime now available no AWS Lambda no AWS Compute Blog.	14 de dezembro de 2023
Runtime do Java 21	O Lambda agora é compatível com o Java 21 como um runtime gerenciado e uma imagem base de contêiner (`java21`).	16 de novembro de 2023

Runtime do Node.js 20.x	O Lambda agora é compatível com o Node.js 20 como um runtime gerenciado e uma imagem base de contêiner (`nodejs20.x`). Para obter mais informações, consulte Node.js 20.x runtime now available in AWS Lambda no Blog AWS Compute.	14 de novembro de 2023
Runtime provided.al2023	O Lambda agora é compatível com o Amazon Linux 2023 como um runtime gerenciado e uma imagem base de contêiner. Para obter mais informações, consulte Introducing the Amazon Linux 2023 runtime for AWS Lambda no Blog AWS Compute.	9 de novembro de 2023
Suporte a IPv6 para sub-redes de pilha dupla	O Lambda agora oferece suporte a tráfego IPv6 de saída para sub-redes de pilha dupla. Para obter mais informações, consulte suporte a IPv6.	12 de outubro de 2023

Testar funções e aplicações com tecnologia sem servidor	Aprenda sobre técnicas para depurar e automatizar testes de funções com tecnologia sem servidor na nuvem. Existe, agora, um capítulo de testes e recursos incluídos nas seções de linguagem Python e Typescript. Para obter detalhes, consulte Testar funções e aplicações com tecnologia sem servidor.	16 de junho de 2023
Runtime do Ruby 3.2	O Lambda agora é compatível com um novo runtime para o Ruby 3.2. Para obter mais informações, consulte Criar funções do Lambda com o Ruby.	7 de junho de 2023
Streaming de respostas	Agora, o Lambda oferece suporte ao streaming de respostas de funções. Para obter mais informações, consulte Configuração de uma função do Lambda para o streaming de respostas.	6 de abril de 2023
Métricas de invocação assíncrona	O Lambda libera métricas de invocação assíncrona. Para obter mais informações, consulte Métricas de invocação assíncrona.	9 de fevereiro de 2023

Controles de versão de runtime	O Lambda realizou o lançamento de novas versões do runtime que incluem atualizações de segurança , correções de bugs e novos recursos. Agora é possível controlar quando as funções são atualizadas para as novas versões do runtime. Para obter mais informações, consulte Lambda runtime updates (Atualizações para o runtime do Lambda).	23 de janeiro de 2023
Lambda SnapStart	Use o Lambda SnapStart para reduzir o tempo de startup das funções Java sem o provision amento de recursos adicionai s ou a implementação de otimizações de performan ce complexas. Para obter mais informações, consulte Aprimoramento da performan ce de inicialização com o Lambda SnapStart.	28 de novembro de 2022
Runtime do Node.js 18	O Lambda passou a oferecer suporte a um novo runtime para o Node.js 18. O Node.js 18 usa o Amazon Linux 2. Para obter detalhes, consulte Criar funções do Lambda com o Node.js.	18 de novembro de 2022

Chave de condição lambda:SourceFunctionArn	Para um recurso da AWS, a chave de condição `lambda:SourceFunctionArn` filtra o acesso ao recurso pelo ARN de uma função do Lambda. Para obter mais detalhes, consulte Trabalhar com credenciais do ambiente de execução do Lambda.	1° de julho de 2022
Runtime do Node.js 16	Agora o Lambda é compatível com um novo runtime para o Node.js 16. O Node.js 16 usa o Amazon Linux 2. Para obter detalhes, consulte Criar funções do Lambda com o Node.js.	11 de maio de 2022
URLs de função do Lambda	O Lambda agora é compatível com URLs de função, que são endpoints HTTP(S) dedicados para funções do Lambda. Para obter mais detalhes, consulte Lambda function URLs (URLs de função do Lambda).	6 de abril de 2022
Eventos de teste compartilhados no console do AWS Lambda	O Lambda passou a oferecer suporte ao compartilhamento de eventos de teste com outros usuários na mesma Conta da AWS. Para obter mais detalhes, consulte Como testar funções do Lambda no console.	16 de março de 2022

PrincipalOrgID em políticas baseadas em recursos	O Lambda agora é compatível com a concessão de permissões a uma organização no AWS Organizations. Para obter mais detalhes, consulte Uso de políticas baseadas em recursos para o AWS Lambda.	11 de março de 2022
Runtime do .NET 6	O Lambda agora é compatível com um novo runtime para o .NET 6. Para obter detalhes, consulte Runtimes do Lambda.	23 de fevereiro de 2022
Filtragem de eventos para fontes de eventos do Kinesis, DynamoDB e Amazon SQS	O Lambda agora é compatível com filtragem de eventos para fontes de eventos do Kinesis, DynamoDB e Amazon SQS. Para obter mais detalhes, consulte Filtragem de eventos do Lambda.	24 de novembro de 2021
Autenticação mTLS para fontes de eventos do Amazon MSK e do Apache Kafka autogerenciado	O Lambda agora oferece suporte a autenticação mTLS para fontes de eventos do Amazon MSK e do Apache Kafka autogerenciado. Para obter detalhes, consulte Usar o Lambda com o Amazon MSK.	19 de novembro de 2021

Lambda no Graviton2	O Lambda agora é compatível com o Graviton2 para funções que usam a arquitetura arm64. Para obter mais detalhes, consulte Arquiteturas do conjunto de instruções do Lambda.	29 de setembro de 2021
Runtime do Python 3.9	O Lambda é agora compatível com um novo runtime para o Python 3.9. Para obter detalhes, consulte Runtimes do Lambda.	16 de agosto de 2021
Novas versões de runtime para Node.js, Python e Java	Novas versões de runtime estão disponíveis para Node.js, Python e Java. Para obter detalhes, consulte Runtimes do Lambda.	21 de julho de 2021
Compatibilidade com RabbitMQ como uma fonte de eventos no Lambda	Agora, o Lambda é compatível com Amazon MQ for RabbitMQ como uma fonte de eventos. O Amazon MQ é um serviço gerenciado de agente de mensagem para o Apache ActiveMQ e o RabbitMQ que facilita a configuração e operação de agentes de mensagem na nuvem. Para obter detalhes, consulte Usar o Lambda com o Amazon MQ.	7 de julho de 2021

Autenticação SASL/PLAIN para Kafka autogerenciado no Lambda	Agora, o SASL/PLAIN é um mecanismo de autenticação compatível com fontes de eventos Kafka autogeren ciadas no Lambda. Os clientes que já usam o SASL/PLAI N em seu cluster Kafka autogerenciado podem usar facilmente o Lambda para criar aplicações de consumido r sem ter que modificar a maneira como elas se autenticam. Para obter mais detalhes, consulte .Usar o Lambda com Apache Kafka autogerenciado.	29 de junho de 2021
API de extensões do Lambda	Disponibilidade geral de extensões do Lambda. Use extensões para ampliar as funções do Lambda. É possível usar extensões fornecidas por parceiros do Lambda ou criar suas próprias extensões do Lambda. Para obter detalhes, consulte API de extensões do Lambda.	24 de maio de 2021
Nova experiência de console do Lambda	O console do Lambda foi reprojetado para melhorar a performance e a consistência.	2 de março de 2021

Runtime do Node.js 14	Agora o Lambda é compatível com um novo runtime para o Node.js 14. O Node.js 14 usa o Amazon Linux 2. Para obter detalhes, consulte Criar funções do Lambda com o Node.js.	27 de janeiro de 2021
Imagens de contêiner do Lambda	Agora o Lambda é compatível com funções definidas como imagens de contêiner. Você pode combinar a flexibilidade das ferramentas de contêineres com a agilidade e a simplicidade operacional do Lambda para criar aplicações. Para obter detalhes, consulte Usar imagens de contêiner com o Lambda.	1º de dezembro de 2020

Assinatura de código para funções do Lambda	O Lambda já é compatível com assinatura de código. Os administradores podem configurar funções do Lambda para aceitar apenas código assinado na implantação. O Lambda verifica as assinatur as para garantir que o código não seja alterado ou adulterad o. Além disso, o Lambda confirma que o código seja assinado por desenvolvedores confiáveis antes de aceitar a implantação. Para obter detalhes, consulte Configura r assinatura de código para o Lambda.	23 de novembro de 2020
Pré-visualização: API de logs de runtime do Lambda	Agora o Lambda oferece suporte à API Runtime Logs. As extensões do Lambda podem usar a API Runtime Logs para assinar transmiss ões de logs no ambiente de execução. Para obter detalhes, consulte API Runtime Logs do Lambda.	12 de novembro de 2020

Nova fonte de eventos para o Amazon MQ	Agora, o Lambda é compatível com o Amazon MQ como uma fonte de eventos. Use uma função do Lambda para processar registros do seu agente de mensagens do Amazon MQ. Para obter detalhes, consulte Usar o Lambda com o Amazon MQ.	5 de novembro de 2020
Pré-visualização: API de extensões do Lambda	Use extensões do Lambda para ampliar as funções do Lambda. É possível usar extensões fornecidas por parceiros do Lambda ou criar suas próprias extensões do Lambda. Para obter detalhes, consulte API de extensões do Lambda.	8 de outubro de 2020
Compatibilidade com Java 8 e runtimes personalizados no AL2	O Lambda já oferece suporte ao Java 8 e a runtimes personalizados no Amazon Linux 2. Para obter detalhes, consulte Runtimes do Lambda.	12 de agosto de 2020
Nova fonte de eventos para transmissão gerenciada da Amazon para Apache Kafka	Agora, o Lambda é compatível com o Amazon MSK como uma fonte de eventos. Use uma função do Lambda com o Amazon MSK para processar registros em um tópico do Kafka. Para obter detalhes, consulte Usar o Lambda com o Amazon MSK.	11 de agosto de 2020

Chaves de condição do IAM para configurações do Amazon VPC	Agora você pode usar chaves de condição específicas do Lambda para configurações da VPC. Por exemplo, você pode exigir que todas as funções em sua organização estejam conectadas a uma VPC. Você também pode especificar as sub-redes e os grupos de segurança que os usuários da função podem e não podem usar. Para obter detalhes, consulte Configurar a VPC para funções do IAM.	10 de agosto de 2020
Configurações de simultaneidade para consumidores de transmissões HTTP/2 do Kinesis	Agora é possível usar as seguintes configurações de simultaneidade para consumidores do Kinesis com distribuição avançada (transmissões HTTP/2): ParallelizationFactor, MaximumRetryAttempts, MaximumRecordAgeInSeconds, DestinationConfig e BisectBatchOnFunctionError. Para obter detalhes, consulte Usar o AWS Lambda com o Amazon Kinesis.	7 de julho de 2020

Janela de lote para consumidores de transmissões HTTP/2 do Kinesis	Agora é possível configurar uma janela de lote (MaximumBatchingWindowInSeconds) para transmissões HTTP/2. O Lambda lê registros da transmissão até coletar um lote inteiro, ou até que a janela de lote expire. Para obter detalhes, consulte Usar o AWS Lambda com o Amazon Kinesis.	18 de junho de 2020
Compatibilidade com sistemas de arquivos do Amazon EFS	Agora é possível conectar um sistema de arquivos do Amazon EFS às funções do Lambda para acesso compartilhado aos arquivos de rede. Para obter detalhes, consulte Configurar o acesso ao sistema de arquivos para funções do Lambda.	16 de junho de 2020
Aplicações de exemplo do AWS CDK no console do Lambda	O console do Lambda agora inclui aplicações de exemplo que usam o AWS Cloud Development Kit (AWS CDK) para TypeScript. O AWS CDK é um framework que permite definir os recursos de sua aplicação em TypeScript, Python, Java ou .NET.	1 de junho de 2020

Compatibilidade com runtime do .NET Core 3.1.0 no AWS Lambda	Agora, o AWS Lambda oferece suporte para o runtime do .NET Core 3.1.0. Para obter detalhes, consulte CLI do .NET Core.	31 de março de 2020
Compatibilidade com APIs HTTP do API Gateway	Documentação atualizada e expandida para uso do Lambda com o API Gateway, incluindo suporte para APIs HTTP. Adição de um aplicativo de exemplo que cria uma API e uma função com o AWS CloudFormation. Para obter detalhes, consulte Usar o Lambda com o Amazon API Gateway.	23 de março de 2020
Ruby 2.7	Um novo runtime está disponível para Ruby 2.7, ruby2.7, que é o primeiro runtime Ruby a usar o Amazon Linux 2. Para obter detalhes, consulte Criar funções do Lambda com o Ruby.	19 de fevereiro de 2020

Métricas de simultaneidade	Agora o Lambda gera relatórios sobre a métrica `ConcurrentExecutions` para todas as funções, aliases e versões. É possível visualizar um gráfico para essa métrica na página de monitoramento da função. Anteriormente, a métrica `ConcurrentExecutions` só era relatada no nível de conta e para funções que usam simultaneidade reservada. Para obter detalhes, consulte Métricas de função do AWS Lambda.	18 de fevereiro de 2020

Atualização de estados de função	Os estados de função agora são impostos para todas as funções por padrão. Quando você conecta uma função a uma VPC, o Lambda cria interfaces de rede elásticas compartilhadas. Isso permite que sua função aumente sem criar interfaces de rede adicionais. Durante esse período, não será possível executar operações adicionais na função, incluindo a atualização de sua configuração e a publicação de versões. Em alguns casos, a invocação também será afetada. Detalhes sobre o estado atual de uma função estão disponíveis na API do Lambda. Essa atualização está sendo lançada em fases. Para obter detalhes, consulte Updated Lambda states lifecycle for VPC networking no blog de computação da AWS. Para obter mais informações sobre estados, consulte Estados de função do AWS Lambda.	24 de janeiro de 2020

Atualizações de saída de API de configuração de função	Foram adicionados códigos de motivo para StateReasonCode (InvalidSubnet, InvalidSecurityGroup) and LastUpdateStatusReasonCode (SubnetOutOfIPAddresses, InvalidSubnet, InvalidSecurityGroup) para funções que se conectam a uma VPC. Para obter mais informações sobre estados, consulte Estados de função do AWS Lambda.	20 de janeiro de 2020
Simultaneidade provisionada	Agora é possível alocar a simultaneidade provisionada para um alias ou uma versão de função. A simultaneidade provisionada permite que uma função seja dimensionada sem flutuações na latência. Para obter detalhes, consulte Gerenciar a simultaneidade para uma função do Lambda .	3 de dezembro de 2019

Criar um proxy de banco de dados	Agora é possível usar o console do Lambda para criar um proxy de banco de dados para uma função do Lambda. Um proxy de banco de dados permite que uma função atinja altos níveis de simultaneidade sem esgotar as conexões de banco de dados. Para obter detalhes, consulte Configurar o acesso ao banco de dados para uma função do Lambda.	3 de dezembro de 2019
Compatibilidade com percentis para a métrica de duração	Agora é possível filtrar a métrica de duração com base em percentis. Para obter detalhes, consulte Métricas do AWS Lambda.	26 de novembro de 2019
Maior simultaneidade para fontes de eventos de transmissão	Uma nova opção para mapeamentos de fontes de eventos de transmissão do DynamoDB e de transmissão do Kinesis permite processar mais de um lote por vez de cada fragmento. Ao aumentar o número de lotes simultâneos por estilhaço, a simultaneidade da sua função pode ser até 10 vezes o número de estilhaços no fluxo. Para obter detalhes, consulte Mapeamentos de fonte de eventos do Lambda.	25 de novembro de 2019

Estados de função	Quando uma função é criada ou atualizada, ela entra em um estado pendente enquanto o Lambda provisiona recursos para oferecer suporte a ela. Se você conectar sua função a uma VPC, o Lambda poderá criar uma interface de rede elástica compartil hada imediatamente, em vez de criar interfaces de rede quando a função for invocada. Isso resulta em melhor performance para funções conectadas à VPC, mas pode exigir uma atualizaç ão para sua automação. Para obter detalhes, consulte Estados de função do AWS Lambda.	25 de novembro de 2019
Opções de tratamento de erros para invocação assíncrona	Novas opções de configura ção estão disponíveis para invocação assíncrona. É possível configurar o Lambda para limitar novas tentativas e definir uma idade máxima do evento. Para obter detalhes, consulte Configura r o tratamento de erros para invocação assíncrona.	25 de novembro de 2019

Tratamento de erros para fontes de eventos de transmissão	Novas opções de configuração estão disponíveis para mapeamentos de origem de evento que leem de fluxos. É possível configurar mapeamentos de fontes de eventos de transmissão do DynamoDB e transmissão do Kinesis para limitar as novas tentativas e definir uma idade máxima para os registros. Quando ocorrem erros, você pode configurar o mapeamento de origem de eventos para dividir lotes antes de tentar novamente e para enviar registros de invocação de lotes com falha para uma fila ou tópico. Para obter detalhes, consulte Mapeamentos de fonte de eventos do Lambda.	25 de novembro de 2019

Destinos para invocação assíncrona	Agora é possível configurar o Lambda para enviar registros de invocações assíncronas a outro serviço. Os registros de invocação contêm detalhes sobre o evento, o contexto e a resposta da função. Você pode enviar registros de invocação para uma fila do SQS, um tópico do SNS, uma função do Lambda ou um barramento de eventos EventBridge. Para obter detalhes, consulte Configurar destinos para invocação assíncrona.	25 de novembro de 2019
Novos runtimes para Node.js, Python e Java	Novos runtimes estão disponíveis para Node.js 12, Python 3.8 e Java 11. Para obter detalhes, consulte Runtimes do Lambda.	18 de novembro de 2019
Compatibilidade com fila FIFO para fontes de eventos do Amazon SQS	Agora é possível criar um mapeamento de origem de eventos que leia a partir de uma fila FIFO (primeiro a entrar, primeiro a sair). Anteriormente, havia suporte apenas para filas padrão. Para obter detalhes, consulte Usar o Lambda com o Amazon SQS.	18 de novembro de 2019

Criar aplicações no console do Lambda	A criação de aplicações no console do Lambda já está disponível ao público em geral. Para obter instruções, consulte Gerenciar aplicações no console do Lambda.	31 de outubro de 2019
Criar aplicações no console do Lambda (beta)	Agora você pode criar uma aplicação do Lambda com um pipeline de entrega contínua integrado no console do Lambda. O console fornece aplicativos de exemplo que você pode usar como ponto de partida para seu próprio projeto. Escolha entre AWS CodeCommit e GitHub para controle de origem. Cada vez que você envia alterações para o repositório, o pipeline incluído faz sua compilaçã o e implantação automatic amente. Para obter instruções, consulte Gerenciar aplicações no console do Lambda.	3 de outubro de 2019

Melhorias de performance para funções conectadas a VPC	Agora o Lambda usa um novo tipo de interface de rede elástica que é compartilhada por todas as funções em uma sub-rede da nuvem privada virtual (VPC). Quando você conecta uma função a uma VPC, o Lambda cria uma interface de rede para cada combinação de grupo de segurança e sub-rede escolhida. Quando as interfaces de rede compartilhadas estão disponíveis, a função não precisa mais criar interfaces de rede adicionais, já que ela é ampliada. Isso melhora significativamente os tempos de startup. Para obter detalhes, consulte Configurar uma função do Lambda para acessar recursos em uma VPC.	3 de setembro de 2019

Configurações de lote de transmissão	Agora você pode configurar uma janela de lote para mapeamentos de fontes de eventos do Amazon DynamoDB e do Amazon Kinesis. Configure uma janela de lote de até cinco minutos para armazenar em buffer os registros de entrada até que um lote completo esteja disponível. Isso reduz o número de vezes que sua função é invocada quando o fluxo está menos ativo.	29 de agosto de 2019
Integração de insights do CloudWatch Logs	A página de monitoramento no console do Lambda agora inclui relatórios do Amazon CloudWatch Logs Insights. Para obter detalhes, consulte Monitorar funções no console do AWS Lambda.	18 de junho de 2019
Amazon Linux 2018.03	O ambiente de execução do Lambda está sendo atualizado para usar o Amazon Linux 2018.03. Para obter detalhes, consulte Ambiente de execução.	21 de maio de 2019

Node.js 10	Um novo runtime está disponível para Node.js 10, nodejs10.x. Esse runtime usa o Node.js 10.15 e será atualizado com a última versão de ponto do Node.js 10 periodicamente. O Node.js 10 também é o primeiro runtime para usar o Amazon Linux 2. Para obter detalhes, consulte Criar funções do Lambda com o Node.js.	13 de maio de 2019
GetLayerVersionByArn API	Use a API GetLayerVersionByArn para fazer download das informações sobre a versão da camada com o ARN da versão como entrada. Comparada como GetLayerVersion, GetLayerVersionByArn permite que você use ARN diretamente, em vez de analisá-la para obter o nome da camada e o número da versão.	25 de abril de 2019
Ruby	Agora o AWS Lambda oferece suporte ao Ruby 2.5 com um novo runtime. Para obter detalhes, consulte Criar funções do Lambda com o Ruby.	29 de novembro de 2018

Camadas	Com camadas do Lambda, é possível empacotar e implantar bibliotecas, runtimes personalizados e outras dependências separadam ente do seu código da função. Compartilhe camadas com suas outras contas ou com o mundo inteiro. Para obter detalhes, consulte Camadas do Lambda.	29 de novembro de 2018
Runtimes personalizados	Construa um runtime personali zado para executar funções do Lambda em sua linguagem de programação de preferência. Para obter detalhes, consulte Runtimes personalizados do Lambda.	29 de novembro de 2018
Acionadores do Application Load Balancer	O Elastic Load Balancing é compatível com funções do Lambda como destino para um balanceador de carga da aplicação. Para obter detalhes, consulte Usar o Lambda com balanceadores de carga da aplicação.	29 de novembro de 2018

Usar consumidores de transmissão HTTP/2 do Kinesis como acionador	Você pode usar os consumidores de streaming de dados do Kinesis HTTP/2 para enviar eventos para o AWS Lambda. Os consumidores de streaming dedicam o throughput de leitura de cada estilhaço no seu streaming de dados e usam o HTTP/2 para minimizar a latência. Para obter detalhes, consulte Usar o Lambda com o Kinesis.	19 de novembro de 2018
Python 3.7	O AWS Lambda agora oferece suporte ao Python 3.7 com um novo runtime. Para obter mais informações, consulte Criar funções do Lambda com Python.	19 de novembro de 2018
Aumento do limite de carga útil de invocação de função assíncrona	O tamanho máximo de carga útil para invocações assíncronas aumentou de 128 KB para 256 KB, o que corresponde ao tamanho máximo da mensagem de um acionador do Amazon SNS. Para obter detalhes, consulte Cotas do Lambda.	16 de novembro de 2018
Região GovCloud (Leste dos EUA) da AWS	Agora, o AWS Lambda está disponível na região GovCloud (EUA-Leste) da AWS.	12 de novembro de 2018

Transferência dos tópicos do AWS SAM para um guia do desenvolvedor separado	Vários tópicos estavam concentrados na criação de aplicativos sem servidor usando o AWS Serverless Application Model (AWS SAM). Esses tópicos foram movidos para o Guia do desenvolvedor do AWS Serverless Application Model.	25 de outubro de 2018
Visualizar aplicações do Lambda no console	Você pode visualizar o status de suas aplicações do Lambda na página Applications (Aplicações) no console do Lambda. Essa página mostra o status da pilha do AWS CloudFormation. Ela inclui links para páginas nas quais você pode visualizar mais informações sobre os recursos na pilha. Você também pode visualizar métricas agregadas para o aplicativo e criar painéis de monitoramento personalizados.	11 de outubro de 2018
Limite de tempo limite de execução de função	Para permitir funções de longa duração, o tempo limite máximo de execução configurável aumentou de 5 minutos para 15 minutos. Para obter detalhes, consulte Limites do Lambda.	10 de outubro de 2018

Compatibilidade com a linguagem do PowerShell Core no AWS Lambda	Agora, o AWS Lambda oferece suporte à linguagem do PowerShell Core. Para obter mais informações, consulte Modelo de programação para criação de funções do Lambda no PowerShell.	11 de setembro de 2018
Compatibilidade com runtime do .NET Core 2.1.0 no AWS Lambda	Agora, o AWS Lambda oferece suporte para o runtime do .NET Core 2.1.0. Para obter mais informações, consulte CLI do .NET Core.	9 de julho de 2018
Atualizações agora disponíveis em RSS	Agora você pode assinar um feed RSS para seguir as versões deste guia.	5 de julho de 2018
Compatibilidade com o Amazon SQS como fonte de eventos	Agora o AWS Lambda oferece suporte ao Amazon Simple Queue Service (Amazon SQS) como uma fonte de eventos. Para obter mais informações, consulte Invocar funções do Lambda.	28 de junho de 2018
Região da China (Ningxia)	Agora o AWS Lambda está disponível na região da China (Ningxia) Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	28 de junho de 2018

Atualizações anteriores

A tabela a seguir descreve as alterações importantes em cada versão do Guia do desenvolvedor do AWS Lambda antes de junho de 2018.

Alteração	Descrição	Data
Suporte ao runtime para o Node.js 8.10	Agora, o AWS Lambda oferece suporte ao runtime do Node.js versão 8.10. Para ter mais informações, consulte Criar funções do Lambda com Node.js.	2 de abril de 2018
IDs de revisão de função e de alias	O AWS Lambda agora oferece suporte aos IDs em suas versões e alias de função. Você pode usar esses IDs para monitorar e aplicar atualizações condicionais ao atualizar a versão de sua função ou os recursos de alias.	25 de janeiro de 2018
Suporte ao runtime para o Go e o .NET 2.0	O AWS Lambda adicionou suporte ao runtime para o Go e o .NET 2.0. Para obter mais informações, consulte Criar funções do Lambda com Go e Construir funções do Lambda com C#.	15 de janeiro de 2018
Novo design do console	O AWS Lambda introduziu um novo console do Lambda para simplificar sua experiência e adicionou um editor de código do Cloud9 para melhorar sua capacidade de depurar e revisar seu código de função. Para ter mais informações, consulte Editar código usando o editor do console do Lambda.	30 de novembro de 2017
Configurar os limites de simultaneidade nas funções individuais	O AWS Lambda agora oferece suporte à definição de limites de simultaneidade nas funções individuais. Para ter mais informações, consulte Configurar a simultaneidade reservada para uma função.	30 de novembro de 2017
Mudar o tráfego com aliases	O AWS Lambda agora oferece suporte à mudança de tráfego com aliases. Para ter mais informações, consulte Criar implantações contínuas para funções do Lambda.	28 de novembro de 2017

Alteração	Descrição	Data
Implantação gradual de código	Agora o AWS Lambda oferece suporte para a implantação segura de novas versões da sua função do Lambda utilizando a implantação de código. Para obter mais informações, consulte Implantação gradual de código.	28 de novembro de 2017
Região China (Pequim)	Agora o AWS Lambda está disponível na região da China (Pequim). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	9 de novembro de 2017
Introdução ao SAM Local	O AWS Lambda apresenta o SAM Local (conhecido agora como CLI do SAM), uma ferramenta da AWS CLI que oferece um ambiente para desenvolver, testar e analisar as aplicações sem servidor localmente antes de fazer upload delas no runtime do Lambda. Para obter mais informações, consulte Teste e depuração de aplicativos sem servidor.	11 de agosto de 2017
Região Canadá (Central)	Agora o AWS Lambda está disponível na região Canadá (Central). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	22 de junho de 2017
Região América do Sul (São Paulo)	O AWS Lambda já está disponível na região América do Sul (São Paulo). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	6 de junho de 2017
Suporte do AWS Lambda para AWS X-Ray.	O Lambda apresenta suporte ao X-Ray, o que permite detectar, analisar e otimizar problemas de performance com suas aplicações do Lambda. Para ter mais informações, consulte Visualizar as invocações da função do Lambda usando o AWS X-Ray.	19 de abril de 2017

Alteração	Descrição	Data
Região Ásia-Pacífico (Mumbai)	O AWS Lambda já está disponível na região da Ásia-Pacífico (Mumbai). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	28 de março de 2017
O AWS Lambda agora oferece suporte ao runtime do Node.js v6.10	O AWS Lambda adicionou suporte ao runtime do Node.js v6.10. Para ter mais informações, consulte Criar funções do Lambda com Node.js.	22 de março de 2017
Região Europa (Londres)	Agora o AWS Lambda está disponível na região Europa (Londres). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	1 de fevereiro de 2017
Suporte do AWS Lambda ao runtime do .NET, o Lambda@Edge (versão prévia), Dead Letter Queues e implantação automatizada de aplicativos sem servidor.	O AWS Lambda adicionou suporte ao C#. Para ter mais informações, consulte Construir funções do Lambda com C#. O Lambda@Edge permite que você execute funções do Lambda em todos os pontos de presença da AWS em resposta a eventos do CloudFront. Para ter mais informações, consulte Usando AWS Lambda com o CloudFront Lambda @Edge.	3 de dezembro de 2016
O AWS Lambda adiciona o Amazon Lex como uma fonte de eventos compatível.	Usando o Lambda e o Amazon Lex, você pode construir rapidamente bots de bate-papo para vários serviços como o Slack e o Facebook. Para ter mais informações, consulte O uso doAWS LambdaCom o Amazon Lex.	30 de novembro de 2016
Região Oeste dos EUA (Norte da Califórnia)	O AWS Lambda já está disponível na região Oeste dos EUA (Norte da Califórnia). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	21 de novembro de 2016

Alteração	Descrição	Data
Introduzido o AWS SAM para criação e implantação de aplicações baseadas no Lambda e usando variáveis de ambiente para definições de configuração da função do Lambda.	AWS SAM: agora você pode usar o AWS SAM para definir a sintaxe para expressar recursos dentro de uma aplicação sem servidor. Para implantar o aplicativo, basta especificar os recursos de que você precisa como parte de seu aplicativo, junto com suas políticas de permissões associadas em um arquivo de modelo do AWS CloudFormation (escrito em JSON ou YAML), empacotar os artefatos de sua implantação e implantar o modelo. Para ter mais informações, consulte Aplicações do AWS Lambda. Variáveis de ambiente: você pode usar variáveis de ambiente para especificar as definições de configuração para sua função do Lambda fora do código da função. Para ter mais informações, consulte Usar variáveis de ambiente no Lambda para configurar valores no código.	18 de novembro de 2016
Região Ásia-Pacífico (Seul)	O AWS Lambda já está disponível na região da Ásia-Pacífico (Seul). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	29 de agosto de 2016
Região Ásia-Pacífico (Sydney)	O Lambda já está disponível na região Ásia-Pacífico (Sydney). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	23 de junho de 2016
Atualizações no console do Lambda	O console do Lambda foi atualizado para simplificar o processo de criação de função.	23 de junho de 2016
O AWS Lambda agora oferece suporte ao runtime do Node.js v4.3	O AWS Lambda adicionou suporte ao runtime do Node.js v4.3. Para ter mais informações, consulte Criar funções do Lambda com Node.js.	07 de abril de 2016

Alteração	Descrição	Data
Região Europa (Frankfurt)	O Lambda já está disponível na região Europa (Frankfurt). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	14 de março de 2016
Suporte à VPC	Agora você pode configurar uma função do Lambda para acessar recursos em sua VPC. Para ter mais informaçõ es, consulte Conceder acesso a funções do Lambda para recursos em uma Amazon VPC.	11 de fevereiro de 2016
O runtime do Lambda foi atualizado.	O ambiente de execução foi atualizado.	4 de novembro de 2015

Alteração	Descrição	Data
Suporte ao versionamento, Python para desenvolvimento de código para funções do Lambda, eventos programados e aumento do runtime	Agora você pode desenvolver o código de sua função do Lambda usando o Python. Para ter mais informações, consulte Criar funções do Lambda com Python. Versionamento: você pode manter uma ou mais versões de sua função do Lambda. O versionamento permite que você controle qual função de Lambda é executada em diferentes ambientes (por exemplo, desenvolvimento, teste ou produção). Para ter mais informações, consulte Versões da função do Lambda. Eventos programados: você também pode configurar o Lambda para invocar seu código em base regular e programada usando o console do Lambda. Você pode especificar uma taxa fixa (número de horas, dias ou semanas) ou especificar uma expressão Cron. Para ter mais informações, consulte Usar o Lambda com o Agendador do Amazon EventBridge. Aumento no runtime: agora você pode configurar suas funções do Lambda para executar por até cinco minutos, permitindo funções em execução por mais tempo, como ingestão de dados de grande volume e trabalhos de processamento.	08 de outubro de 2015
Suporte ao DynamoDB Streams	O DynamoDB Streams já está disponível e você pode usá-lo em todas as regiões onde o DynamoDB estiver disponível. Você pode habilitar o DynamoDB Streams para sua tabela e usar uma função de Lambda como um trigger para a tabela. Os triggers são ações personalizadas executadas em resposta a atualizações feitas na tabela do DynamoDB. Para ver uma demonstração de exemplo, consulte Tutorial: Usar o AWS Lambda com o Amazon DynamoDB Streams.	14 de julho de 2015

Alteração	Descrição	Data
O Lambda agora oferece suporte para invocar funções do Lambda com clientes compatíveis com REST.	Até agora, para invocar a sua função do Lambda na Web, dispositivo móvel ou aplicação IoT, você precisava de AWS SDKs (por exemplo, AWS SDK for Java, AWS SDK for Android ou AWS SDK for iOS). Agora, o Lambda oferece suporte à invocação de uma função do Lambda com clientes compatíveis com REST por meio de uma API personalizada que você pode criar usando o Amazon API Gateway. Você pode enviar solicitações para a URL de endpoint de sua função do Lambda. Você pode configura r a segurança no endpoint para permitir acesso aberto, utilizar o AWS Identity and Access Management (IAM) para autorizar acesso ou usar chaves de API para mensurar o acesso de outras pessoas às suas funções do Lambda. Para ver um exemplo de exercício de Conceitos básicos, consulte Invocar uma função do Lambda usando um endpoint do Amazon API Gateway. Para obter mais informações sobre o Amazon API Gateway, consultehttps://aws.amazon.com/api-gateway/.	09 de julho de 2015
O console do Lambda agora oferece esquemas para criar e testar funções do Lambda facilmente.	O console do Lambda fornece um conjunto de esquemas. Cada esquema fornece uma configuração de exemplo de fonte de evento e o código de exemplo para a função do Lambda que você pode usar para criar aplicativos baseados em Lambda facilmente. Todos os exercícios de conceitos básicos do Lambda agora usam os esquemas. Para ter mais informações, consulte Conceitos básicos do Lambda.	09 de julho de 2015
O Lambda agora oferece suporte a Java para criar suas funções do Lambda.	Agora você pode criar código do Lambda em Java. Para ter mais informações, consulte Construir funções do Lambda com Java.	15 de junho de 2015

Alteração	Descrição	Data
O Lambda agora oferece suporte à especificação de um objeto do Amazon S3, como a função .zip, ao criar ou atualizar uma função do Lambda.	Você pode fazer upload de um pacote de implantação de função Lambda (arquivo .zip) para um bucket do Amazon S3 na mesma região em que você deseja criar uma função do Lambda. Em seguida, você pode especificar o nome do bucket e o nome da chave do objeto ao criar ou atualizar uma função do Lambda.	28 de maio de 2015
O Lambda agora está disponível com suporte adicional para backends móveis	O Lambda agora está disponível para uso em produção. A versão também apresenta novos recursos que tornam ainda mais fácil criar backends para celular, tablet e Internet das Coisas (IoT) usando o Lambda que é dimensionado automaticamente, sem provisionamento ou gerenciamento da infraestrutura. Agora, o Lambda oferece suporte a eventos em tempo real (síncronos) e com eventos assíncronos. Recursos adicionais incluem configuração e gerenciamento mais fáceis de origens de eventos. O modelo de permissões e o modelo de programação foram simplificados pela introdução de políticas de recursos para suas funções Lambda. A documentação foi atualizada de maneira adequada. Para obter informações, consulte os seguintes tópicos: Conceitos básicos do Lambda AWS Lambda	9 de abril de 2015
Versão de visualização	Versão de visualização do (Developer Guide) Guia do desenvolvedor do AWS Lambda.	13 de novembro de 2014